Decentralized k-Clique Matching

Chmielowiec, A.

2014

document version
Publisher's PDF, also known as Version of record

Decentralized k-Clique Matching

Anna Chmielowiec

ISBN 978-94-6259-266-7

Cover design by Mind Design, Amsterdam
http://minddesign.info
Printed and bound by Ipskamp Drukkers, Enschede
http://www.ipskampdrukkers.nl

SIKS Dissertation Series No. 2014-28
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

The research reported in this thesis has been conducted under the auspices of Network Institute of the Vrije University Amsterdam.

The research reported in this thesis has been conducted at the Department of Computer Science of the Vrije Universiteit Amsterdam.

VRIJE UNIVERSITEIT

DECENTRALIZED k-CLIQUE MATCHING

ACADEMISCH PROEFSCHRIFT

ter verkrijging van de graad Doctor aan
de Vrije Universiteit Amsterdam,
op gezag van de rector magnificus
prof.dr. F.A. van der Duyn Schouten,
in het openbaar te verdedigen
ten overstaan van de promotiecommissie
van de Faculteit der Exacte Wetenschappen
op donderdag 18 september 2014 om 11.45 uur
in de aula van de universiteit,
De Boelelaan 1105

door

ANNA AGNIESZKA CHMIELOWIEC

geboren te Warszawa, Polen

promotor: prof.dr.ir. M.R. van Steen
copromotor: dr. S. Voulgaris

Moim Rodzicom

CONTENTS

List of Figures

List of Tables

ACKNOWLEDGEMENTS

There is only one author of this dissertation but do not let that deceive you. This work would not have been possible without the valuable advice, moral support, and love of many people to whom I am truly grateful.

First and foremost, I would like to thank my supervisor, Maarten van Steen, for his steady support and patience during all the years of my Ph.D. studies. Our meetings, when we evaluated my recent research progress (aka. graphs, graphs, and more graphs) and brainstormed new ideas, had almost magical power of restoring my enthusiasm and motivation after weeks of what would have appeared to be useless work. I also really cherished the meetings when our discussions veered away from the research domain into the realm of life. As it turns out, Maarten, you were not only supervisor of my Ph.D. research but also became my life mentor. My huge thanks go to my co-supervisor, Spyros Voulgaris, who brought a more practical outlook on my research and made sure that I do not stay in the state of vague ideas and hand-waving for too long but instead that I keep producing tangible, cohesive, and comprehensible output. I am also very grateful to Jaap Gordijn and Guillaume Pierre, who supervised my work along with Maarten at the beginning of my Ph.D. studies. Thank you for your valuable advice on conducting research when I still had only a vague idea what a challenging task I was taking on.

I thank the members of my Ph.D. committee, Roberto Baldoni, Dick Epema, Wan Fokkink, and Ivar Vermeulen for their time, and valuable comments which helped me improve the text of this dissertation bringing it to this final state.

I would like to heartily thank my friends in Amsterdam for providing me with moral support and a great deal of pleasant distractions during my PhD years. I fear I will forget somebody if I try to name all of you, but you know who you are. Yet there are a few exceptions I would like to make, as three of you deserve a special mention. Asia, I always admired how devoted you were towards your research

and I tried to emulate you but could never equal. Yet, what is more important, you have been my best friend throughout all the years of my Ph.D. studies and you still are. Michaś, you are one of the kindest, open-hearted persons I have ever met; the warmth and optimism emanates from you and it makes me so happy that I can both laugh with you at tiny silly things and share with you really tough life problems. Konrad, thank you for our long discussions; I have never had so much pleasure of disagreeing with anyone else.

I must also mentioned here my girlfriends from MIMUW: Danka, Ela, Kasia, Marianna, and Marta. Moving abroad is exciting but also makes one realize how much there is to miss about one's country and especially one's hometown. Yet, with you girls just after a few hours spent over coffee I always felt as if I had never left Warsaw.

My very special thanks go to Marcin for his love, for always being there for me, for making me laugh whenever I was a bundle of nerves with a very unhappy face, and for all our big and small adventures.

Last but not least, I thank my parents for their love, support, and encouragement throughout my endeavor. You were my first teachers of life. You were the ones who instilled in me how important it is to keep on learning new things. I love you more than I could ever express. This dissertation is dedicated to you, Mom and Dad.

Anna Chmielowiec

San Mateo, CA, July 2014

CHAPTER 1

Introduction

1.1. PROBLEM DEFINITION

The ubiquity of the Web is rapidly and continuously progressing. The Internet grabs hold of increasingly more parts of our life, changing inadvertently the ways that we work, communicate and spend our free time. To a large extent, this process is driven by services and information goods that are moving en masse to the online world. Nowadays, almost any imaginable service can be found, acquired or accessed on the Web. First of all there are classic Internet services and products such as web or email hosting, office applications or games. Then, there are services that were once only analog and now are fully digitized, such as Internet radios, movie rentals and streaming, VoIP communication services, or online courses, as well as electronic versions of information goods, such as ebooks or digital versions of newspapers. Moreover, there is an abundance of online services that act as front line for acquiring other goods and services, such as online retailers, plane and hotel booking services, and event ticket sellers. Finally, we should not forget to mention the plethora of social networking and online community services. The list could go on forever, as services and digital goods continue to thrive online.

Taking a closer look at the services we mentioned, we recognize that many of them are actually composed of other simpler services. For example, social networks consist of e-mail, instant messaging, and often photo sharing. Similarly, online retailers combine ordering of goods with payment and delivery services. On the other hand, many services can be further combined together to create more complex services. For instance, hotel, airplane and events booking can be mashed up together into a full-fledged vacation planning service. Digital goods can also be combined together, just as various office applications for editing documents, preparing presentations and creating spreadsheets are offered as an office suite applications or when games are sold in bundles. Moreover, we also notice that

such compositions are characterized by a varying level of integration between their components, with more tightly coupled social networks to loosely coupled game bundles.

Currently, almost any such composition of services or goods is done manually and on a case by case basis, which can be time consuming. Yet, time is precisely the one resource that providers do not have. On one hand, they have to constantly keep up with changing consumer needs and preferences. On the other hand, as the number of online services and goods explodes, the competition between them becomes more and more fierce. As a result, providers feel pressured to act more dynamically, and also more strategically. One thing that can help providers to be more agile is inventing novel solutions that can be relatively easily created by composing or bundling together already existing services or goods. Moreover, they do not have to rely only on the availability of their own products. They can take advantage of the abundance of services and goods offered by other providers, and venture into multilateral collaborations, in which one product is co-created by multiple companies. This way, the created product can benefit from the expertise of each of the participating providers.

With this approach, the very first challenge a provider has to face is finding the most suitable services and goods for composition. This task has never been an easy one, but in the realm of online services it ironically became even harder. First of all, the choice is vast as the very nature of online goods and services makes it relatively easy from a technical perspective to combine them in endless configurations. For example, in many cases a simple bundle of a few services or goods (e.g. a bundle of game applications or ebooks that do not require any integration between them) can be an attractive offer to customers. Secondly, providers are no longer restricted by their geographical location, as with current communication technology it is possible for any provider to team up with companies from all over the world. Finally, if the combinations of more than just two services/goods are to be considered, the decision process becomes even more complex, because extending the offer by one additional good or service increases the number of possible offers exponentially. Still, the importance of making the best possible choice cannot be neglected, as it can play a decisive role in how successful the bundle will be.

Therefore, providers might be interested in ways to expedite the task. We envisage that a possible solution would be to establish a system capable of finding promising service combinations on behalf of providers. Any provider would be able to enter their service into such a system by supplying the service's profile. Moreover, such a profile could be further augmented with additional information gathered from other sources. For example, consumers opinions about the service and service's provider can be taken into account. Those can be used, for exam-

ple, to create *brand images* (a quantitative representations of associations held by consumers between a brand and selected qualitative attributes). Those brand images can in turn be used to establish whether the combination of two services from different brands will be favorably received by consumers. We discuss brand images and how a service-matching system can take advantage of them in Chapter 5. What is crucial here is that consumer opinions are readily available scattered around the Web, which means that the whole process of collecting consumer opinions, processing them into brand images, and comparing any two images to assess their mutual fit can be performed in a fully automated way. As a result, the vision of an entire system capable of discovering promising service combinations in an automated way appears feasible.

To better understand the technical challenges faced by such a system, we can imagine that each service entered into the system is a vertex in a graph. The edge connecting any two services represents a potential composition between them, and the weight of this edge computed from the profiles is a measure of the *fit* between those services. Further, as we foresee that combinations of more than two services are also possible, such a combination could be defined as a clique in the graph; the pairwise fit measured between all services in such a combination (all edge weights from the clique) can be used to compute its overall fitness. The task that the system has to perform can be thus translated into the problem of finding in this graph a set of cliques of highest weight such that each of the vertices is in at least one such clique. There might be additional constraints imposed on the problem. For example, the number of vertices per clique can be limited, as the bigger the size of service bundle, the higher are management costs of the composition incurred by providers. Moreover, the number of cliques per vertex can also be constrained, as each additional service composition requires also additional management workload, and also increases the possibility of a conflict of interests. We can formalize this task together with the two constraints as a *weighted K-clique b-matching problem*.

Definition 1.1. *Let K be a set of positive integers, $G = (V,E)$ be a weighted undirected graph, and b be a vector of individual vertex capacities $b[v]$ for each $v \in V$.* Weighted K-clique b-matching *in G is a set of cliques M such that the size of each clique from M belongs to K, and such that each vertex v in G belongs to at most $b[v]$ cliques from M. The weight of such a matching M is defined as a sum of the weights of all the cliques in M.*

Problem Statement 1.2. Weighted K-clique b-matching problem *involves finding the K-clique b-matching whose weight is the highest.*

Naturally, simplifications of this problem exist, for example with $K = \{k\}$ or $b = 1$, yet even simple k-clique (1-)matching is NP-hard for $k \geq 3$. Therefore,

creating a system that finds the optimal solution may not be feasible and we will have to settle for an approximate solution.

When it comes to a high-level architecture, one option would be to have a system that is fully centralized. Such an approach has some obvious advantages. For example, a centralized system has full access to the entire knowledge about all services that need to be matched. On the other hand, centralized components raise issues of trust and fairness. Concentration of decision making in the hands of a single actor simplifies computations but can also lead to possible bias in decisions made by the controlling party. For example, if the dedicated centralized component is associated with one of the providers, it might be tempted to influence the solution to the advantage of this provider, and other nodes can justifiably question its fairness. Moreover, even if a third party unrelated to any provider undertakes the role of the broker, it may come to the conclusion that it is more profitable if it can get paid in return for nodes gaining higher priority in choosing a clique.

For these reasons, we aim to investigate if we can do the service matching in a decentralized fashion. In such an approach, each service would have a dedicated node in the system that is responsible for discovering the most suitable potential partners and forming cliques with them. This means that services that join the system enter into a co-opetition environment, in which nodes have to *co-op*erate with other nodes which might be their direct com*petition* in search for most suitable partners. Moreover, although the nodes are forced to collaborate with each other, still each one of them is trying to maximize its own choices. In such a system, one of the biggest challenges would be to devise an algorithm that would enforce the rules that would allow nodes to agree on their choices.

1.2. RESEARCH GOALS

The aim of this work is to devise a *fair* distributed algorithm capable of finding a *good quality* weighted K-clique b-matching *efficiently* and *reliably*, so that it can be used in the system described above. Here we discuss each of the four specified properties. We explain the importance of each of these properties on determining if the proposed algorithm would be suitable for use in the created service matching system and we list what challenges need to be faced in order to provide each of these properties.

Research Goal 1: Algorithm's Fairness. Before any provider decides to enter its service into a matching system, it would first want to ensure that its service would have an equal chance of finding most suitable partners as any other service entered into the system. A good rule of thumb would be then that none of the providers should be able to obtain a privileged position due to the way

the system is implemented. Therefore, one of the most important properties of our desired K-clique b-matching algorithm should be for the nodes to be equal, with no node playing a specific, critical role, as this could allow such a node to gain too much control and bias the matching process to its own advantage. Yet reaching an agreement when all nodes are equal and there is no leader to make a final decision is not trivial. This brings us to our first challenge: **How to devise a distributed algorithm that would converge to a correct K-clique b-matching without breaking the equality of all nodes?** This challenge is closely related to our next issue revolving around the constructed solution. Although each of the nodes is selfish and would like to form the best possible cliques, it might be impossible to grant those wished to every node due to the constraints imposed on the number of cliques per node. As a result, some nodes would have to settle for a suboptimal solution. The crucial question is: **What properties of the constructed K-clique b-matching can ensure that it will be perceived as fair by all nodes?**

Research Goal 2: Quality of the Solution. Weighted K-clique b-matching falls into the family of NP-hard problems. In fact, the only subset of this problem that is solvable in polynomial time is the weighted b-matching problem ($K = \{2\}$ in that case), and even for this simpler problem there is not known a distributed algorithm capable of finding an optimal solution in polynomial time, even if we additionally assume that b=1 for every node, although many distributed approximation algorithms exist. This means that finding the optimal solution for weighted K-clique b-matching in a distributed fashion might be too expensive. Nonetheless, the weight of a clique corresponds directly to the level of fitness between the services in this clique, which in turn can affect the success of the service bundle. Therefore, we would like to be able to give some guarantees about the K-clique b-matching solution. One possible way would be to create a distributed algorithm that finds an approximate solution. This naturally will not guarantee the quality of the individual cliques, which is more important from the point of view of the individual services that operate on the selfish agenda and are thus interested only in maximizing the quality of their own cliques and are oblivious to the quality of other cliques or the weight of the entire K-clique b-matching. Yet, from our perspective, the approximation factor would be a good indicator of the quality of the average clique in the solution. An important question is therefore: **Is a distributed approximation algorithm for solving K-clique b-matching that is also fair, reliable, and efficient possible?** Followed immediately by a question: **What would the approximation factor of such an algorithm be?**

Research Goal 3: Scalability. We would like for our distributed algorithms to be able to operate on a wide range of network sizes. In this dissertation we target scenarios with a few hundred up to a few thousand nodes, but our goal is to

scale to the order of tens of thousands or even more. Yet, just as the complexity of K-clique b-matching problem forces us to keep our expectations with regard to the quality of solution in check, so we must ground in reality our expectations towards efficient operation of our algorithm in large networks. The increase in the network size is equivalent from each node's perspective to the increase in the number of potential partners. As each node tries to find for itself the best cliques, with the increase of the network size the number of possible clique combinations grows superlinearly, which translates to the growing global costs of the algorithm. Even if we settle for an approximated solution instead of an optimal one, the complexity of the devised algorithm might remain high. Additionally, as we are in the realm of distributed algorithms, reaching the solution would require some level of communication between nodes, and the impact of the communication on the execution costs of the algorithm can be at least threefold. Firstly, there is the obvious cost related to the number of messages that are sent by the nodes. Secondly, the information received by the nodes can force them to reevaluate their own decisions, thus the computational cost can additionally increase. Lastly, each node might need to store information about available potential partners and the state of the algorithm locally, imposing some costs on the data storage. We investigate questions such as the following: **How does the increase in the network size affect the costs of our algorithms execution? Which types of workloads — computational, communication, memory — are impacted the most? What kind of approaches can we use to limit the negative impact of network size on these costs and what tradeoffs are associated with each of these approaches?**

Research Goal 4: Robustness. Large distributed systems are not monoliths, but are composed out of thousands, if not millions, of interconnected components. As a result, there are many points that can fail: messages can get lost or become scrambled, the delivery order might be different from the order in which messages were sent, or a communication channel can break altogether, machines might freeze or die, or they might reboot and come back to operation. Failures are intrinsic to the nature of large distributed systems and should be taken into account already at the earliest stages of distributed algorithm design. Otherwise, even a seemingly meaningless failure may significantly impair the operation of a distributed algorithm up to the point of stalling it in an incorrect state and prohibiting it from further execution. Thus, a good approach is to *design for failure*, that is assume from the very start that failures do occur and design an algorithm in such a way that it can recover from a failure when it happens. Interesting questions are therefore: **What approaches can we use to equip our algorithm in self-healing mechanisms? What type of failures our algorithm would be able to recover from?**

1.3. RESEARCH METHODOLOGY

The ideas laid forward in this dissertation are supported by both theoretical analysis and experimental validation. Where possible, we present formal proofs for the properties of introduced algorithms. This allows us, among others, to establish beyond any doubt the correctness of our algorithms, to give hard guarantees for the quality of solutions, or to ensure that our algorithms converge in a timely manner and to provide the worst-case or probabilistic estimated time bounds.

The experimental part of our work consists of simulations, all of which were carried out using PeerSim, an open-source simulator developed for testing peer-to-peer protocol on a single physical machine [MJ09]. The simulations serve two distinct purposes. First, they complement the formal analysis. For example, we were able to contrast the theoretical worst-case time bounds with an average number of rounds needed for convergence by our simulations. Second, the simulations help us examine the behavior of our more complicated algorithm frameworks which consists of our clique matching algorithms combined together with selected gossiping protocols. Given the generally chaotic behavior of gossiping protocols and complex interactions between the protocols, the formal analysis of these frameworks would be too complicated. In such cases, simulations let us peek into the progression of the network state over time and to reason about the effectiveness of our proposed solutions.

1.4. OUTLINE AND CONTRIBUTION

The remainder of this dissertation is organized into five chapters whose main contributions we discuss here briefly.

Chapter 2 introduces the basic distributed algorithm for finding weighted k-clique matching and focuses on providing theoretical proofs for the algorithm's and its final outcome properties. In particular, we prove the algorithm's correctness, self-stabilization, and convergence bounds, and we show that the resulting k-clique matching is unique, stable and at most k times worse than the optimal solution in terms of the total cliques weight. This chapter is complemented by Appendix A in which we perform experimental investigation on the relation between edge-weight distributions and convergence speed of our algorithm.

In Chapter 3 we explore how the basic algorithm can be modified in order to solve generalizations of weighted k-clique matching problem. First, we explain how our algorithm can be adapted to solve the problem in which instead of a single size k we admit cliques to be of any size from some limited set of values, K. Then, we do the same for the problem in which instead of at most one clique per node,

nodes can specify a number of cliques, b, they want to be part of. Finally, we combine the two variants of the algorithm to solve the K-clique b-matching – a combination of the two previous problems. Along, we discuss how the properties discussed in Chapter 2 are preserved also for each of the presented variations of our basic algorithm.

Chapter 4 focuses on the practical aspects of the presented algorithms including improvements in terms of computational and communication load. We start by discussing how various types of heuristics can limit the cost of computations performed locally by each node and how they influence other algorithm's properties, especially convergence time estimates. Further we focus on how a node can discover other nodes in the system by means of existing gossiping protocols and how the so-called partial views created by these protocols can be used by our heuristics. Lastly, we propose to use a separate gossip protocol also for dissemination of local status information, replacing costly broadcasting.

In Chapter 5 we present brand alliance formation as a possible application for our algorithms. To help brands find the most attractive partnerships, we combine our findings on heuristics, partial views and status dissemination via gossiping into one framework, which we test on real-life data. The preliminary simulation results lead to further framework improvements.

Chapter 6 concludes this dissertation by summarizing our most significant observations and results and outlining possible directions for the future research.

CHAPTER 2

Weighted k-Clique Matching Algorithm

In this chapter, we focus on the simpler, although still NP-hard, version of weighted K-clique b-matching problem in which $K = \{k\}$ and for each node $b = 1$. Thus, the sought-for solution is a set of equal-size cliques (with k nodes each) whose combined weight is the highest, and such that every node can be a part of at most one clique. We refer to the problem of finding such a set as a weighted k-clique matching. On the following pages, we describe our distributed algorithm for finding a $1/k$-approximation of the optimal solution. We pay special attention to proving a variety of the algorithm's properties that guarantee its correctness, convergence (even if some transient fault occurs), fairness of solution (which we will reformulate as a stability of the final k-clique matching) and the solution's quality. Finally, we also provide a proof on the upper bound of convergence time and compare it with the results of simulations for selected values of k. A more extensive version of this analysis for various distributions of edge weights is included in Appendix A.

2.1. MODEL OUTLINE

We consider a set of N *entities*, be they commercial service or 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 form 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.

The target is to group entities in cliques of k members in such a way that the aggregate clique weights are maximized.

2.1.1. k-Clique Matching Problem Formalization

The problem described above 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 clique-weight functions 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 2.1. *(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. The* weighted k-clique matching problem *concerns the task of finding in a given graph a k-clique matching with the largest weight.*

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: $O(|V|(|E| + |V|log|V|))$ [Gab90]. Yet for any $k \geq 3$, a weighted k-clique matching problem becomes NP-hard (see [KH78]).

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.

2.2. THE ALGORITHM

2.2.1. Weighted 2-Clique Matching

Our protocol for creating a k-clique matching overlay is inspired by the self-stabilizing algorithm by Manne and Mjelde [MM07], which finds a matching that 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, 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 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 [MM07] 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. Yet, before we provide pseudocode for this algorithm, we first present a sequential algorithm by Preis [Pre99] 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 [Avi83], 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 coincident 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 coincident to it from E until E becomes empty (see Figure 2.1).

It is easy to show that the approximation factor for Preis's algorithm (and similarly for the algorithm by Manne and Mjelde) cannot be larger than 1/2. Consider the graph in Figure 2.2. The maximum weighted matching in this graph consists of two edges $\{a,b\}$ and $\{c,d\}$ and its total weight is 2. Yet, the only locally heaviest weight edge in this graph is $\{b,c\}$ and this edge will be chosen by Preis's

Input: graph $G = (V, E)$
1: $M \leftarrow \emptyset$
2: **while** $E \neq \emptyset$ **do**
3: $e \leftarrow$ locally heaviest edge from E
4: $M \leftarrow M + \{e\}$
5: $E \leftarrow E - \{e\} - \{e' \in E : e'$ is coincident to $e\}$

Figure 2.1: Preis's sequential greedy algorithm for weighted matching.

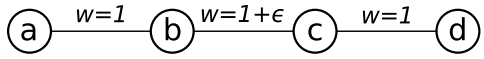


Figure 2.2: Graph in which matching created by Preis's algorithm is only (1+e)/2 of the optimal matching.

algorithm as the first edge to be added to the constructed matching. At that point the coincident edges $\{a,b\}$ and $\{c,d\}$ are removed and the remaining set of edges becomes empty. The algorithm finishes with only one edge in the matching and the weight of this matching is $1+\varepsilon$. As ε is an arbitrary small non-negative number, the weight of the constructed matching is arbitrarily close to $1/2$ of the weight of the optimal matching.

Hoepman [Hoe04] describes how Preis's algorithm can be distributed deterministically. But we can also look at the algorithm from [MM07] 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 w_v, eliminates from the matching all other edges that have v as one of the ends and a weight smaller than 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 coincident to it, then the nodes at the ends of this edge will 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.

2.2.2. Weighted 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 [MM07] for solving the weighted k-clique matching problem, we start by changing the variables stored by each node.

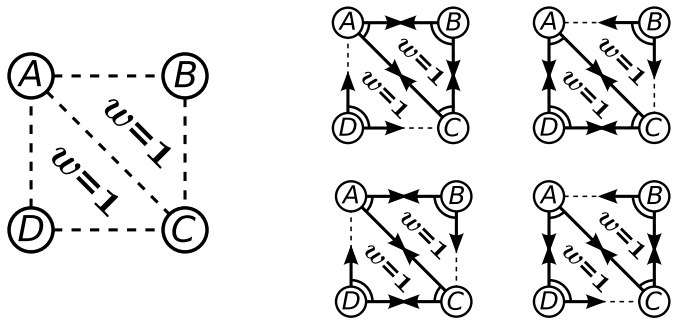


Figure 2.3: Two overlapping 3-cliques of equal weight impeding the correct convergence of the protocol and 4 possible system states: the contents of $\mathbf{C}_v$'s are visualized by outgoing arrows.

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 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 safe 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 safe 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 2.3. Here we have two overlapping 3-cliques of equal weight and nodes v_A and v_C can choose either clique, moreover, they can keep changing their decision, as either choice is equally good. As v_B and v_D are informed only about the weight of cliques that nodes v_A and v_C pursue, they have no means to discriminate between the four possible configurations from Figure 2.3. Thus, v_B and v_D are stuck in their choice of v_A and v_C 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.

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 to-

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

Active thread:

1: **loop**
2: $C \leftarrow \{\}$
3: **for all** $U \equiv \{u_1,\ldots,u_{k-1}\} \subseteq N(v)$ **do**
4: **if** $attr_v(U) > attr_v(C)$ **then**
5: $C \leftarrow U$
6: $\mathbf{C}_v \leftarrow C$
7: $\mathbf{w}_v \leftarrow w(\{v\}+\mathbf{C}_v)$
8: **send** $\mathbf{w}_v$ **to** all $u \in N(v)$

Passive thread:

1: **loop**
2: **receive** $\mathbf{w}_u$ **from** any $u \in N(v)$
3: **store** $\mathbf{w}_u$ **locally**

Figure 2.4: Self-stabilizing weighted k-clique matching protocol (executed by node v).

tal 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.

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 pseudocode of our weighted k-clique matching protocol is presented in Figure 2.4. Each node in the network executes two threads: active one and passive one. In the active thread, in an infinite loop each node 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* from the v's perspective if and only if

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

and denote this fact 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 $attr_v()$:

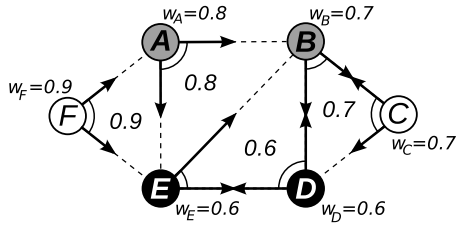
$$attr_v(C) = \begin{cases} w(\{v\}+C) & \text{if } proper(v, C) \\ -\infty & \text{otherwise} \end{cases}$$

As a result, we can now check whether C' is better than C by evaluating the expression $attr_v(C') > attr_v(C)$. By executing lines 3–5, node v chooses the heaviest admissible (most attractive) clique, setting $\mathbf{C}_v$ and $\mathbf{w}_v$ accordingly.

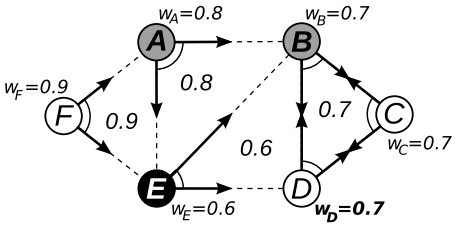
Finally, node v sends the new value of $\mathbf{w}_v$ to all of its neighbors (line 8). Node v could also send the information about the contents of $\mathbf{C}_v$, but this would be redundant. Because we have ensured that clique weights are unique, each node can easily differentiate cliques solely by their weights.

The communication between the nodes is fully asynchronous. The **send** operations are non-blocking. Once the message is sent the thread immediately continues with subsequent computations without the need to wait for the recipient to receive the sent message. The task of receiving the messages from v's neighbors is undertaken by a separate (*passive*) thread. The only job of this thread is to wait for new messages from v's neighbors and to store them in the local memory which is shared with the *active* thread. The active thread can then at the beginning of each loop execution read in all the information about the cliques chosen by all v's neighbors. Such decoupling of concerns allows nodes to be constantly looking for the best clique without the need to synchronize their communication with other nodes.

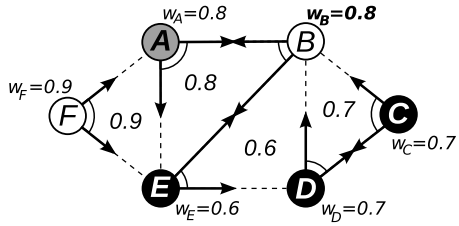
To sum up, what every node is doing in each round boils down to solving a version of the heaviest k-subgraph problem in a graph induced by the node itself and all its neighbors. What is different from the classical k-subgraph problem is that: (a) we are interested only in k-subgraphs that are cliques,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



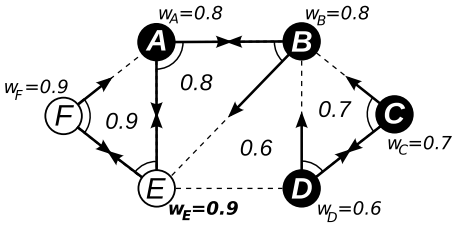
(a) initial state



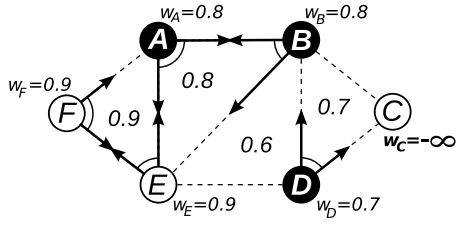
(b) D has moved



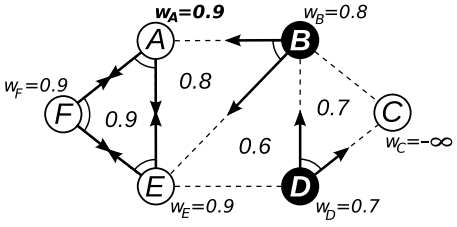
(c) B has moved



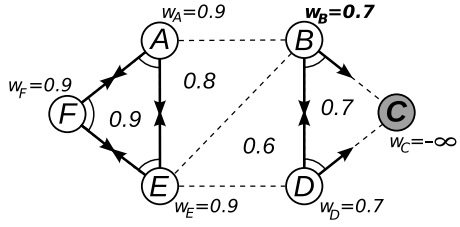
(d) E has moved



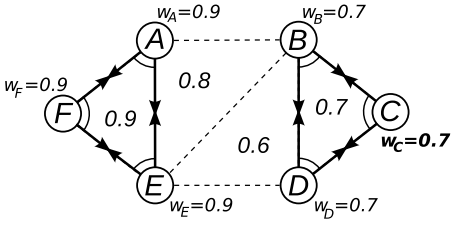
(e) C has moved



(f) A has moved



(g) B has moved



(h) C has moved

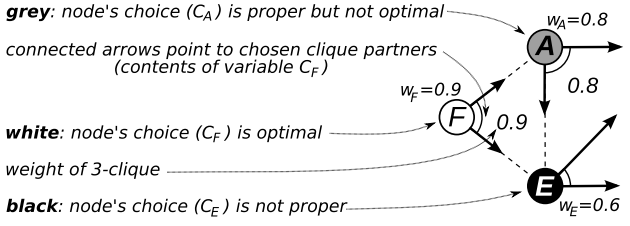


Figure 2.5: Step-by-step illustration of k-clique matching protocol for $k = 3$.

(c) each of the neighbors imposes a constraint on the minimum weight of a clique it can be part of.

To better understand how the algorithm works, consider a possible execution of the 3-clique matching protocol on the graph of six vertices and four cliques depicted in Figure 2.5. The weight of each 3-clique is given inside the clique. Let us assume that only one node can execute its active thread at a time, and that all other nodes wait until they receive an updated value of this node's chosen clique weight. Imposing such a strict coordination between nodes is only for the purposes of this example. In a real life implementation all nodes execute the code of their active (and passive) threads in parallel, and all the communication is asynchronous.

In the initial state (Figure 2.5(a)), no clique is matched; there are two nodes, C and F, whose current choice of clique partners is optimal, and there are 4 nodes, A, B, D, and E, which can improve their choices of cliques given the current information about the neighbors' clique weights. For A and B (in grey circles), they current choices are proper but there exist better (heavier) proper cliques — $\triangle AEF$ for A and $\triangle ABE$ for B — that are available. For D and E (in black circles), their current choice of clique $\triangle BDE$ is not even proper, as the weight of $\triangle BDE$ is smaller than the weight reported by B. Assume that vertex D makes a move first (Figure 2.5(b)). It's current choice of $\triangle BDE$ is not proper because $w(\triangle BDE) = 0.6 < 0.7 = \mathbf{w}_B$. The only other clique that D can be part of is $\triangle BCD$, which is proper because $w(\triangle BCD) = 0.7 \geq 0.7 = \mathbf{w}_B$ and $w(\triangle BCD) = 0.7 \geq 0.7 = \mathbf{w}_C$. Thus, D points to clique $\triangle BCD$ and sets $\mathbf{w}_D = 0.7$. Now all three vertices B, C, and D point to each other, thus clique $\triangle BCD$ is matched, although $\triangle BCD$ is not the optimal choice from the point of view of node B. If vertex B makes a move now (Figure 2.5(c)), it will point to $\triangle ABE$ which is proper ($w(\triangle ABE) = 0.8 \geq 0.8 = \mathbf{w}_A$ and $w(\triangle ABE) = 0.8 \geq 0.6 = \mathbf{w}_E$) and heavier than $\triangle BCD$, breaking up the newly matched clique $\triangle BCD$. If vertex E pointed now to $\triangle ABE$, this clique would become matched. But there is a more attractive clique $\triangle AEF$, and this is the clique that E chooses (Figure 2.5(d)). After B has pointed to $\triangle ABE$, vertex C has no clique it could choose. The only clique that C can point to is $\triangle BCD$, yet because B is pointing to a heavier clique ($\mathbf{w}_B > \triangle BCD$) , $\triangle BCD$ is not proper and, thus, no longer a viable choice. Therefore, C is left without any clique (Figure 2.5(e)), it has to set $\mathbf{C}_C = \emptyset$ and $\mathbf{w}_C = -\infty$. Now, vertex A abandons clique $\triangle ABE$ for heavier $\triangle AEF$, and $\triangle AEF$ becomes matched (Figure 2.5(f)). After A's move, the only vertices that can improve their situation are B and D, which both point to cliques that are not proper given current state of the network. If B makes a move now, it will point to the only available clique $\triangle BCD$ (Figure 2.5(g)). One of the consequences of B's decision is that D no longer has to change its state, as $\triangle BCD$ has become proper again. Another consequence is that now $\triangle BCD$

becomes available for vertex C. After C points to clique $\triangle BCD$, $\triangle BCD$ becomes matched and the entire network reaches a safe state, in which no vertices exist that could improve their choices.

2.3. ALGORITHM PROPERTIES WITH PROOFS

2.3.1. Short Introduction to Self-Stabilization

Self-stabilization is a term coined by Dijkstra in [Dij74] to describe systems that are guaranteed to converge to a correct state irrespectively of the state they are initially in. Such types of systems are especially desirable in situations where transient faults may occur. Transient faults are arbitrary in nature, but it is assumed that there is a point in time after which they no longer occur. These faults can cause the system to reach an arbitrary state. Self-stabilizing systems can automatically recover from such faults, although they do permit the system to exhibit erratic behavior during the convergence to a correct state. This is referred to in [Tel00] as an *optimistic* approach to fault tolerance; in contrast to robust algorithms that usually try to mask the occurrence of failures.

More formally, the self-stabilizing property of a system is defined with respect to a specific predicate P [Sch93]. Thus, for the system to be self-stabilizing with respect to predicate P, the following two conditions must be met:

- *convergence*: Regardless of the initial state, the system reaches, in a finite number of state transitions, a state that satisfies P.
- *closure*: Once the system is in a state that satisfies P, it can transit only to states that satisfy P.

States that satisfy P are referred to as *safe* or *legitimate*. More on self-stabilization can be found in the monograph by Dolev [Dol00].

2.3.2. The Proof

We will provide the proof for the algorithm in the case of a shared-memory model of communication.[1] Our reasoning largely follows the organization of the proof provided by Manne and Mjelde, but is otherwise different, if only for the reason that we are dealing with the generalized case. We return to these matters below.

In a shared-memory model, we do not have to concern ourselves with the messages in the communication queues. The nodes communicate with each other by means of shared registers. Each node can read the registers of its neighbors and write to its own registers. For our protocol, nodes keep in shared-memory values

[1] Dolev, in [Dol00], shows how a system with shared-memory model can be converted into a system with a message-passing model without the loss of self-stabilization.

of the two variables $\mathbf{w}_v$ and $\mathbf{C}_v$. These two variables form the local state of a node. The state of the system as a whole consists of the local states of all its nodes.

To make our deliberations on the correctness and the convergence of our protocol easier, we rewrite the protocol in the form of guarded commands [Dij76]:

$$\langle \text{guarded command} \rangle ::= \langle \text{boolean expression} \rangle \rightarrow \langle \text{statement list} \rangle$$

The semantics of the guarded command impose that a node can execute the statements of a particular guarded command only if the corresponding boolean expression (the guard) evaluates to true for this node. Subsequently, we call a node *enabled* to make a move if the guard of one of the node's commands evaluates to true.

The version of our protocol using guarded commands is depicted in Figure 2.6. In fact, our protocol uses only one guarded command with a guard (G) that is composed as an alternative of four boolean expressions (G0), (G1), (G2), and (G3). For a given node v, (G0) checks whether the size of set $\mathbf{C}_v$ is correct, (G1) checks whether there are any discrepancies between the value of $\mathbf{w}_v$ and the weight of $\{v\} + \mathbf{C}_v$, (G2) checks if any of the neighbors from $\mathbf{C}_v$ is pursuing a clique with weight higher than the weight of $\{v\} + \mathbf{C}_v$, and (G3) checks if there exists for node v an available clique better than the current one. If any of these expressions evaluates to true, node v executes statements (S1) and (S2) that recompute the values of $\mathbf{C}_v$ and $\mathbf{w}_v$.

Observe, that the algorithms from Figure 2.4 and Figure 2.6 are equivalent (apart from the operations of sending and receiving values of $\mathbf{w}_v$ which are not needed in memory-shared model). Statements (S1) and (S2) are only a mathematical notation for what the algorithm in Figure 2.4 does in lines 2-7. Moreover, observe that if guarding expressions (G0), (G1), (G2), and (G3) are false, the execution of (S1) and (S2) does not change the values of $\mathbf{C}_v$ and $\mathbf{w}_v$.

We now proceed to proving that the algorithm defined in Figure 2.6 is self-stabilizing with regard to predicate P:

$$\begin{aligned} P: \quad & \forall_v \big(|\mathbf{C}_v| + 1 = k \ (\vee\ \mathbf{C}_v = \emptyset) && \neg(G0) \\ & \quad \wedge\ \mathbf{w}_v = w(\{v\} + \mathbf{C}_v) && \neg(G1) \\ & \quad \wedge\ \forall_{u \in \mathbf{C}_v}\ w(\{v\} + \mathbf{C}_v) \geq \mathbf{w}_u && \neg(G2) \\ & \quad \wedge\ \nexists_{U \subseteq N(v)}\ (|U| + 1 = k && \neg(G3) \\ & \qquad\qquad \wedge\ \forall_{u \in U}\ \mathbf{w}_u < w(\{v\} + U)) \\ & \qquad\qquad \wedge\ \mathbf{w}_v < w(\{v\} + U) \big). \end{aligned}$$

Note that predicate P contains the negations of guard (G) for all nodes in the system. Thus, we can summarize it shortly with: no node is eligible for a move, i.e., $P : \forall_v \neg(G0) \wedge \neg(G1) \wedge \neg(G2) \wedge \neg(G3)$. As a direct consequence of the definition

Constants:
k

Variables:
$\mathbf{w}_v$; $\mathbf{C}_v$

Actions:
do forever:

$$
\begin{aligned}
&|\mathbf{C}_v| + 1 \neq k \ (\wedge\ \mathbf{C}_v \neq \emptyset) && \text{(G0)}\\
&\vee\ \mathbf{w}_v \neq w(\{v\} + \mathbf{C}_v) && \text{(G1)}\\
&\vee\ \exists_{u \in \mathbf{C}_v}\ w(\{v\} + \mathbf{C}_v) < \mathbf{w}_u && \text{(G2)}\\
&\vee\ \exists_{U \subseteq N(v)}\ \big(|U| + 1 = k && \text{(G3)}\\
&\qquad\qquad \wedge\ \forall_{u \in U}\ \mathbf{w}_u \leq w(\{v\} + U)\\
&\qquad\qquad \wedge\ \mathbf{w}_v < w(\{v\} + U)\big)\\
&\rightarrow\ \mathbf{C}_v := \operatorname{argmax}_U \big\{ w(\{v\} + U) : && \text{(S1)}\\
&\qquad\qquad U \subseteq N(v)\ \wedge\ |U| + 1 = k\\
&\qquad\qquad \wedge\ \forall_{u \in U}\ \mathbf{w}_u \leq w(\{v\} + U) \big\}^*\\
&\quad \mathbf{w}_v := w(\{v\} + \mathbf{C}_v) && \text{(S2)}
\end{aligned}
$$

*) In case no such U exists, $\mathbf{C}_v := \emptyset$ and $\mathbf{w}_v := -\infty$.

Figure 2.6: k-clique matching protocol in guarded commands formalism.

of P we have that the system is closed with regard to P. That is, from the definition, once the system reaches a state that satisfies P, it will be able to transit only to states that satisfy P, or, in our particular case, the system just stays in the state satisfying P indefinitely, because once P becomes true, no node can perform any action, so the state of the system cannot change any more and P stays true. Thus, the second condition for showing that our protocol is indeed self-stabilizing is met.

Lemma 2.2. *(Closure) The k-clique matching algorithm as defined in Figure 2.6 is closed with regard to predicate P.*

Proof. (By contradiction) Assume the contrary, that the system is in a state S that satisfies P (safe state), and that due to one or more actions of the nodes, the system transits to a state S' in which P is false. In order for the transition from state S to any other state to take place, there must exist at least one node v that is eligible for a move in state S. This is possible only if at least one of the guards $(G0)$-$(G3)$ evaluates to true for node v in state S. Yet, from the assumption, S satisfies predicate P, which states that for any node, thus also for node v, each of the guards $(G0)$-$(G3)$ is false, which leads to a contradiction. There does not exist a node able to perform any action in S, which means that the system stays in S forever. □

Before we prove the convergence of the system with regard to P, we will first

prove the correctness of the algorithm, that is once the system reaches a safe state, the sets of links stored by nodes in $\mathbf{C}_v$ forms a correct k-clique matching. We also show that this k-clique matching is stable and is a $1/k$ approximation of the optimal matching. Moreover, we give a proof of the uniqueness of the solution produced by the algorithm.

Correctness

Lemma 2.3. *(Initial observation) In a safe state,* $|\mathbf{C}_v| + 1 = k \ (\vee \ \mathbf{C}_v = \emptyset)$ *and* $\mathbf{w}_v = w(\{v\} + \mathbf{C}_v)$ *for each v.*

Proof. This statement is true, otherwise, (G0) or (G1) is true for v, making node v eligible for a move and predicate P false. □

Lemma 2.4. *In a safe state, for each node v it holds that if* $\mathbf{C}_v \neq \emptyset$*, then* $\forall_{u \in \mathbf{C}_v} \ \mathbf{C}_u + \{u\} = \mathbf{C}_v + \{v\}$.

Proof. (By contradiction) Assume the contrary, that in a safe state there exists a node v such that $\mathbf{C}_v \neq \emptyset$ and $\exists_{u \in \mathbf{C}_v} \ \mathbf{C}_u + \{u\} \neq \mathbf{C}_v + \{v\}$. From Lemma 2.3 and the assumption of the uniqueness of clique weights, we have that because $\mathbf{C}_v + \{v\} \neq \mathbf{C}_u + \{u\}$, also that $\mathbf{w}_v \neq \mathbf{w}_u$. Let's partition $\mathbf{C}_v$ into three sets:
$U^{>} = \{u \in \mathbf{C}_v : \mathbf{w}_u > \mathbf{w}_v\}$
$U^{=} = \{u \in \mathbf{C}_v : \mathbf{w}_u = \mathbf{w}_v\}$
$U^{<} = \{u \in \mathbf{C}_v : \mathbf{w}_u < \mathbf{w}_v\}$
Because there exists at least one node $u \in \mathbf{C}_v$ such that $\mathbf{w}_u \neq \mathbf{w}_v$, then at least one of the sets $U^{>}$ or $U^{<}$ is not empty:
a) if $U^{>} \neq \emptyset$ then (G2) is true for v. Thus, v is eligible for a move, contradicting our assumption of the safe state.
b) if $U^{>} = \emptyset$ then $U^{<} \neq \emptyset$. In this situation, any node $u \in U^{<}$ is eligible to make a move because $\mathbf{C}_v - \{u\} + \{v\}$ is such that (G3) is true for u. □

Lemma 2.5. *In a safe state, for any two nodes v and u either* $(\mathbf{C}_v + \{v\}) \cap (\mathbf{C}_u + \{u\}) = \emptyset$ *(are disjoint) or* $\mathbf{C}_v + \{v\} = \mathbf{C}_u + \{u\}$.

Proof. (By contradiction) Assume to the contrary that there exist two nodes v and u such that $(\mathbf{C}_v + \{v\}) \cap (\mathbf{C}_u + \{u\}) \neq \emptyset$ and $\mathbf{C}_v + \{v\} \neq \mathbf{C}_u + \{u\}$. Because the intersection of $\mathbf{C}_v + \{v\}$ and $\mathbf{C}_u + \{u\}$ is non-empty, there must exist node z such that $z \in (\mathbf{C}_v + \{v\})$ and $z \in (\mathbf{C}_u + \{u\})$. If $z = v$ (or $z = u$), then naturally, $\mathbf{C}_z + \{z\} = \mathbf{C}_v + \{v\}$ ($\mathbf{C}_z + \{z\} = \mathbf{C}_u + \{u\}$). Otherwise, for v from Lemma 2.4, for each $t \in \mathbf{C}_v$ holds $\mathbf{C}_t + \{t\} = \mathbf{C}_v + \{v\}$, thus also for z holds $\mathbf{C}_z + \{z\} = \mathbf{C}_v + \{v\}$. Likewise, from the same lemma for node u follows that $\mathbf{C}_z + \{z\} = \mathbf{C}_u + \{u\}$. Therefore, $\mathbf{C}_v + \{v\} = \mathbf{C}_z + \{z\} = \mathbf{C}_u + \{u\}$ which contradicts our assumption that the two sets $\mathbf{C}_v + \{v\}$ and $\mathbf{C}_u + \{u\}$ are not equal. □

Corollary 2.6. *In a safe state, $M = \{Q_k : \exists_{v \in V} V(Q_k) = \{v\} + \mathbf{C}_v\}$ forms a correct k-clique matching.*

Proof. From Lemma 2.5 follows that sets of $\{v\} + \mathbf{C}_v$ for different nodes are either equal to each other or disjoint. Therefore, M does contain only cliques that are pairwise disjoint. □

Stability

The *stability* of the matching is defined as an absence of pairs of nodes v and u such that both v and u would be better off if they were matched together as compared to their current situation. We can extend this definition to k-clique matching by saying that a k-clique matching is stable if there does not exist a subset of k mutual neighbors such that all of them would prefer to be in a clique together than to remain at their current choices.

Lemma 2.7. *In a safe state, for each k-clique Q_k we have that if Q_k does not belong to the k-clique matching M generated by the algorithm, then there must exist at least one node $v \in Q_k$ such that $\mathbf{w}_v > w(Q_k)$.*

Proof. (By contradiction) Assume that in a safe state there exists a k-clique Q_k that does not belong to the k-clique matching generated by the protocol from Figure 2.6 and that for each node $v \in Q_k$ $\mathbf{w}_v \leq w(Q_k)$. Then for all nodes $v \in Q_k$ for which the strict inequality of $\mathbf{w}_v < w(Q_k)$ is true, also (G3) evaluates to true; for each of these nodes v, $U = Q_k - \{v\}$ fulfills all the conditions of (G3). Therefore, these nodes are eligible for a move, which leads to a contradiction. If no such node exists, then for all $v \in Q_k$ $\mathbf{w}_v = w(Q_k)$. Thus, from the uniqueness of clique weights and Lemma 2.3, $\mathbf{C}_v = Q_k - \{v\}$ for each $v \in Q_k$. From Corollary 2.6, this means that Q_k belongs to M, also leading to contradiction. □

Corollary 2.8. *In a safe state, the weighted k-clique matching M generated by the algorithm is a* maximal[2] *k-clique matching in terms of an unweighted graph, i.e. any clique in graph G overlaps with at least one clique from M.*

Approximation Factor of $1/k$

Apart from the properties such as stability or maximality, the quality of the k-clique matching M created by our algorithm can be also assessed in the terms of its total weight. In Theorem 2.10 we show that the total weight of M is equal to at

[2]For an unweighted graph, *maximal k-clique matching* is a k-clique matching that cannot be extended by another k-clique without breaking the constraint of cliques disjointness in the graph. *Maximum (unweighted) k-clique matching*, on the other hand, is a k-clique matching with the largest number of k-cliques.

least $1/k$ of the total weight of the optimal k-clique matching M^*; in other words, we show that our algorithm has approximation factor of $1/k$.

The idea for the proof of the following theorem on the approximation factor is to show that there exists a mapping $f : M^* \to M$ from cliques of an optimal k-clique matching M^* into cliques from the k-clique matching M returned by our algorithm that has these three properties:

- $Q_k^* \cap f(Q_k^*) \neq \emptyset$ for any $Q_k^* \in M^*$: to each clique from M^* is assigned a clique from M with which it overlaps,
- $w(Q_k^*) \leq w(f(Q_k^*))$ for any $Q_k^* \in M^*$: to every clique Q_k^* from M^*, f assigns a clique Q_k from M that is of equal or higher weight than Q_k^*,
- $|f^{-1}(Q_k)| \leq k$ for any $Q_k \in M$: any k-clique Q_k from M is assigned to at most k k-cliques from M^*.

The first two properties follow from Lemma 2.7 and will be proven in the following corollary. The third property is a direct consequence of the definition of k-clique matching. Any vertex in a graph can be incident to at most one clique from a given matching. Thus also, any k-clique Q in a graph can overlap with at most k k-cliques from M^*; one clique per each vertex from clique Q. Because f creates a mapping only between overlapping cliques, the third property surfaces.

Corollary 2.9. *For each k-clique Q_k^* that belongs to the optimal k-clique matching M^* for graph G, there exists a k-clique Q_k from the k-clique matching M from the safe state such that these two cliques have at least one node in common and the weight of Q_k^* is smaller than or equal to the weight of Q_k.*

Proof. (By contradiction) First, we will prove that the two cliques Q_k^* and Q_k have at least one node in common: Assume to the contrary that there is a clique Q_k^* in the optimal k-clique matching that does not overlap with any of the k-cliques from the k-clique matching from the safe state. Thus, for each node v from Q_k^*, $\mathbf{C}_v$ must be empty and $\mathbf{w}_v = -\infty$. Then for these k nodes guard (G3) is true and they are eligible for a move, and therefore the safe predicate is false.
Now, we can also prove that the weight of $Q_k^* \in M^*$ is smaller than or equal to the weight of $Q_k \in M$: Assume to the contrary that none of the weights of k-cliques overlapping with Q_k^* is greater than or equal to the weight of Q_k^*. Then nodes from Q_k^* are again eligible for a move, because (G3) is true for them. Contradiction. □

Theorem 2.10. *k-Clique Matching from the safe state has a total weight that is at most a factor k worse than the weight of the optimal k-clique matching M^*.*

Proof. For each k-clique Q_k^* from M^* there exists at least one k-clique from M such that its weight is equal to or greater than that of Q_k^*. Let A_{M,M^*} denote a set of such k-cliques from M. The power of this set is at least $|M^*|/k$ because each of the k-cliques from A_{M,M^*} can overlap with at most k k-cliques from M^*. The

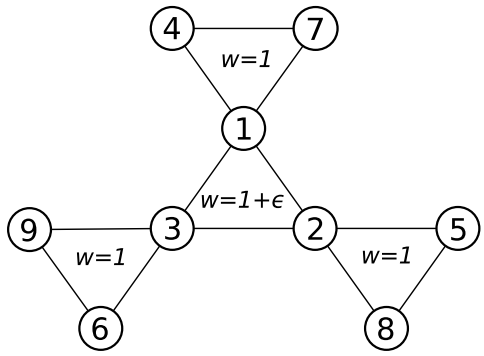


Figure 2.7: Graph in which matching created by the protocol is only $(1+\varepsilon)/k$ of the optimal matching.

total weight of M^* is: $w(M^*) = \sum_{Q_k^* \in M^*} w(Q_k^*)$. Let $f : M^* \to M$ be a function that assigns to each $Q_k^* \in M^*$ a k-clique from M that overlaps with Q_k^* and whose weight is greater or equal to that of Q_k^*. From Corollary 2.9 we know that such a clique must exist for each Q_k^*.

$$\begin{aligned} w(M^*) &= \sum_{Q_k^* \in M^*} w(Q_k^*) \\ &\leq \sum_{Q_k^* \in M^*} w(f(Q_k^*)) \\ &\leq k \cdot \sum_{Q_k \in f(M^*)} w(Q_k) \\ &\leq k \cdot \sum_{Q_k \in M} w(Q_k) \\ &= k \cdot w(M) \end{aligned}$$

The third row of the equation comes from the observation that each of the Q_k from $f(M^*)$ can be assigned to at most k cliques from M^*. From the above reasoning follows $w(M) \geq (1/k) \cdot w(M^*)$. □

Observation 2.11. *The bound of $1/k$ on the approximation factor from Theorem 2.10 is sharp.*

Proof. For each k we can create a graph, for which the weight of the matching resulting from execution of the protocol can be infinitely close to $(1/k) \cdot w(M^*)$. An example of such a graph consists of exactly $k+1$ k-cliques. There is one k-clique with the weight of $1+\varepsilon$ where $\varepsilon > 0$. The other k k-cliques have weight 1 and each of them has exactly one node in common with the first clique but they themselves are pairwise non-overlapping. An example of such a graph is shown in Figure 2.7 for $k = 3$.

There are only two maximal (in unweighted terms) k-clique matchings in such a graph. One consists of k k-cliques of weight 1 and it is the optimal weighted k-clique matching with weight of k. The other maximal k-clique matching consists

only of one clique with weight $1+\varepsilon$ and, thus, its total weight is $1+\varepsilon$. This second matching, as can easily be verified is the matching created by our protocol. As ε can have an infinitely small positive value, the ratio between the weight of this matching and the weight of the optimal matching is $(1+\varepsilon)/k \to_{\varepsilon\to 0} 1/k$. □

Uniqueness of Solution

Lemma 2.12. *If M_1 and M_2 are both k-clique matchings arising from the execution of the protocol on the same graph, then $M_1 = M_2$.*

Proof. (By contradiction) Assume that the contrary is true and that two different executions of the protocol on the same graph produced in a safe state two different k-clique matchings M_1 and M_2. Let $Q_k^{1,1}$ denote a k-clique from M_1 such that $Q_k^{1,1}$ does not belong to M_2. From Lemma 2.7 there exists at least one node $v \in Q_k^{1,1}$ such that the value of $\mathbf{w}_v$ from the second execution of the protocol is greater than $w(Q_k^{1,1})$. Denote the clique $\{v\} + \mathbf{C}_v$ from the second execution, which obviously belongs to M_2, by $Q_k^{2,1}$. Now for $Q_k^{2,1}$ we can find in M_1 a clique $Q_k^{1,2}$ such that $w(Q_k^{1,2}) > w(Q_k^{2,1})$ and $Q_k^{1,2} \cap Q_k^{2,1} \neq \emptyset$. We repeat the process choosing cliques interchangeably from M_1 and M_2. As a result, we obtain an infinite sequence of overlapping cliques in which every subsequent clique is heavier than the previous one, which is a contradiction of the finiteness of the graph G. □

Convergence

Self-stabilizing systems progress from one state to another in time steps. In each time step any node can perform a single atomic step. For our system, we assume *composite atomicity* in which an atomic step consists of the execution of the two statements (S1) and (S2) (which makes an atomic step equivalent to a move). Execution of (S1) and (S2) implicitly requires reading by a node v the values of $\mathbf{w}_u$ of all its neighbors and writing to the shared-memory the new values of $\mathbf{w}_v$ and $\mathbf{C}_v$. This is in contrast with read/write atomicity in which at most one value can be read or written into shared-memory in a single atomic step.[3]

The decision which nodes can perform their atomic steps in the given time step is made by a daemon (also referred to as a scheduler). The most common daemons are: a *central daemon* which allows to make a move by only one of the enabled nodes; a *synchronous daemon* which allows to make a move by all nodes that are enabled; and a *distributed daemon* which chooses an arbitrary subset of all enabled nodes to make a move. Obviously that last type of a daemon is the most

[3] Yet, as shown in [Dol00], it is possible to transform system with composite atomicity into system with read/write atomicity without the loss of self-stabilization property.

general one and both a central daemon and a synchronous daemon are just its special cases. Another independent classification of daemons differentiates between fair and adversary daemons. The fair daemon makes sure that each node that is enabled infinitely often will execute infinitely often, while adversary daemon makes absolutely no guarantees with respect to executions of particular nodes.

In [MM07], Manne and Mjelde proved that their matching algorithm converges in $2|M|+1$ rounds under the distributed fair daemon, in $O(3^{|V|})$ moves under the distributed adversarial daemon, and in $O(2^{|V|})$ moves under the central (*aka* sequential) adversarial daemon. Turau and Hauck improved the convergence bound for this algorithm to $O(|V||E|)$ moves under the central adversarial daemon, and also showed a modified version of this algorithm that reaches the safe state in $O(|V||E|)$ under a distributed adversarial daemon [TH11].

In view of our simulations, the most interesting is the convergence under a fair distributed daemon. In our simulations, at the beginning of each round a random ordering of all nodes is created and then the nodes execute their loops in this ordering. This ordering can be seen as a special case of the fair central scheduler which in turn is a special case of fair distributed scheduler.

Theorem 2.13. *Under a fair distributed daemon, the k-clique matching protocol converges in at most* $2|M|+1$ *rounds, where one round is the minimum time span in which each node that was enabled at the beginning of a round either made at least one move or became disabled at some point.*

Proof. (By Induction) First, observe that due to our assumption that each k-clique has a unique weight, we can order all k-cliques in the graph by their weight. We can similarly order all k-cliques in the resulting matching.
Base step: The nodes from the heaviest k-clique Q_k^1 stabilize in 2 rounds.
At the beginning of the execution, the nodes may have incorrect values of variables w and C. In such a situation nodes that belong to the heaviest clique in the graph in the first round will set values of w to less than $w(Q_k^1)$. In the second round, these nodes can assign $w(Q_k^1)$ to their variables w.
Induction step: Given that nodes from the i heaviest k-cliques from the final matching have already stabilized, only 2 rounds are needed for the nodes from the $(i+1)$-th k-clique to stabilize.
First, observe that once the nodes from the i heaviest k-cliques from the final matching create their cliques, they will never become enabled again. For every node in j-th clique, any clique that is better that Q_k^j, has at least one node that already belongs to one of the cliques from the final matching with an index smaller than j. And with nodes that do not belong to the first $j-1$ cliques, nodes from the j-th clique can create a clique whose weight will never be higher than the weight of Q_k^j.

Thus, when the i heaviest k-cliques from the final matching have already stabilized, the nodes from the next heaviest k-clique have the first round to correct their values of w which can be higher than that of Q_k^{i+1}. In the next round they all set their values of w to $w(Q_k^{i+1})$.
Thus, $2 \cdot M$ rounds are needed by the nodes from cliques that belong to the final matching to stabilize. The last round is for all the remaining nodes to set their values of C to an empty set and value of w to $-\infty$. □

The convergence bound of $O(3^{|V|})$ ($O(2^{|V|})$) moves for the k-clique matching protocol under an adversary distributed (resp. central) daemon can be proven in the same way as it is done by Manne and Mjelde in [MM07] for a simple weighted matching.

2.3.3. Discussion

We have provided proofs of some of the k-clique matching algorithm properties given the shared-memory model. Nonetheless, the algorithm can be easily modified to be consistent with a message-passing model without any loss of any of the presented properties. In fact, our algorithm provided in Figure 2.4 is such a message-passing algorithm. To be exact, this is true from the point of view of communication between the nodes, as there is still shared-memory communication between active and passive threads executed by a single node.

Using guarded-command notation the active thread can be expressed as a slight modification of guarded commands from Figure 2.6. Note, that the **send** operation here is non-blocking.

$$
\begin{aligned}
&\text{do forever:}\\
&G \rightarrow\ S1;\\
&\qquad S2;\\
&\qquad \forall_{u \in N(v)}\ \text{send } \mathbf{w}_v \text{ to } u
\end{aligned}
$$

As for the passive thread, it can be represented using the following guarded command:

$$\text{receive } \mathbf{w}_u \text{ from } u \in N(v) \rightarrow\ \text{store } \mathbf{w}_u \text{ locally}$$

When there are multiple messages ready to be received by the passive thread, we assume that the one to be received next is chosen non-deterministically. The last necessary adjustment is related to predicate P, which needs to be extended to additionally make sure that for any pair of neighbors v and u all messages in message queue from v to u and u's local information about v are fully consistent with value of v's variable $\mathbf{w}_v$.

Our decision to make the broadcast of value $\mathbf{w}_v$ independent from guard (G) and the recomputations of $\mathbf{C}_v$ and $\mathbf{w}_v$, ensures the resilience to any transient link

failures, such as fair loss, arbitrary reordering, duplication or alteration of messages. Fair loss of messages is neutralized by the infinite retransmission of the current value of $\mathbf{w}_v$. Arbitrary reordering of messages from the same neighbor can affect node v only for a limited time, as the messages with more recent values will soon replace the older one. Similarly, if a message gets scrambled in transmission, it will be soon replaced by a correct or more recent value. Also, arbitrary message reordering from different sources has no long-lasting consequences, as the values from these messages are independent and stored separately. Finally, finite duplication of messages does not hamper the algorithm as the operation of receiving and storing the values sent by neighbors is idempotent and receiving the same message twice does not change the knowledge about a node's neighborhood and has no impact on the computations of $\mathbf{C}_v$ and $\mathbf{w}_v$.

On the other note, one of the algorithm's properties is especially interesting from the point of view of perceivable fairness of the created solution. The objectives of the participating nodes are selfish. Each node is interested in finding a best possible clique for itself, and is totally indifferent to the quality of cliques formed by other nodes or the quality of the solution at large. Yet due to constraints imposed on the number of cliques per node, it is possible that some of the nodes will not be able to get their first choice and will have to settle for a clique of a lesser weight. The nodes still can perceive this situation as fair thanks to the stability property of the final k-clique matching. The stability property guarantees each node that in the final solution there will not exist a clique such that this node and other $k-1$ nodes would be better off if they formed this clique together as compared to their final k-clique choices. The arrival at the k-clique matching solution that has this stability property is encoded into the algorithm itself. The stable solution emerges from the single rule being applied by each node round by round of choosing the heaviest clique among those that to other members of this clique would be equally or more attractive than their current choices. Thus, the algorithm allows nodes to operate on a selfish agenda but also imposes that the nodes respect egoistic behavior of other nodes.

To complete our discussion, we come back to our initial remark on the differences between our proof and the proof provided by Manne and Mjelde in [MM07]. As we mentioned already, our reasoning follows the organization of the proof from [MM07]. Nonetheless, there are a few crucial differences that distinguish our proof from its predecessor. First and foremost, our proof deals with the generalized case where $k \geq 2$. Moreover, we have extended the proof by adding Lemma 2.12 on the uniqueness of the solution and Observation 2.11 with an example showing that the bound on the approximation factor is in fact sharp. Additionally, for the purposes of our proof we adopted the definition of self-stabilization as provided in [Sch93] that requires not only a proof of convergence but also a

proof of closure; the latter is absent in [MM07]. Lastly, instead of pseudo-code we explicitly used guarded-command representation of our algorithm, which we hope aids the reader to follow our reasoning.

2.4. EXPERIMENTAL EVALUATION

2.4.1. Simulation Setup

In the following simulations, we examine the efficiency of the presented protocol in partitioning the network into k-cliques. The execution of simulations is based on the notion of rounds. In each round, node after node execute the entire loop body exactly once, and the updated value of their clique weight is immediately received by their neighbors. 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 performance measure 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 measure reflects more closely the differences between the computational complexities of the protocols.

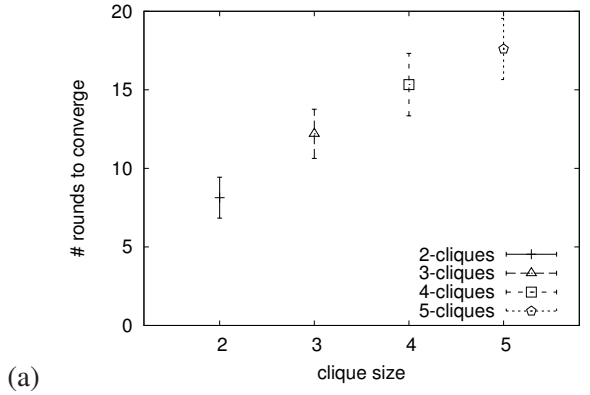
All of our simulations are conducted on complete graphs, which means that for each node v the neighbor set $N(v)$ is equal to $V - \{v\}$. As a result, any k nodes can potentially form a clique. This guarantees that, irrespectively of the distribution of edge weights, it is always possible to find a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

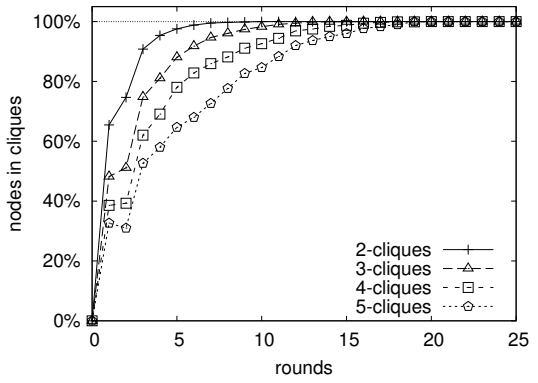
All simulations presented here were conducted using PeerSim [MJ09], an open-source simulator for peer-to-peer protocols. Each presented curve is an average of 15 simulations executed with the same parameters but different random number generator seeds. In each simulation, all nodes start without any clique chosen.

2.4.2. Performance of k-Clique Matching Protocol

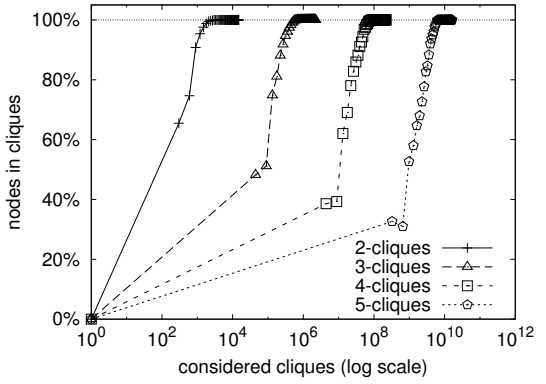
First, let us focus on the convergence speed of the basic k-clique matching protocol from Figure 2.4. Figure 2.8(a) depicts the average number of rounds (together with



(a)

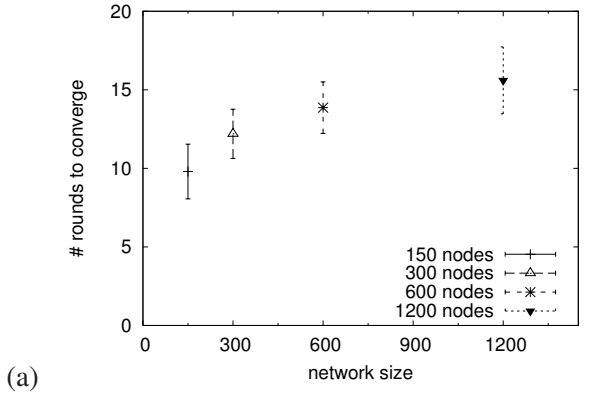


(b)

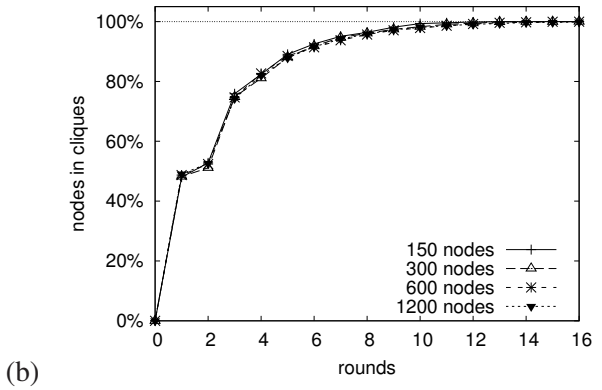


(c)

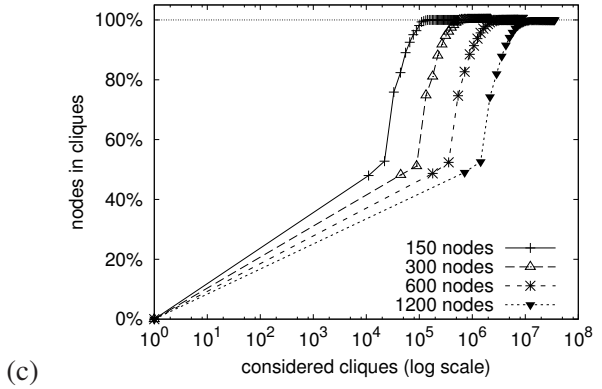
Figure 2.8: Performance of the basic k-clique matching protocol for $k \in \{2,3,4,5\}$ in networks of 300 nodes; (a) average convergence time in number of rounds, (b)-(c) percentages of nodes matched into cliques for various values of k plotted against: (b) number of rounds, (c) average number of cliques evaluated by a single node since the beginning of a simulation.



(a)



(b)



(c)

Figure 2.9: Performance of the basic k-clique matching protocol for $k = 3$ in networks of various sizes; (a) average convergence time in number of rounds, (b)-(c) percentages of nodes matched into 3-cliques for various network sizes plotted against: (b) number of rounds, (c) average number of cliques evaluated by a single node since the beginning of a simulation.

standard deviations) needed for a network of 300 nodes to converge. As we can see, the number of rounds necessary for all nodes to form disjoint cliques increases with the size of the cliques, provided k is small. Nonetheless, for all clique sizes tested the time needed for full convergence does not exceed 20 rounds, which is way below our theoretical upper bound of $2 * |M| + 1$ which for $k = 2$ amounts to 301 rounds and for $k = 5$ to 121 rounds.

Figure 2.8(b) shows in more detail how the formation of cliques in a network progresses. For each round we plot the the percentages of nodes that are in a correctly matched clique. These percentages may naturally be higher than the respective percentages of nodes that have stabilized. Moreover, it is possible that before the network fully converges, the number of correctly formed cliques decreases from one round to another (as can be seen for 5-cliques in the first and second round).

Although the clique size, k, does not have a dramatic effect on the protocol performance in terms of convergence rounds, it does affect the performance with respect to the amount of computation required by each node. This is depicted in Figure 2.8(c), which plots the same measurements (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 2.4.

The next three graphs show the differences in the convergence speed of the basic protocol for different sizes of networks. In Figure 2.9(a) we observe only a slight increase in the number of rounds needed by all nodes to find their cliques; the eightfold growth of the network size resulted in the average number of rounds increasing not 8 times but only 1.6 times. Moreover, the percentages of cliques matched at any given round are almost identical for all tested network sizes. 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.

2.5. 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) net-

works. Protocols such as T-Man [JMB09] and Vicinity [Vou06] assume weights (referred to as "proximity") between nodes. Nodes have a fixed outdegree of k neighbors, and periodically gossip with (some of) them to encounter new potential neighbors and replace higher proximity ones by lower proximity ones. When converged, each node has links to its k most proximal nodes 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 [MV80], Blum [Blu90], or Gabow and Tarjan [GT91]. In this chapter 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 nonoverlapping subgraphs, each of which is isomorphic to some given graph H. This type of problem is commonly referred to as an *H-packing* [HKNP05; CH07; CHW08], an *H-matching* [Kan94; CK98; Yus07] or an H-partition [KH78]. In the special case when graph H is a clique of size k, we obtain a k-clique matching[4] problem that we address in this dissertation.

Whereas finding a maximum weighted matching can be done in polynomial time using, for instance, the algorithm by Gabow [Gab90] 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 [KH78] which proves that finding a perfect H-matching of graph G is

[4] We adopted the terminology of the k-clique matching from [Kan94] where the term H-matching is used 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 in the highly cited [CK98]. 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.

NP-complete if H contains a connected component with at least 3 vertices.

Nonetheless, in unweighted graphs a simple greedy algorithm that finds a *maximal* H-matching has an approximation factor of $|V(H)|$ [Yus07]. This approximation factor is tight and comes from the reasoning similar to the one we used for proving the approximation factor of our k-clique matching algorithm in Theorem 2.10: every subgraph from the solution returned by the greedy algorithm intersects at most $|V(H)|$ subgraphs from the optimal H-matching. Moreover, one can achieve an even better approximation factor of $k/2+\varepsilon$ for unweighted k-clique matchings by using a polynomial time algorithm proposed by Hurkens and Schrijver [HS89]. This algorithm was originally proposed to solve the problem of k-set packings: finding a largest collection of pairwise disjoint k-sets (for $k \geq 3$), of which the k-clique matching problem is a special case. In this algorithm, a current k-clique matching is repeatedly improved by replacing $p \leq s$ cliques from the matching with $p+1$ cliques that are not in the current matching and such that the cliques in the matching remain disjoint. The constant s depends on ε and the algorithm ends once no further improvement is possible. Based on the results from [HS89], local-improvement approximation algorithms finding weighted k-set packings have been developed. Examples include Arkin and Hassin in [AH98] and Chandra et al. in [CH99] with approximation factors $k-1$ and $2(k+1)/3$ respectively. Whether any of these algorithms could be distributed remains an open question. Apart from approximation algorithms based on work of Hurkens and Schrijver [HS89], other algorithms for weighted k-clique matchings exist. For example, Hassin and Rubinstein in [HR06a] propose a randomized algorithm in which the solution is constructed from the maximum weighted cycle cover in a graph. The algorithm has approximation factor of 43/83 [HR06b] but it is limited only to cliques of size $k = 3$.

As far as 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 [Hoe04], the $O(log|V|)$-time $(1/2-\varepsilon)$-approximation algorithm by Lotker at al. [LPSP08], and a $(1-\varepsilon)$-approximation algorithm by Nieberg [Nie08]. From our perspective, the most interesting are self-stabilizing algorithms for weighted matchings, such as a $1/2$-approximation algorithm by Manne and Mjelde [MM07], which we extended in this chapter to solve the more general problem of weighted k-clique matching.

Whether other distributed approximation algorithms for weighted matching problems can also be extended to solve weighted k-clique matching remains an open question. Additionally, the research on distributed algorithms dedicated to finding H-matchings for any H other then 2-clique is scarce. For unweighted graphs, there exist a few approximation algorithms for solving H-matching prob-

lems, for example, distributed algorithms for unit-disk graphs [CH07] and planar graphs [CHW08]. And, to our best knowledge, there is no work on distributed algorithms for H-matchings in *weighted* graphs. Yet, there exist distributed algorithms, which we will discuss in the related work section of the next chapter, that tackle generalizations of weighted k-clique matching problem [SK98; TA05; SPNW06; GP10b; GP12; PS10; KW05; KMW06; KY09; KY11]. Nonetheless, none of these algorithms is self-stabilizing, making our work unique in this regard.

2.6. CONCLUDING REMARKS

In this chapter, we presented a distributed algorithm for finding a weighted k-clique matching in a graph. Using this algorithm, nodes in a network can form with their neighbors disjoint groups of a fixed small size. The formation of these groups emerges from the decisions of nodes based only on the information limited to their direct neighborhoods.

We gave formal proofs for several properties of our algorithm. First and foremost, we showed the correctness of the algorithm. Second, we proved that the algorithm is self-stabilizing and gave the upper bound of $2|M|+1$ rounds for its convergence. Third, we showed that two distinct executions of the algorithm on identical graphs always converge to the same solution. Finally, we demonstrated that the k-clique matching created by the algorithm is stable, and that its total weight is at most k times worse than this of the optimal k-clique matching.

We implemented our algorithm using the PeerSim simulator and evaluated the algorithm's performance for various sizes of k and various sizes of networks. In these simulations we assumed that networks create complete graphs with edge weights drawn randomly from the $(0,1)$ interval and arithmetic mean as a measure of cliques value. Firstly, we showed that in such settings for small values of k the convergence time is at least 6 times smaller in comparison to our estimations of $2|M|+1$ rounds. Secondly, we showed that for fixed $k=3$, the convergence time grows sublineary with the linear increase in the network size. Finally, we compared the number of rounds needed for convergence to the total computational workload experienced by each node. In accordance with our theory, the simulations showed that for small values of k the linear increase in k results in exponential increase of the total computational workload. Additionally, there is a polynomial growth of the total computational workload of a node with respect to the network size. This suggests that if the algorithm is to be used in large networks in which nodes have large neighborhoods we need to find a way to limit the computational load incurred by the nodes.

CHAPTER 3

Extensions of k-Clique Matching Algorithm

Our clique matching algorithm in its basic version imposes strict constraints on the number and the size of cliques that nodes could create. Each node can be part of at most one clique and the size of the cliques created is globally fixed. These constraints might be too restrictive for some applications. For example, when players are searching for other teammates, there might be only an upper limit on the team size. On the other hand, in case of companies looking for potential partners, each company may have a different amount of available resources, and therefore each company may be interested in participating in a different number of partnerships.

In this chapter we explore how we can extend our basic algorithm to provide more flexibility both in terms of the possible sizes of cliques created and the number of cliques that each node can be part of. We present how to relax these two constraints by introducing only minor changes to the basic algorithm. First, we will describe solutions for each of these two issues separately, and then we will combine the two solutions together to provide a single consolidated algorithm that can be easily adjusted by each node individually to fit its individual needs and capabilities.

3.1. BOUNDED VARIABLE-SIZED CLIQUE MATCHING

3.1.1. The Algorithm

The first extension concerns the ability of nodes to form cliques of various sizes. The necessary algorithm modification that would allow nodes to choose from

Constants:

K possible sizes of cliques; globally fixed
(or K_v specific to the given node v)

Variables:

$\mathbf{C}_v$ set of $k-1$ neighbors with which v wants to create a clique

$\mathbf{w}_v$ the weight $w(\{v\}+\mathbf{C}_v)$

Active thread:

1: **loop**
2: $C \leftarrow \{\}$
3: **for all** $k \in K$ **do**
4: **for all** $U \equiv \{u_1, \ldots, u_{k-1}\} \subseteq N(v)$ **do**
5: **if** $attr_v(U) > attr_v(C)$ **then**
6: $C \leftarrow U$
7: $\mathbf{C}_v \leftarrow C$
8: $\mathbf{w}_v \leftarrow w(\{v\}+\mathbf{C}_v)$
9: **send** $\mathbf{w}_v$ **to** all $u \in N(v)$ {attach to the above messages the contents of K_v}

Passive thread:

1: **loop**
2: **receive** $\mathbf{w}_u$ **from** any $u \in N(v)$
3: **store** $\mathbf{w}_u$ **locally**

Figure 3.1: Self-stabilizing weighted various-size clique matching algorithm (executed by node v).

cliques of different sizes is introduced in the form of an additional loop in lines 3–6 in Figure 3.1. The loop iterates over different sizes of cliques that the node is interested in. Set K can be set globally or each node can choose K individually. In the latter case, nodes need to be informed of other nodes' respective K sets. Only this way will the nodes know which nodes can be considered for creation of a clique with a specific size.

Naturally, adding a loop over possible values of k increases the computational load of the algorithm. In the basic algorithm, a node had to compare $\binom{N(v)}{k-1}$ combinations of potential clique members. Now it is: $\sum_{k\in K}\binom{N(v)}{k-1}$ which is $O(N(v)^{k_{max}-1})$, so although more cliques are considered, the order of complexity stays the same as for the basic algorithm with $k = k_{max}$.

In order for the algorithm to converge correctly, we have to make a stronger assumption about the uniqueness of weights. Now, the weights of cliques should be unique across all possible values of k used by nodes. This again can be easily achieved by augmenting each clique's weight by the sorted list of ids of partici-

Constants:
k

Variables:
$\mathbf{w}_v$; $\mathbf{C}_v$

Actions:

$$
\begin{aligned}
&|\mathbf{C}_v|+1 \notin K \ (\wedge\ \mathbf{C}_v \neq \emptyset) && (G_{vs}0)\\
&\vee\ \mathbf{w}_v \neq w(\{v\}+\mathbf{C}_v) && (G_{vs}1)\\
&\vee\ \exists_{u\in\mathbf{C}_v}\ w(\{v\}+\mathbf{C}_v) < \mathbf{w}_u && (G_{vs}2)\\
&\vee\ \exists_{U\subseteq N(v)}\ \big(|U|+1 \in K && (G_{vs}3)\\
&\qquad\qquad \wedge\ \forall_{u\in U}\ \mathbf{w}_u \leq w(\{v\}+U)\\
&\qquad\qquad \wedge\ \mathbf{w}_v < w(\{v\}+U)\big)\\
&\rightarrow\ \mathbf{C}_v := \operatorname{argmax}_U \big\{w(\{v\}+U): && (S_{vs}1)\\
&\qquad\qquad U\subseteq N(v)\ \wedge\ |U|+1\in K\\
&\qquad\qquad \wedge\ \forall_{u\in U}\ \mathbf{w}_u \leq w(\{v\}+U)\big\}^*\\
&\quad\ \mathbf{w}_v := w(\{v\}+\mathbf{C}_v) && (S_{vs}2)
\end{aligned}
$$

*) In case no such U exists, $\mathbf{C}_v := \emptyset$ and $\mathbf{w}_v := -\infty$.

Figure 3.2: Clique matching algorithm in guarded commands formalism.

pating nodes and imposing a lexicographical order on such new weights.

3.1.2. Algorithm Properties

As we did for our k-clique matching algorithm, we express the clique matching algorithm extension using guarded commands; see Figure 3.2. Compared to the formalization of the basic algorithm (see Figure 2.6), there is not much of a difference here, and the overall structure remains the same with only one guarded command that contains an alternative of four guards and two statements:

	$(G0)$	if $\mathbf{C}_v$ has incorrect size,
$\vee$	$(G1)$	or if $\mathbf{w}_v$ holds an incorrect value,
$\vee$	$(G2)$	or if (any) $\mathbf{C}_v$ is not proper,
$\vee$	$(G3)$	or if a better alternative exists,
$\rightarrow$	$(S1)$	recompute $\mathbf{C}_v$, and
	$(S2)$	recompute $\mathbf{w}_v$

The only things that needed adjustment are related to the size of $\mathbf{C}_v$; all expressions that regarded equality to k (e.g., $= k$, $\neq k$) have been replaced by expressions of being an element of K (e.g. $\in K$, $\notin K$).

In the simplest case the set of available clique sizes K is a singleton, contains only one value k. For such a case we have already provided in Section 2.3 proofs of self-stability (closure and convergence), correctness, stability of created k-clique matching, approximation factor of $1/k$ to the optimal k-clique matching,

and uniqueness of the solution. Almost all of these properties (with the exception of the approximation factor of $1/k$) of the clique matching algorithm are invariant to the cardinality or particular contents of the set K. This follows from the fact that when proving all of these properties for the k-clique matching algorithm, we have not referred in any way to the equipotence (equality of cardinality) of cliques, but relied solely on the uniqueness of clique weights. Therefore, the following statements are true for any size of set K:

Lemma 3.1. *(Closure) The clique matching algorithm as defined in Figure 3.2 is closed with regard to predicate P_{vs} :*

$$\begin{aligned} P_{vs}: \quad & \forall_v \big(|\mathbf{C}_v| + 1 \in K \ (\wedge\ \mathbf{C}_v = \emptyset) \\ & \wedge\ \mathbf{w}_v = w(\{v\} + \mathbf{C}_v) \\ & \wedge\ \forall_{u \in \mathbf{C}_v}\ w(\{v\} + \mathbf{C}_v) \geq \mathbf{w}_u \\ & \wedge\ \nexists_{U \subseteq N(v)}\ (|U| + 1 \in K \\ & \qquad \wedge\ \forall_{u \in U}\ \mathbf{w}_u < w(\{v\} + U)) \\ & \qquad \wedge\ \mathbf{w}_v < w(\{v\} + U)\big). \end{aligned}$$

Lemma 3.2. *(Convergence) Under a fair distributed daemon, the clique matching algorithm for a given set of clique sizes K converges in at most $2|M| + 1$ rounds, where one round is the minimum time span in which each node that was enabled at the beginning of a round either made at least one move or became disabled at some point.*

Theorem 3.3. *(Self-Stabilization) On the grounds of Lemmas 3.1 and 3.2, the clique matching algorithm is self-stabilizing with regard to predicate P_{vs}.*

Lemma 3.4. *(Correctness) $M = \{Q_k : \exists_{v \in V} V(Q_k) = \{v\} + \mathbf{C}_v\}$ forms a correct clique matching with clique sizes belonging to K.*

Lemma 3.5. *(Stability of solution) In a safe state, for each k-clique Q_k, where $k \in K$ we have that if Q_k does not belong to the clique matching M generated by the algorithm, then there must exist at least one node $v \in Q_k$ such that $\mathbf{w}_v > w(Q_k)$.*

Theorem 3.6. *($1/k_{max}$ Approximation Factor) Clique matching from the safe state has a total weight that is at most a factor k_{max} worse than the weight of the optimal clique matching M^*.*

Lemma 3.7. *(Uniqueness of solution) If M_1 and M_2 are both clique matchings arising from the execution of the algorithm on the same graph and for the same set of clique sizes K, then $M_1 = M_2$.*

3.1.3. Experimental Evaluation

The setup for simulations is identical to that of the basic algorithm in the previous chapter. We assume complete graphs in which the weights of the edges are drawn according to a uniform distribution from the interval $(0,1)$. Moreover, we assume that K is globally fixed and contains a full interval of natural values from 2 to some given k.

On clique weights and sizes

An interesting behavior of this extension of the algorithm can be observed when the function ω used to compute the weights of cliques is monotonic with respect to the clique size. The function ω is monotonic if for any two sets of nodes S and T such that $S \subset T$ we have $\omega(S) \leq \omega(T)$. Given our assumption that each clique weight is unique, ω must exhibit an even stronger property of *strict monotonicity*: for any two sets of nodes S and T such that $S \subset T$ we have $\omega(S) < \omega(T)$. When such a strictly monotonic function is used to compute the weights of cliques, nodes have a strong preference towards creating cliques of maximum possible size. Therefore, if a complete graph is assumed, all (or almost all, if n is not divisible by k_{max}) cliques created will consist of k_{max} nodes. In case n is not divisible by k_{max}, the remaining $n \mod k_{max}$ nodes will be forced to create a clique of a smaller size. Similarly, if ω is strictly monotonically decreasing, all (or almost all, if n is not divisible by k_{min}) cliques created will consist of k_{min} nodes. In case n is not divisible by k_{min}, the remaining $n \mod k_{min}$ nodes will be left without any clique. Moreover, the same phenomena that take place when the function is strictly monotonically decreasing, can also be observed even when a weaker property holds. It is enough that for any T that there exists at least one subset $S \subset T$ such that $\omega(S) > \omega(T)$. And this is one of the properties[1] of the arithmetic mean of edge weights that we have used in the previous chapter to evaluate the basic k-clique matching algorithm.

From that we can infer that the distribution of the sizes of the cliques created by the extended matching algorithm will largely depend not only on the value of K but also on the function chosen to compute clique weights. To illustrate that, we have conducted our experiments on the family of functions defined as follows:

$$\omega_x(H) = \frac{\sum_{e \in E(H)} w(e) + \left(\binom{k_{max}}{2} - |E(H)|\right) \cdot x}{\binom{k_{max}}{2}},$$

where H is a clique whose number of nodes is $|V(H)| \in K$ and $x \in [0,1]$. The

[1] The proof that indeed for cliques computed as arithmetic mean of edge weights this property takes place see Appendix B

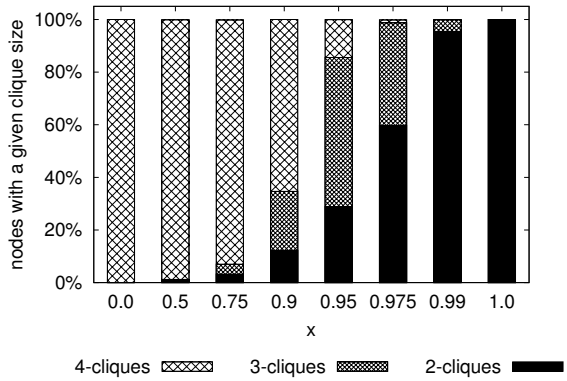


Figure 3.3: Clique size distribution (expressed as a percentage of nodes with given clique size).

equation is a computation for which we extend clique H to a clique of k_{max} nodes and all edges that were absent in the original clique H receive weight x. The weight of clique H is then the arithmetic mean of the edge weights in this extended clique. Therefore, if there are two cliques S and T such that $S \subset T$, the nodes that are in T but are absent from S have to contribute their incident edges that on average have at least weight x. Depending on the value of x, the probability that $\omega_x(T)$ is greater than $\omega_x(S)$ will vary. For example, when $x = 0$ $\omega_0(T) > \omega_0(S)$ for any T such that $S \subset T$ because all edges in the graph have weights from the $(0,1)$ interval. On the other hand, when $x = 1$, for any T such that $S \subset T$ we have $\omega_1(T) < \omega_1(S)$.

In Figure 3.3, we have plotted the distribution of nodes with respect to the size of the clique they belong to. The value of K is set globally to $K = \{2,3,4\}$. When $\omega_{x=1}()$ is used, only cliques of size 2 are created, while if we use $\omega_{x=0}()$ only cliques of size 4 are created.

In Figure 3.4, the number of rounds needed by the extended algorithm is plotted against different values of x used to compute clique weights with the $\omega_x()$ function. We observe that the time necessary for the network to converge (measured in rounds) remains almost unchanged irrespectively of the value of x. Moreover, the results do not differ much from the results obtained for the basic k-clique matching algorithm.

Additionally, we observe that the target distribution of clique sizes does not effectively affect the convergence time. The only exception is for very high values of x, resulting in 2-cliques dominating the network, which, as we have seen in Chapter 2, converges significantly faster.

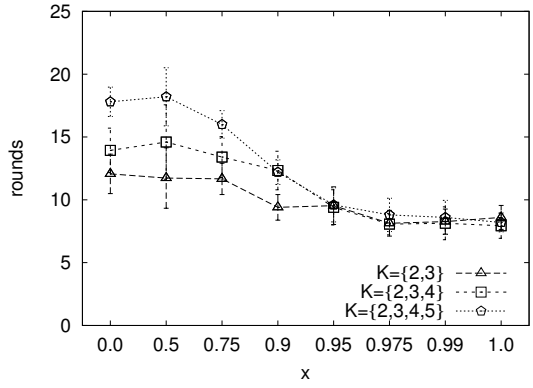


Figure 3.4: Number of rounds needed for the network to converge as a function of parameter x.

3.2. MULTIPLE CLIQUES PER NODE MATCHING

Our second extension of the basic algorithm removes the constraint of a single clique per node. Instead, each node v can set its own limit, $b[v]$, on the number of cliques it wants to be part of. Thus, the new problem we want to solve is to find a set of k-cliques, M, such that each node v belongs to at most $b[v]$ cliques from M and such that the total weight of the set M is maximized. If $k = 2$, this problem is known in the literature as *weighted b-matching* problem. Correspondingly, for any $k \geq 2$ we will refer to our newly defined problem as *weighted k-clique b-matching* problem.

3.2.1. The Algorithm

The number of cliques that node v is interested in participating in is saved as a constant $b[v]$ and instead of variable $\mathbf{C}_v$ that held a set of $k-1$ neighbors with which node v wished to create a clique, there is now the variable $\mathbf{LC}_v$ that holds a list of sets with $k-1$ neighbors each. The length of this list is limited by the constant $b[v]$, which can be kept secret from all other nodes in the network and can also be freely and seamlessly adjusted during the algorithm's execution if needed.

The main change in the algorithm is related to the way in which the list of cliques is constructed. In the basic algorithm, the task of the main loop was to find the possibly best available clique. Now in the main loop, node v has to find the best $b[v]$ available cliques. This is done by comparing the *attractiveness* of each considered combination of $k-1$ neighbors to the attractiveness of the least attractive set of $k-1$ neighbors currently in L. If the considered combination is

Constants:

k	size of cliques; globally fixed
$b[v]$	maximum number of cliques node v wants to participate in

Variables:

$\mathbf{LC}_v$	a list of sets of $k-1$ neighbors each with which v wants to create a clique
$\mathbf{w}_v$	the minimum of clique weights $w(\{v\}+C)$ such that $C \in \mathbf{LC}_v$

Active thread:

1: **loop**
2: $L \leftarrow \{\}$
3: **for all** $U \equiv \{u_1, \ldots, u_{k-1}\} \subseteq N(v)$ **do**
4: **if** $attr_v(U) > \min(attr_v(C) : C \in L)$ **then**
5: $L \leftarrow L + U$
6: **if** length $(L) > b[v]$ **then**
7: removeMin(L)
8: $\mathbf{LC}_v \leftarrow L$
9: **if** length $\mathbf{LC}_v = b[v]$ **then**
10: $\mathbf{w}_v \leftarrow \min(w(\{v\}+C) : C \in \mathbf{LC}_v)$
11: **if** length $\mathbf{LC}_v < b[v]$ **then**
12: $\mathbf{w}_v \leftarrow -\infty$
13: **send** $\mathbf{w}_v$ **to** all $u \in N(v)$

Passive thread:

1: **loop**
2: **receive** $\mathbf{w}_u$ **from** any $u \in N(v)$
3: **store** $\mathbf{w}_u$ **locally**

Figure 3.5: Self-stabilizing weighted k-clique b-matching algorithm (executed by node v).

more attractive, it is added to list L. Additionally, if due to this new addition the length of list L increased over $b[v]$, the least attractive set of $k-1$ neighbors is removed from L to keep the list in the $b[v]$ limit.

The contents of the messages communicated by the nodes has not changed, nodes still send out only a single value, but its semantics have changed. It no longer indicates a node's interest in joining a specific clique (or lack of any clique, if the message contains $-\infty$), but rather advertises the *minimum* weight of a clique that a node would be interested in. Therefore, if the list $\mathbf{LC}_v$ is full, then the message contains the smallest weight among the cliques from this list. If the list has less than $b[v]$ cliques, then the message contains $-\infty$, indicating that any

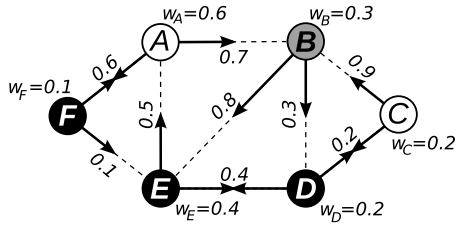
1: $M \leftarrow \emptyset$
2: **while** $E \neq \emptyset$ **do**
3: $\quad \langle u,v \rangle \leftarrow$ locally heaviest edge from E
4: $\quad M \leftarrow M + \{\langle u,v \rangle\}$
5: $\quad E \leftarrow E - \{\langle u,v \rangle\}$
6: $\quad$ **if** $|\{\langle u,z \rangle \in M : z \in V\}| = b[u]$ **then**
7: $\quad\quad E \leftarrow E - \{e' \in E : e' \text{ is incident to } u\}$
8: $\quad$ **if** $|\{\langle v,z \rangle \in M : z \in V\}| = b[v]$ **then**
9: $\quad\quad E \leftarrow E - \{e' \in E : e' \text{ is incident to } v\}$

Figure 3.6: Sequential greedy algorithm for weighted matching.

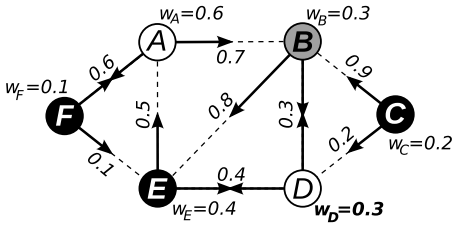
proper clique will be welcome.

To better understand what is the form of the final matching created by this algorithm, recall the greedy algorithm by Preis [Pre99] that we presented in the previous chapter. Preis's algorithm produces the same matching as that resulting from the execution of our basic algorithm for $k = 2$. In Figure 3.6, we present a modification of Preis's algorithm that raises the limit of the number of edges from the final matching that each node can be incident to. In brief, the algorithm starts with an empty set M where the final matching will be stored and a set E that contains all the edges from the graph. In a loop, a locally heaviest edge $\langle u,v \rangle$ is removed from all the edges stored currently in E and added to set M. Then, if by adding this edge to M we have increased the number of edges that are incident to one of the vertices of $\langle u,v \rangle$ to this vertex limit, $b[u]$ or $b[v]$ respectively, we remove from E all the edges incident to that vertex that still remain in E. These steps are then repeated until E becomes empty.

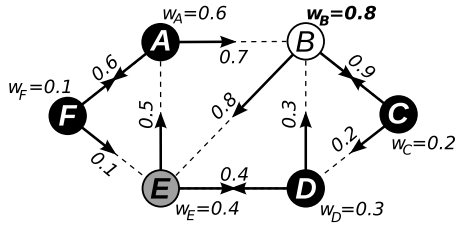
To better understand how the network converges to a k-clique b-matching consider a possible execution of the algorithm for $k = 2$ and $b = 2$ by six nodes depicted in Figure 3.7. At the beginning, every node has two random 2-cliques in its list and the values of variables $\mathbf{w}_v$ are set accordingly to the contents of these lists. Although there are in the network 2-cliques that are matched: AF, CD, and DE, the network is not in a stable state. Nodes E and F need to remove from their lists cliques that are non-proper, for example, E cannot point to A because the weight of clique AE is smaller than the current value of $\mathbf{w}_A$. Moreover, nodes B and D can improve their current choices of cliques. For instance, the value of $\mathbf{w}_C$ allows node B to choose clique BC, thus a list of cliques BC and BE is much more attractive for B than its current list of BD and BE. To make it easier to follow changes in the network, let's assume that only one node makes a move at a time. If D is the first to make a move (Figure 3.7(b)), then it can choose out of three possible cliques CD, BD, and DE. Each of these cliques is *proper*, therefore D



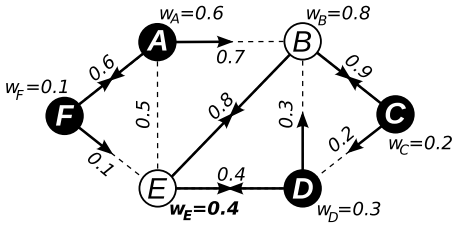
(a) initial state



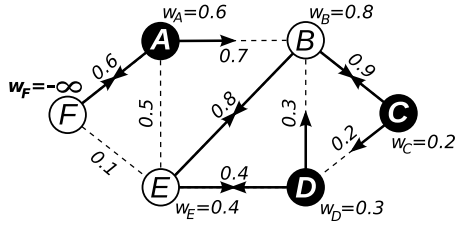
(b) D has moved



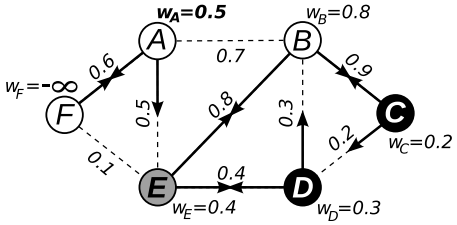
(c) B has moved



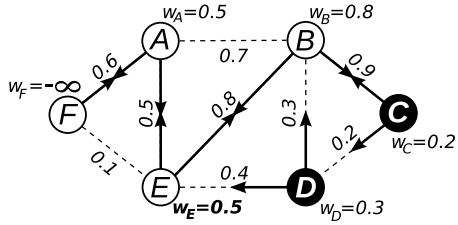
(d) E has moved



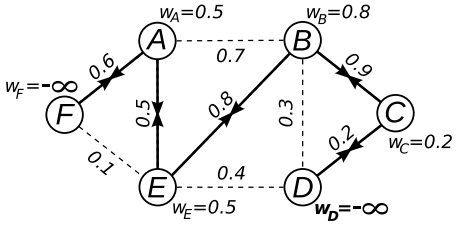
(e) F has moved



(f) A has moved



(g) E has moved



(h) D has moved

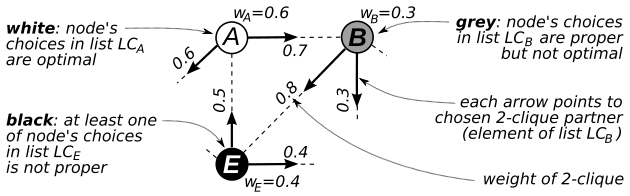


Figure 3.7: Step-by-step illustration of k-clique b-matching algorithm for $k = 2$ and global $b = 2$.

chooses the ones with highest weight BD and DE. As a consequence, node C has now one non-proper clique CD in its list ($w(C,D) < \mathbf{w}_D$), while clique BD became matched with B and D pointing to each other. Yet, clique BD falls apart as soon as B makes a move (Figure 3.7(c)). Out of four possible proper cliques, B chooses the two heaviest: BC and BE. Clique BD, whose weight is much smaller than the weight of those two cliques, is not attractive enough. Now, if node E makes a move (Figure 3.7(c)), it can choose out of four possible 2-cliques AE (weight 0.5), BE (weight 0.8), DE (weight 0.4), and EF (weight 0.1). All of these cliques are proper with the exception of AE, which is also in E's current list. Therefore, after executing a loop of the algorithm, node E chooses BE and DE as its new cliques.

Node F doesn't have much of a choice; there are only two 2-cliques incident to it, and one of them, EF, is non-proper. Thus, after making a move, it keeps in its list only one clique AF, and because its list is not full, F sets its $\mathbf{w}_F$ to $-\infty$. If node A makes a move now, it won't be able to keep clique AB in its list any longer, as this clique has become unavailable once B made a move. The only two proper cliques incident to A are AE and AF and they will now be in A's list. The change in the value of $\mathbf{w}_A$ results in E being able to improve its choice of cliques. Because $\mathbf{w}_A$ has decreased from 0.6 to 0.5, clique AE became *proper* for E. After execution of a single loop of the algorithm, E will now have in its list BE and AE, and the value of its variable $\mathbf{w}_E$ will increase from 0.4 to 0.5. Out of six nodes four have optimal possible cliques in their lists. The remaining two nodes C and D have in their lists at least one non-proper clique each: C has CD, whose weight is smaller than $\mathbf{w}_D$ and D has BD and DE, and both $w(B,D) < \mathbf{w}_B$ and $w(D,E) < \mathbf{w}_E$. Yet, once D moves (Figure 3.7(h)), and changes its list to contain the only proper clique CD, and sets its variable $\mathbf{w}_v$ to $-\infty$, C doesn't have to change anything about its list. The network is fully converged, without any nodes knowing the contents of other nodes' lists in the process.

3.2.2. Algorithm Properties

As before, we express the extended algorithm using guarded commands (see Figure 3.8). In fact, the entire algorithm still requires only one guarded command. The guard G_{mc} is composed as an alternative of six boolean expressions: ($G_{mc}0a$) and ($G_{mc}0b$) ensure that the list of cliques of node v is properly constructed, having no more than $b[v]$ sets with $k-1$ neighbors each; ($G_{mc}1a$) and ($G_{mc}1b$) guarantee that the value of $\mathbf{w}_v$ is correct; ($G_{mc}2$) makes sure that only proper cliques are in $\mathbf{LC}_v$; while ($G_{mc}3$) checks whether there exists a proper clique not present in $\mathbf{LC}_v$ whose weight is higher than the value of $\mathbf{w}_v$. If any of these guards evaluates to true, then commands ($S_{mc}1$) and ($S_{mc}2$) are executed. First one finds (at most) $b[v]$ best proper cliques, and the second one updates $\mathbf{w}_v$ accordingly to the new contents of $\mathbf{LC}_v$.

Constants:
k, $b[v]$

Variables:
$\mathbf{w}_v$; $\mathbf{LC}_v$

Actions:

$$\begin{array}{ll}
\exists_{C\in\mathbf{LC}_v}|C|+1\neq k & (G_{mc}0a)\\
\vee\ |\mathbf{LC}_v|>b[v] & (G_{mc}0b)\\
\vee\ \big(|\mathbf{LC}_v|=b[v]\wedge\mathbf{w}_v\neq\min(w(\{v\}+C):C\in\mathbf{LC}_v)\big) & (G_{mc}1a)\\
\vee\ \big(|\mathbf{LC}_v|<b[v]\wedge\mathbf{w}_v\neq-\infty\big) & (G_{mc}1b)\\
\vee\ \exists_{C\in\mathbf{LC}_v}\exists_{u\in C}\ w(\{v\}+C)<\mathbf{w}_u & (G_{mc}2)\\
\vee\ \exists_{U\subseteq N(v)\wedge U\notin\mathbf{LC}_v}\ \big(|U|+1=k & (G_{mc}3)\\
\qquad\qquad\wedge\forall_{u\in U}\ \mathbf{w}_u\leq w(\{v\}+U) & \\
\qquad\qquad\wedge\mathbf{w}_v<w(\{v\}+U)\big) & \\
\rightarrow\ \mathbf{LC}_v:=\big\{U\subseteq N(v):|U|+1=k\ \wedge\ \forall_{u\in U}\ \mathbf{w}_u\leq w(\{v\}+U)\big\} & (S_{mc}1)\\
\qquad\text{such that } |\mathbf{LC}_v|\leq b[v] \text{ and} & \\
\qquad\forall_{U\subseteq N(v)\wedge U\notin\mathbf{LC}_v}\ \big(|U|+1=k\ \wedge\ \forall_{u\in U}\mathbf{w}_u\leq w(\{v\}+U)\big) & \\
\qquad\qquad\Rightarrow\big(|\mathbf{LC}_v|=b[v] & \\
\qquad\qquad\qquad\wedge\ \forall_{C\in\mathbf{LC}_v}\ w(\{v\}+C)>w(\{v\}+U)\big) & \\
\mathbf{w}_v:=\left\{\begin{array}{ll}\min(w(\{v\}+\mathbf{C}_v):C\in\mathbf{LC}_v) & |\mathbf{LC}_v|=b[v]\\ -\infty & |\mathbf{LC}_v|<b[v]\end{array}\right. & (S_{mc}2)
\end{array}$$

Figure 3.8: k-clique matching algorithm in guarded commands formalism.

If we compare this guarded command to the one provided for the basic algorithm (see Figure 2.6), we can see that the main structure remains unchanged:

	$(G0)$	if $\mathbf{C}_v$ has incorrect size,
$\vee$	$(G1)$	or if $\mathbf{w}_v$ holds incorrect value,
$\vee$	$(G2)$	or if (any) $\mathbf{C}_v$ is not proper,
$\vee$	$(G3)$	or it a better alternative exists,
$\rightarrow$	$(S1)$	recompute $\mathbf{C}_v$, and
	$(S2)$	recompute $\mathbf{w}_v$

The only difference is that now instead of a list that could store at most one set of $k-1$ neighbors, node v can keep a list with up to $b[v]$ sets of $k-1$ neighbors. In the basic algorithm, the fact that each node indeed keeps a list of cliques has been obscured by using variable $\mathbf{C}_v$ that stored either a set of $k-1$ neighbors, when there was one element in the list, or an empty set, if the list was empty. Now the use of lists is made explicit. Moreover, we have also made a clear distinction between the values of $\mathbf{w}_v$ depending on the saturation of the list $\mathbf{LC}_v$; if the list is full then $\mathbf{w}_v$ holds the minimum weight of cliques from $\mathbf{LC}_v$, if there are less than $b[v]$

sets in $\mathbf{LC}_v$, then $\mathbf{w}_v$ is set to $-\infty$. The differences between other guards and commands encompass only the fact that the value of constant $b[v]$ can be now larger than 1, for example, instead of finding set U that creates the heaviest proper clique with node v in command ($S1$), we are looking for $b[v]$ heaviest proper cliques in command ($S_{mc}1$).

All changes of $\mathbf{C}_v$ into $\mathbf{LC}_v$ are only cosmetic, and nodes remain unaware of the possible sizes of other nodes' lists. Moreover, $\mathbf{w}_v$ remains the only value communicated between nodes, and the principles on which this variable is updated by $\mathbf{w}_v$, as well as the rules by which $\mathbf{w}_v$ is used by v's neighbors stay the same. As a consequence, proving all the properties of self-stabilization, correctness, stability, uniqueness, and approximation factor can be conducted in the same way as it was done for our k-clique matching algorithm. For completeness we provide here the main lemmas and theorems.

Lemma 3.8. *(Closure) The clique matching algorithm as defined in Figure 3.8 is closed with regard to predicate P_{mc}:*

$$
\begin{aligned}
P_{mc}: \quad \forall_v \Big(& \forall_{C\in\mathbf{LC}_v} |C|+1=k \\
& \wedge\ |\mathbf{LC}_v| \le b[v] \\
& \wedge\ \big(|\mathbf{LC}_v| = b[v] \Rightarrow \mathbf{w}_v = \min(w(\{v\}+C) : C \in \mathbf{LC}_v)\big) \\
& \wedge\ \big(|\mathbf{LC}_v| < b[v] \Rightarrow \mathbf{w}_v = -\infty\big) \\
& \wedge\ \forall_{C\in\mathbf{LC}_v} \forall_{u\in C}\ w(\{v\}+C) \ge \mathbf{w}_u \\
& \wedge\ \nexists_{U\subseteq N(v)\wedge U\notin\mathbf{LC}_v} \big(|U|+1=k \\
& \qquad\qquad \wedge\ \forall_{u\in U}\ w(\{v\}+U) \ge \mathbf{w}_u \\
& \qquad\qquad \wedge\ \mathbf{w}_v < w(\{v\}+U)\big)\Big)
\end{aligned}
$$

Lemma 3.9. *(Convergence) Under a fair distributed daemon, the clique matching algorithm for a given limits of cliques per node converges in at most $2|M|+1$ rounds, where one round is the minimum time span in which each node that was enabled at the beginning of a round either made at least one move or became disabled at some point.*

Theorem 3.10. *(Self-Stabilization) Following from Lemmas 3.8 and 3.9 the multiple clique matching algorithm is self-stabilizing with regard to predicate P_{mc}.*

Lemma 3.11. *(Correctness) In a safe state, $M = \{Q_k : \exists_{v\in V} \exists_{C\in\mathbf{LC}_v} V(Q_k) = \{v\}+C\}$ forms a correct k-clique b-matching.*

Lemma 3.12. *(Stability) In a safe state, for each k-clique Q_k we have that if Q_k does not belong to the k-clique b-matching M generated by the algorithm, then there must exist at least one node $v \in Q_k$ such that $\mathbf{w}_v > w(Q_k)$.*

Lemma 3.13. *(Uniqueness of Solution) If M_1 and M_2 are both k-clique b-matchings arising from the execution of the algorithm on the same graph, then $M_1 = M_2$.*

The idea for the proof of the following theorem on an approximation factor is exactly the same as for the proof of Theorem 2.10 on $1/k$ approximation factor of the basic k-clique matching algorithm. We are going to show that there exists a function f that assigns to every clique Q_k^* from the optimal matching M^* a clique Q_k from the k-clique b-matching M returned by the algorithm such that:

- Q_k^* and Q_k overlap,
- Q_k is of equal or higher weight than Q_k^*,
- Q_k is assigned to at at most k cliques from optimal matching M^*.

Finding a function that has the two first properties does not pose much difficulty, and the main challenge lies now in showing that we can find a function f that also possesses the third property, because in a k-clique b-matching, for any clique Q_k from M there can be up to $\sum_{v \in Q_k} b[v]$ cliques in the optimal matching M^* overlapping with it. Yet, before we show how to construct function f, let us first make the following observation related to the first two properties:

Observation 3.14. *In a safe state, for each k-clique Q_k^* that belongs to the optimal matching we have that either Q_k^* belongs to the k-clique b-matching M generated by the algorithm, or there must exist at least one node $v \in Q_k^*$ such that $\mathbf{w}_v > w(Q_k^*)$. In the latter case, the size of $\mathbf{LC}_v$ is equal to $b[v]$ which means there are $b[v]$ k-cliques in M overlapping with Q_k^* whose weight is larger than weight of Q_k^*.*

Theorem 3.15. *(1/k Approximation Factor) A k-clique b-matching from a safe state has a total weight that is at most a factor k worse than the weight of the optimal k-clique b-matching M^*.*

Proof. Let's partition M^* into two disjoint sets: $M^* \cap M$ and $M^* \backslash M$ and let us first focus only on the latter set $M^* \backslash M$. Let $g : M^* \backslash M \to V$ be a function that to each $Q_k^* \in M^* \backslash M$ assigns such a v that $|\mathbf{LC}_v| = b[v]$ and $\mathbf{w}_v > w(Q_k^*)$. The existence of such a v is guaranteed by Lemma 3.12 (see Observation 3.14). Observe that for any $v \in g(M^* \backslash M)$:

$$
\begin{aligned}
&|g^{-1}(v)| + |\{Q_k^* : Q_k^* \in M^* \cap M \wedge v \in Q_k^*\}| \\
&\leq |\{Q_k^* : Q_k^* \in M^* \backslash M \wedge v \in Q_k^*\}| + |\{Q_k^* : Q_k^* \in M^* \cap M \wedge v \in Q_k^*\}| \\
&= |\{Q_k^* : Q_k^* \in M^* \wedge v \in Q_k^*\}| \\
&\leq b[v] \\
&= |\mathbf{LC}_v| \\
&= |\{\{v\} + C : C \in \mathbf{LC}_v \wedge \{v\} + C \in M^* \backslash M\}| \\
&\quad + |\{\{v\} + C : C \in \mathbf{LC}_v \wedge \{v\} + C \in M^* \cap M\}|.
\end{aligned}
$$

The first inequality comes from the definition of function g, the second inequality is directly related to the definition of k-clique b-matching, and two rightmost

equalities come from our assumption about v. From this sequence of dependencies and the fact that $|\{Q_k^* : Q_k^* \in M^* \cap M \wedge v \in Q_k^*\}| = |\{\{v\}+C : C \in \mathbf{LC}_v \wedge \{v\}+C \in M^* \cap M\}|$, we have $|g^{-1}(v)| \leq |\{\{v\}+C : C \in \mathbf{LC}_v \wedge \{v\}+C \in M^* \backslash M\}|$.

This means that there exists a set of functions $h_v : g^{-1}(v) \rightarrow \{\{v\}+C : C \in \mathbf{LC}_v \wedge \{v\}+C \in M^* \backslash M\}$ where each h_v is an injection and for any $Q_k^* \in g^{-1}(v)$ we have that $w(Q_k^*) < w(h_v(Q_k^*))$.

We will now define function $f : M^* \rightarrow M$, using the previously defined functions g and h:

$$f(Q_k^*) = \begin{cases} h_{g(Q_k^*)}(Q_k^*) & \text{if } Q_k^* \in M^* \backslash M \\ Q_k^* & \text{if } Q_k^* \in M^* \cap M \end{cases}$$

Such an f assigns to each $Q_k^* \in M^*$ a k-clique from M such that $w(Q_k^*) \leq w(f(Q_k^*))$. Moreover, for any $Q \in M$, $|f^{-1}(Q)| \leq k$, because either $Q \in f(M^* \cap M)$ and there is only one $Q_k^* \in M^* \cap M$ such that $f(Q_k^*) = K$, or $Q \in f(M^* \backslash M)$ and there are at most k different functions h_v where $v \in K$ yet each of them is an injective function.

Having defined f, we can now show that $w(M) \geq \frac{1}{k} w(M^*)$ in exactly the same way as be did for the basic algorithm in Theorem 2.10:

$$\begin{aligned} w(M^*) &= \textstyle\sum_{Q_k^* \in M^*} w(Q_k^*) \\ &\leq \textstyle\sum_{Q_k^* \in M^*} w(f(Q_k^*)) \\ &\leq k \cdot \textstyle\sum_{Q_k \in f(M^*)} w(Q_k) \\ &\leq k \cdot \textstyle\sum_{Q_k \in M} w(Q_k) \\ &= k \cdot w(M). \end{aligned}$$

□

3.2.3. Experimental Evaluation

The most interesting aspect of this extension's performance is the impact of the number of cliques that each node can be part of on the convergence. We use here the simulation setup identical to those in previous evaluations: a network of 300 nodes that create a complete graph, the weight of an edge between any two nodes is drawn from a uniform distribution over the interval $(0, 1)$. Moreover, we set up globally the number of cliques that each node can participate in.

Figure 3.9 shows the convergence times measured in number of rounds for various sizes of cliques and for various lengths of the clique lists. In each new simulation setup we double the number of cliques, b, each node is allowed to be part of starting from 1 through 2, 4, 8, 16 up to 32. When $b = 1$, the algorithm reduces itself to the basic algorithm for k-clique matching. What can be observed from the graph is that the increase in the value of b does not cause deterioration in the convergence times. For 2-cliques the convergence time stays almost constant for any value of b tested. For 3- and 4-cliques the number of rounds increases only

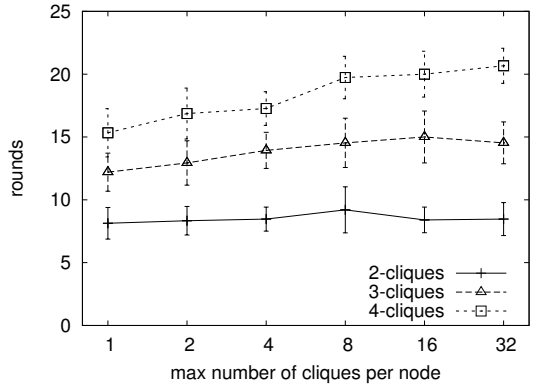


Figure 3.9: Convergence of extended k-clique matching that allows multiple cliques per node — in rounds for different sizes of clique lists.

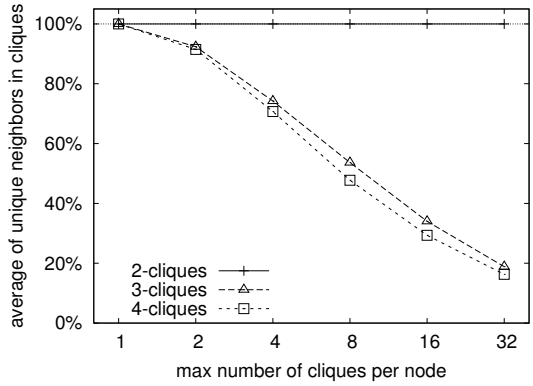


Figure 3.10: The average percentage of unique neighbors in individual clique lists in the converged state.

by a few, while the value of b increases from 1 to 32. This suggests that our bound on the convergence time provided in Section 3.2.2 equal to the $2|M|+1 \leq 2\frac{b\cdot|V|}{k}+1$) is very conservative and is often much higher than the actual convergence times.

In our k-clique b-matching algorithm, when nodes fill up their lists of cliques, they are not concerned with chosen cliques overlapping each other. The sole determinants of whether a clique should be included in a node's list, are the clique's weight and the minimum weight values reported by the clique members which control if the clique is proper. Thus, it is not surprising that for $k > 2$ the average percentage of neighbors that are in only one of the cliques in a particular node's

Constants:

K	possible sizes of cliques; globally fixed (or K_v specific to the given node v)
$b[v]$	maximum number of cliques node v wants to participate in

Variables:

$\mathbf{LC}_v$	a list of sets of $k-1$ neighbors with which v wants to create a clique, where each $k \in K$ (or $k \in K_v$, respectively)
$\mathbf{w}_v$	the minimum of clique weights: $w(\{v\}+C)$ such that $C \in \mathbf{LC}_v$

Active thread:

```
1:  loop
2:    L ← {}
3:    for all k ∈ K do
4:      for all U ≡ {u_1, ..., u_{k-1}} ⊆ N(v) do
5:        if attr_v(U) > min(attr_v(C) : C ∈ L) then
6:          L ← L + U
7:          if length (L) > b[v] then
8:            removeMin(L)
9:    LC_v ← L
10:   if length LC_v = b[v] then
11:     w_v ← min(w({v} + C) : C ∈ LC_v)
12:   if length LC_v < b[v] then
13:     w_v ← −∞
14:   send w_v to all u ∈ N(v)
```

Passive thread:

```
1:  loop
2:    receive w_u from any u ∈ N(v)
3:    store w_u locally
```

Figure 3.11: Self-stabilizing weighted various-sized clique b-matching algorithm (executed by node v).

list is below 100% for $b > 1$. In Figure 3.10, we can see that for $k = 2$, the percentage of unique neighbors in individual clique lists is exactly 100% for any b (any lower percentage for $k = 2$ would indicate that the cliques in a node's list are not unique, which is impossible). Yet, for $k > 2$ this percentage decreases with the increase of the list sizes, indicating growing overlaps between formed cliques.

3.3. COMBINING BOTH EXTENSIONS INTO ONE ALGORITHM

Nothing stands in the way to combine the two extensions together allowing nodes to be part of many cliques of various sizes (see Figure 3.11). Such an algorithm retains all of the properties: self-stabilization of the algorithm, correctness, stability, uniqueness, and $\frac{1}{k_{max}}$-approximation factor of the solution.

3.4. RELATED WORK

3.4.1. $\mathcal{G}$-Packings

Our first extension of the k-clique matching algorithm that allows cliques of various sizes to be established, produces as an outcome a set of disjoint cliques whose sizes are constrained to the values from set K. We will refer to such a set as a K-clique matching, which is a special case of a $\mathcal{G}$-packing. As defined for unweighted graphs by Hell and Kirkpatrick in [HK84], a $\mathcal{G}$-packing is a set M of vertex-disjoint subgraphs of a graph where each subgraph is isomorphic to some graph from the family $\mathcal{G}$ and the size of M is defined as $|V(M)|$. Moreover, as proved in [HK84], for *unweighted* graphs, there exists a polynomial time algorithm for finding a maximum $\mathcal{G}$-packing if $\mathcal{G} \subseteq \{Q_2, Q_3, Q_4, \ldots\}$ contains Q_2 (maximum K-clique matching for $K \subseteq \{2, 3, \ldots\}$ if K contains 2), and that $\mathcal{G}$-packing is NP-complete for any other $\mathcal{G} \subseteq \{Q_3, Q_4, \ldots\}$. Independently Cornuejols et al. provided the same results in [CHP82], along with a polynomial algorithm for finding maximum $\mathcal{G}$-packing if $\mathcal{G}$ consists of Q_2 and of a family of graphs[2] whose every subgraph of all but one vertices has a perfect matching. They also mentioned how this algorithm can be modified to find for such $\mathcal{G}$'s maximum weighted $\mathcal{G}$-packing, but only for graphs with weighted vertices, and not weighted edges.

For finding clique packings in general edge-weighted graphs, various sequential algorithms exist that find exact solutions, e.g., [DP94; GW89], or approximated ones, e.g., [dABR92]. Such algorithms found many applications, among others, in flight gate scheduling [DJP08; DJP12], biology [KGAW05], political and social sciences, or group technology of flexible manufacturing systems [WAGK06]. Yet, from our perspective the more interesting ones are distributed approaches.

Distributed clique partitioning algorithms find their natural application in multi-agent systems. For example, Shehory and Kraus [SK98] propose a distributed algorithm for task allocation via agent coalition formation. This algorithm has

[2] Such graphs are called hypomatchable or factor-critical.

been later adapted also to multi-robot domains by Vig and Adams [VA06; VA07] and Service and Adams [SA11]. The algorithm is based on a greedy algorithm for solving the set partitioning problem which in this case is equivalent to the weighted clique partitioning problem in complete graphs. The initial setting for the algorithm consists of a system of agents with a vector of capabilities each and a list of tasks, T, which the agents are supposed to perform. To minimize the costs of executing the tasks, agents try to create coalitions of bounded size. The limit on the coalition size is motivated by growing communication and computation costs in larger coalitions. Before agents start creating any coalitions, they divide among themselves the collection of all possible coalitions to reduce the individual workloads. Then, in each round every agent computes the least costly coalition-task pair and shares its result with all other agents. The coalition with the smallest cost among all announced is unanimously chosen and this coalition is immediately created and task is removed from the pending list.

The weak point of the algorithm of Shehory and Kraus is the need for synchronization after each round, which in case of any differences in computational capabilities of the nodes or uneven distribution of workload related to computations of coalition-task costs, leads either to forcing agents to wait for the slowest agent to finish one round in order to all other agents to progress further or to the necessity of incorporating additional mechanisms to balance the workload in each round. The need for synchronization is hard to eliminate as agents have to make sure that each task is assigned to only one coalition. In our algorithm this problem does not exist, the communication is fully asynchronous and nodes can progress in their computations without waiting for other nodes. Moreover, instead of just one coalition per round, in our algorithm more than one coalition can simultaneously emerge. This comes at the price of being able to apply our algorithm to settings where each task needs to be assigned to exactly one coalition as assumed in [SK98][3]. Yet, our intended application is in the domains where the formation of the coalitions is not driven by the tasks that need to be assigned and performed but where the goal is to create long-term coalitions whose value depends on the characteristics and mutual relations between its members. Such domains will be present in multi-agent systems in which it is important that the coalitions created need to be suited to perform various tasks, thus the objective is to form coalitions in which combined capabilities of the agents provide the biggest flexibility.

Apart from the above differences related to the existence of task list, the algorithm of Shehory and Kraus [SK98] guarantees also a better approximation factor of $ln(k)$ than our algorithm does. Yet, this factor is specific to the properties of the coalition cost function adopted by the authors as a clique weight function.

[3] A workaround would be to create for each task a mock agent which then acts as one of the coalition members.

In [SA11], Service and Adams modify the algorithm from [SK98] to guarantee the approximation factor of k in terms of coalition's utility instead of coalition's cost acting as a clique weight function. This approximation factor is equal to our approximation factor, which is applicable for any arbitrary clique weight function where all weights are nonnegative.

Another fully distributed algorithm for coalition formation that is based on the idea of clique packing is by Tošić and Agha [TA05]. In their algorithm the formation of coalition is not driven by particular tasks. Instead, agents try to organize in coalitions of maximal possible (but bounded) sizes, such that any two agents in a coalition can communicate directly, or of highest possible value in terms of joint resources and capabilities of all coalition members. Agents start by exchanging their lists of neighbors and then proceed in a round based fashion (no agent can start another round unless all its neighbors have finished the previous one). In each round each agent that looks for a coalition chooses the most attractive clique among the ones still available and sends a message about it to all its neighbors. If it happens that all other members of this tentative clique have also sent a message with this clique in the same round, then in the next round all these agents announce that this coalition is formed. On the other hand, if one of the agent's neighbors becomes part of some coalition, all cliques that include this neighbor are removed from the list of a node's potential coalitions. Finally, based on its situation, an agent can abandon its current tentative clique, in favor for another one that is just as good (or even worse). By doing so, it agrees to never choose the abandoned clique again, which contributes to the timely termination of the algorithm. Unfortunately, the number of rounds necessary for the algorithm to terminate is in the order of $O(|V(G)|^{c+1})$, provided that the maximum degree of nodes is at most $c \cdot \log |V(G)|$. Moreover, the algorithm proposed by Tošić and Agha does not provide any guarantees with regard to the quality of the formed coalitions. As they cautiously admit, "it is easy to construct examples of the underlying graphs and the particular runs of the algorithm such that, once every agent joins a coalition and the algorithm terminates, several agents end up in trivial coalitions," which most likely is the result of haste manner in which agents abandon prospective clique. With respect to that, our algorithm has the advantages of both providing smaller upper bound on the number of rounds needed for convergence and delivering solutions which have guaranteed quality.

Worth mentioning is also the application of clique packing algorithms for node clustering in wireless ad hoc networks. Organizing nodes in wireless ad hoc networks in clusters serves multiple purposes including load balancing, fault-tolerance, extension of a network's lifetime, facilitating routing, and reduction of delay in communication. In comparison to many other clustering approaches based on construction of spanning trees, dominating sets or independent sets, the

advantage of clique packing is that it guarantees that each two nodes in a cluster are in each others communication range, which can further improve fault tolerance; e.g. even if a cluster head dies, the cluster remains connected and the role of the head can be assumed by any other node from the cluster.

A particular example of the clustering algorithm based on clique packing presented by Sun et al. in [SPNW06], additionally provides means to, firstly, prevent external attackers from participating in the process of cluster formation and, secondly, identify and remove from the network any internal malicious nodes that do not follow the algorithm semantics. In a benign environment, the algorithm requires only 5 steps to converge: (i) exchanging neighbor lists and computing a local maximum clique, Q_1, using a greedy heuristic, (ii) exchanging the prospective clique, Q_1, with neighbors and updating its own clique based on received information, adopting if possible better clique sent by a neighbor (iii) exchanging the updated clique, Q_2, and obtaining a final clique, Q_3, by removing from Q_2 any nodes that found a better clique, (iv) exchanging the final clique Q_3. If the final choices are inconsistent, the malicious nodes are identified and removed from neighbor lists and the algorithm is restarted. As we can see, the algorithm converges really fast. Moreover, it has low computational complexity due to the use of the heuristic to compute local maximum clique. Although our algorithm was not designed with wireless ad hoc networks in mind and does not offer as impressive convergence times or computational savings, it still could be used provided the network is sparse, in which case the computation of all possible cliques a node can create with its neighbors would not be too big of a burden. And the advantages of using our algorithm would be fault-tolerance due to the self-stabilization and the possibility to create clustering in which each cluster is a clique chosen based on some more detailed information than just direct connectivity between members, for example, pairwise strengths of the signal, node's degrees, transmission and battery power.

3.4.2. b-Matchings

Another type of generalization of the matching problem involves relaxation of the condition that the edges must be non-incident. Instead, each vertex v has a quota $b[v]$ for the number of neighbors it can be matched with. Thus, for weighted graphs we can define the maximum weighted b-matching problem as a problem of finding a subset S of edges from a graph such that each vertex v is incident to at most $b[v]$ edges and the total weight of the edges from S is maximized. For this problem the optimal solution can be found in polynomial time [Edm65; Edm70; MHS99] in a centralized setting.

In a distributed setting, and especially in the context of our approach (to solving maximum weighted b-matching), algorithms which need to be mentioned are

the algorithms proposed by Georgiadis and Papatriantafilou [GP10b; GP12]. Similarly to our algorithms, their algorithms are based on a greedy heuristic, which repeatedly selects a locally heaviest edge that does not conflict with already selected edges and does not violate the constraint of the number of selected edges per node. This greedy heuristic finds a $1/2$-approximation of the optimal maximum weighted b-matching.

The first version of the algorithm given by Georgiadis and Papatriantafilou [GP10b] is a generalization of the deterministic distributed weighted matching algorithm by Hoepman [Hoe04] and has been designed to work under strict assumptions of fault-free static environment, in which no messages are lost and no nodes join or leave the network. In short, the nodes construct a maximum weighted b-matching by sending two types of messages: PROP to propose formation of a matched edge and REJ to reject the offer. Each node sends PROP messages to the heaviest-weight neighbors and if two nodes exchange PROP messages, the edge between them becomes part of the final b-matching. At any time for each node v, the sum of already established matched edges and remaining PROP messages awaiting reply cannot exceed $b[v]$. Moreover, once the number of established matched edges reaches $b[v]$, v will send only REJ massages to all its remaining neighbors. The modified version of the first algorithm [GP12] lessens the assumptions, allowing for dynamic networks in which nodes join, leave or where the weights change. This is accomplished by extending the communication model such that nodes can break previously established matched edges and form new ones by sending PROP (REJ) messages to the neighbors to which REJ (PROP) had been previously sent, or a special WAKE message. Although this algorithm can handle the joins and leaves, it is still susceptible to transient faults of communication links. For example, if two messages sent to the same neighbor PROP and REJ become reordered, it can affect the decisions of many nodes in the network resulting in a b-matching of a lesser quality. Therefore, our algorithm can be more attractive in environments in which nodes can face transient faults.

On the other hand, despite the differences in the communication model, the weighted b-matching created by Georgiadis and Papatriantafilou's algorithms is identical to the weighted b-matching created by our algorithm. Which means that our algorithm can be easily applied to the many-to-many matching with preferences[4] discussed in [GP10b; GP12], yielding the same solution but offering better

[4] In spite of the similarity in the name, weighted b-matching and many-to-many matching (also referred to as b-matching [Mat08] or stable fixtures [IS07] problem) with preferences are different problems. In many-to-many matching with preferences, edges are unweighted; instead, each node has a preference ranking of its neighbors. The typical goal is to find a subset S of edges that would be stable from the point of nodes preferences, i.e. outside of S there should be no edge (v,u) such that both v and u would prefer to be matched together having a matched edge with another neighbor included in S. Such a stable subset S might not exist in a graph with arbitrary

behavior if transient faults of communication links occur.

A different approach to finding a maximum weighted b-matching is proposed by Panconesi and Sozio [PS10]. They designed a randomized distributed $(6+\varepsilon)$-approximation algorithm that requires $O(\frac{\log^3 |V(G)|}{\varepsilon^3} \log^2 \frac{C_{max}}{C_{min}})$ rounds provided all edge weights are in the $[C_{min}, C_{max}]$ interval. Although they offer a better convergence time complexity, the tradeoff lies in the worse approximation factor in comparison with our algorithms and algorithms by Georgiadis and Papatriantafilou.

Note on another possible application of distributed weighted b-matchings algorithms

We have already mentioned peer-to-peer algorithms such as Vicinity [Vou06] and T-Man [JMB09] pointing out to the similarities between overlays created by these algorithms and the k-clique matchings produced by our basic algorithm from previous chapter. Now, in the context of our second extension that allows more than one clique per node, we can see that the overlays produced by those algorithms bear even stronger resemblance to the stable 2-clique b-matchings. To recap, in the networks where Vicinity or T-Man run, each node gradually creates a list of s neighbors which are chosen from the entire population of nodes based on some proximity or ranking function. Similarly, if we choose $k = 2$ and the number of cliques per node, b, to be equal exactly s, then the 2-clique s-matching produced by our second extension can be interpreted as an overlay in which every node also has at most s neighbors in its list, which have been chosen based on some proximity function (weights). The only difference between this overlay and those of Vicinity and T-Man is that in the first overlay the created links are guaranteed to be symmetric, while for the latter this doesn't have to be the case. In applications where reciprocity between nodes is expected this difference can be crucial and act in favor of our algorithm.

3.4.3. K-clique b-Matchings

The problems of maximum weighted K-clique matchings and maximum weighted b-matchings can be seen as special cases of an even more general problem. We will refer to such a problem as a maximum weighted K-clique b-matching. In a distributed setting, to solve the problem of finding a maximum weighted K-clique

preferences which can form cycles. Therefore, Georgiadis and Papatriantafilou attack the problem from a different angle and aim for maximizing the total satisfaction of the nodes with their choices. Based on the preferences, they compute the edge weights such that the weight expresses the level of satisfaction of nodes at its ends. Next, they use these weights as an input to their algorithm and find an approximation of the maximum weighted b-matching which can be translated back to the approximation solution of the many-to-many matching with references.

b-matching one can utilize the distributed randomized α-approximation algorithm for fractional packing proposed by Kuhn et al. in combination with distributed randomized rounding [KW05; KMW06]. This produces an $O(\alpha k_{max})$-approximation for the maximum weighted K-clique b-matching in an expected logarithmic number of rounds with respect to the total number of cliques. Another approach can be to use the randomized distributed algorithm for maximum weighted b-matching in hypergraphs by Koufogiannakis and Young [KY09; KY11]. In such a case, a hypergraph $H = (V, E')$ is constructed out of the original graph $G = (V, E)$ such that each clique from G of size in K is interpreted as a hyperedge in H. The expected running time of the algorithm is $O(\log^2 |E'|)$, which is a logarithmic factor slower than the algorithm by Kuhn et al., but its approximation factor is $O(k_{max})$ which is substantially better. Although both of these algorithms offer better (expected) running times in comparison to our algorithm, they are fairly complicated, and their operation can be easily hampered by transient faults, from which our self-stabilizing algorithm can swiftly recover.

3.5. CONCLUDING REMARKS

In the previous chapter, we have presented a self-stabilizing distributed $1/k$-approximation algorithm for finding a maximum weighted k-clique matching. In this chapter, we modified this basic algorithm in such a way as to allow nodes greater freedom in deciding about how many cliques and of what sizes they want to participate in; i.e., we transformed the algorithm into a self-stabilizing distributed $1/k$-approximation algorithm for a maximum weighted K-clique b-matching problem.

As the modifications necessary to let nodes create cliques of various sizes and keep more than one clique are completely independent, we discussed each of them separately before we combined them into one algorithm. First, we provided an algorithm to address the issue of variable-sized clique, i.e., finding a maximum weighted K-clique matching. We also gave an explanation why the new algorithm retains all of the basic algorithm's properties of self-stabilization, correctness, stability and uniqueness of solution, as well as, why the approximation factor of modified algorithm became $1/k_{max}$. Furthermore, we presented results of some simulations focusing on the differences in convergence times and distributions of the sizes of formed cliques when various functions to compute clique weights are used. Among other things, our experiments showed that the network tends to converge slightly faster if the clique weight function favors smaller cliques. We followed the same suit, when we presented the algorithm for a maximum weighted k-clique b-matching problem. First, we provided the motivation as to why all

the properties, including the $1/k$-approximation factor, still hold for the modified algorithm. Subsequently, we discussed simulation results, which are especially encouraging with regard to the number of rounds needed for convergence which seem barely affected by the maximum number of cliques per node. Finally, we combined the two algorithms.

In the end, the basic k-clique matching algorithm underwent in this chapter only cosmetic changes, yet these were powerful enough to invite great flexibility. The computational complexity of the algorithm did not increase, and the final algorithm can now solve a much wider range of problems. This became especially evident in the previous section, where we discussed other distributed algorithms for solving maximum weighted K-clique matching or b-matching algorithms, and mentioned their intended applications in the fields of multi-agent systems (coalition formation), wireless ad hoc networks (clustering) or peer-to-peer networks (topology construction to facilitate searching or resource sharing).

CHAPTER 4

Heuristics, Pruning, and Gossiping

In some applications of our k-clique matching 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 algorithms from Chapters 2 and 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 algorithm 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.

In this chapter we discuss heuristics and pruning, partial views, and gossiping, each of which has the potential to solve one of the above issues. We present our

research using k-clique matching algorithm as a basis, although all introduced methods can be easily applied to all of the extensions of this algorithm from the previous chapter.

4.1. HEURISTICS FOR FINDING A K-CLIQUE

To circumvent consideration of all possible $k-1$ neighbor combinations, and thus, to reduce the high computational complexity of a single round, each node can use a heuristic to find its most attractive k-clique. Naturally, this replacement of the exact algorithm for finding the heaviest proper clique by a heuristic can result in the degradation of the protocol's performance, including the increase of convergence time and the reduction of the final clique matching quality. Which aspects of the algorithm's performance will be affected, will largely depend on the properties of the heuristic used. We discuss this issue later on in this chapter providing specific heuristics as examples. First, we explain how our algorithm must be modified in order to preserve its correctness irrespective of the employed heuristic.

4.1.1. Algorithm Preparation

In the high-level representation of the basic k-clique matching algorithm from Figure 2.4, we distinguish two operations: (i) finding the maximum-weight proper k-clique whose contents and weight are then assigned to variables $\mathbf{C}_v$ and $\mathbf{w}_v$, and (ii) sending the value of $\mathbf{w}_v$ to all neighbors:

1: **loop**
2: $C \leftarrow$ EXACTMAXPROPERCLIQUE$(k, v, G[\{v\} + N(v)], S)$
3: $\mathbf{C}_v \leftarrow C$
4: $\mathbf{w}_v \leftarrow w(\{v\} + \mathbf{C}_v)$
5: **send** $\mathbf{w}_v$ **to** all $u \in N(v)$

To find the new value of variables $\mathbf{C}_v$ and $\mathbf{w}_v$ a function is used that returns the current heaviest proper k-clique if given as an input the size of the clique, k, the node for which the clique has to be found, v, its set of neighbors $N(v)$ with accompanying information about all relevant edge weights, and the neighbors' values of variables $\mathbf{w}$ constituting part of the system's state S. Using a function that returns the exact solution for the problem of finding the heaviest proper k-clique, takes care, as a by-product, also of other tasks, which are crucial for the correctness and self-stabilization of the algorithm. Firstly, it guarantees that in each round, the clique found in the previous round is reconsidered, and if that clique is proper, the newly chosen clique will be at least as heavy, which prevents the degradation

of the quality of each node's clique[1]. Secondly, if one of node v's neighbors has found a clique which according to v's current knowledge is proper, v will choose a different proper clique in its current round only if it is heavier than the one found by the neighbor. Those two properties are sufficient to achieve the convergence to a correct k-clique matching.

Yet, these properties may be absent in an arbitrary heuristic for finding the heaviest proper k-clique, which is only required to return in finite time a value that is semantically correct, i.e., it is a subset of $k-1$ neighbors or an empty set. Firstly, in each execution, the heuristic can investigate a different portion of the solution space, returning each time a different clique, possibly worse than the previously found one. Such a behavior can be found in most heuristics that incorporate some level of randomization to avoid getting stuck in the local optimum. Secondly, some heuristics restrict their search only to a portion of the solution space and even if executed infinitely over and over again they will never explore it entirely. As a result, it is possible that one node chooses an optimal proper clique that will never even be considered by other members of this clique and, thus, the nodes' individual choices will never agree and will never form a correct k-clique matching. Therefore, if we simply replaced EXACTMAXPROPERCLIQUE$(k, v, G[\{v\} + N(v)], S)$ with a heuristic function, a network in which such an algorithm is executed might be unable to ever reach a safe state, stay in the safe state once it is reached, or form and maintain any correct k-clique matching at all.

Nonetheless, even if we treat a given heuristic as a black box, we can still adapt the basic k-clique matching algorithm in such a way that even if this heuristic is used in place of the exact function, the algorithm is bound to eventually achieve and maintain a correct k-clique matching. In order to do that we will incorporate the two discussed properties directly into the algorithm, separately from the heuristic. Firstly, to avoid discarding a good clique candidate found in the previous round when a heuristic is used, in the modified protocol (see Figure 4.1) this clique is re-evaluated (lines 2–4). If this clique is still proper, it will be kept if no better clique is found.

Secondly, not only does node v send out to all its neighbors the information about the weight of its currently pursued clique (line 13) but it also informs all members of this clique of the clique's exact content (lines 14–15), which can be interpreted as an offer to join a particular clique. As all v's neighbors follow the same rules, node v receives full information about cliques in which v is contained and which are pursued by its neighbors. Thanks to that additional knowledge, v is able to improve its own clique choice (lines 5–8), even if on its own it would

[1] Naturally, if the clique from the previous round is not proper anymore, then the newly chosen clique, if any proper clique still exists for the given node, is not necessarily heavier

Active thread:

```
1:  loop
2:    if not proper(v, C_v) then
3:      C_v ← {}
4:      w_v ← −∞
5:    for all (u, C_u) ∈ received offers do
6:      if attr_v(C_u + {u} − {v}) > attr_v(C_v) then
7:        C_v ← C_u + {u} − {v}
8:        w_v ← w({v} + C_v)
9:    C ←HEURISTIC(k, v, G[{v} + N(v)], S)
10:   if attr_v(C) > attr_v(C_v) then
11:     C_v ← C
12:     w_v ← w({v} + C_v)
13:   send w_v to all u ∈ N(v)
14:   offer ← (v, C_v)
15:   send offer to all u ∈ C_v
```

Passive thread:

```
1:  loop
2:    receive w_u (or an offer) from any u ∈ N(v)
3:    store w_u (or an offer) locally
```

Figure 4.1: k-Clique matching protocol with a heuristic.

have never found this clique with the given heuristic. What happens here can be interpreted at large as a parallelization of the heuristic computation, which can also have a beneficial effect on the convergence speed.

Lastly, in line 9, the heuristic algorithm is executed. As a starting point for its execution, the value of C computed in lines 2–8 may be utilized if appropriate. The result of the heuristic is then assigned to variable $\mathbf{C}_v$ (see lines 10–12) only if it is more attractive than the clique found so far. Finally, updated values are sent to appropriate neighbors and a new execution of the loop begins.

The changes introduced in lines 2–8, 10–12, and 14–15 ensure that irrespective of the heuristic used in line 9, the network will converge eventually to a correct k-clique matching.

Self-Stabilization With Arbitrary Heuristic

So far the only thing we have assumed about the heuristic is that it executes in finite time and returns a subset of a node's neighbors of correct size, either $k-1$ or 0. Note that this means that we do not make any assumptions on the quality of

the heuristic's result, i.e., whether the returned clique is proper or more attractive than the one currently stored in variable $\mathbf{C}_v$. Even with such weak assumptions about the heuristic properties, we are able to show that the algorithm will converge to a correct k-clique matching in a finite number of rounds.

As previously, to simplify our discussion we will provide our proof for a shared-memory model and assume the composite atomicity with an atomic step consisting of a single execution of the loop. This single step requires reading by a node v the values of $\mathbf{w}_u$ and the relevant values of $\mathbf{C}_u$ (i.e., where $v \in \mathbf{C}_u$) of its neighbors, and writing (possibly multiple times) to the shared memory the new values of $\mathbf{w}_v$ and $\mathbf{C}_v$. Because each atomic step consists of an execution of the entire loop, only the last update of $\mathbf{w}_v$ and $\mathbf{C}_v$ will be visible to the neighbors. Such coarse granularity of the atomic step is therefore the closest to the message-passing model, where the new values of $\mathbf{w}_v$ and $\mathbf{C}_v$ are sent to the neighbors only at the end of the loop.

To be able to talk about the convergence of the algorithm, we will assume that we have access to an oracle that can tell us the heuristic's probability of finding a more attractive clique than the one indicated by variable $\mathbf{C}_v$ given that the local knowledge of the node remains unchanged and the heuristic is executed infinitely many times:
$\mathscr{P}_H(v,S) = \mathscr{P}\big(attr_v(\text{HEURISTIC}(k,v,G[\{v\}+N(v)],S)) > attr_v(\mathbf{C}_v)\big)$
The local knowledge of the node v in this case is simply the state of the system restricted to the closest neighborhood of v consisting of all values of $\mathbf{w}$ and $\mathbf{C}$ of v's neighbors.

We now proceed to proving that the algorithm defined in Figure 4.1 is self-stabilizing with regard to predicate P_H:

$$
\begin{aligned}
P_H: \quad \forall_v \Big(& |\mathbf{C}_v| + 1 = k \ (\vee\ \mathbf{C}_v = \emptyset) && (P_H 0)\\
& \wedge\ \mathbf{w}_v = w(\{v\} + \mathbf{C}_v) && (P_H 1)\\
& \wedge\ \forall_{u \in \mathbf{C}_v}\ w(\{v\} + \mathbf{C}_v) \geq \mathbf{w}_u && (P_H 2)\\
& \wedge\ \nexists_{u \in N(v)} \big(v \in \mathbf{C}_u \wedge attr_v(\mathbf{C}_u + \{u\} - \{v\}) > attr_v(\mathbf{C}_v) && (P_H 3a)\\
& \wedge\ \mathscr{P}_H(v,S) = 0 \Big). && (P_H 3b)
\end{aligned}
$$

Note that subpredicates $(P_H 0)$, $(P_H 1)$ and $(P_H 2)$ of predicate P_H are identical to the respective subpredicates of predicate P provided for the basic k-clique matching algorithm in Chapter 2, while the subpredicates $(P_H 3a)$ and $(P_H 3b)$ replace the last subpredicate of predicate P, which was tightly related to the impossibility of discovering a more attractive clique. Now, the impossibility of discovering a more attractive clique is divided among two possible sources for such a clique: from any of the neighbors in $(P_H 3a)$ and from the heuristic in $(P_H 3b)$. Moreover, observe that P_H contains the negations of the guards from the condi-

tional statements of the algorithm: $(P_H 2)$ for the guard in line 2, $(P_H 3a)$ for the guard in line 6, and $(P_H 3b)$ for line 10. As a result, if predicate P_H is true, no node v is able to modify their content of their variables $\mathbf{C}_v$ and $\mathbf{w}_v$, as all the statements modifying these variables are inside conditional statements guarded by expressions that are false if P_H is true. Thus, we can summarize it shortly with: no node is eligible for a move. From that immediately follows that the system is closed with regard to P_H; once P_H becomes true, no node can perform any action and change the state of the system, so P_H stays true:

Lemma 4.1. *(Closure) The k-clique matching algorithm as defined in Figure 4.1 is closed with regard to predicate P_H.*

Proof. This lemma follows directly from the definition of predicate P_H. If S is a safe state, then it can change only if at least one of the variables $\mathbf{C}_v$ or $\mathbf{w}_v$ changed for some node v. This is possible only if one of the guards of conditional statements in the algorithm from Figure 4.1 evaluated to true, yet because P_H is true in S, all these guards will evaluate to false for any node in the system. □

Without any knowledge about properties of the heuristic used, it is impossible to show, as we did for the basic algorithm, that the resulting k-clique matching is stable or unique, or to provide any guarantees with regard to the solution's quality. Yet what we can show is that once the system reaches a safe state, the sets of links stored by nodes in $\mathbf{C}_v$ form a correct k-clique matching.

Correctness

Lemma 4.2. *In a safe state, for each node v it holds that if $\mathbf{C}_v \neq \emptyset$, then $\forall_{u \in \mathbf{C}_v} \mathbf{C}_u + \{u\} = \mathbf{C}_v + \{v\}$.*

Proof. (By contradiction) Assume the contrary, that in a safe state there exists a node v such that $\mathbf{C}_v \neq \emptyset$ and $\exists_{u \in \mathbf{C}_v} \mathbf{C}_u + \{u\} \neq \mathbf{C}_v + \{v\}$. From the assumption on the uniqueness of clique weights, we have that because $\mathbf{C}_v + \{v\} \neq \mathbf{C}_u + \{u\}$, also $\mathbf{w}_v \neq \mathbf{w}_u$. As we did in Lemma 2.4, let's partition $\mathbf{C}_v$ into three sets:
$U^> = \{u \in \mathbf{C}_v : \mathbf{w}_u > \mathbf{w}_v\}$
$U^= = \{u \in \mathbf{C}_v : \mathbf{w}_u = \mathbf{w}_v\}$
$U^< = \{u \in \mathbf{C}_v : \mathbf{w}_u < \mathbf{w}_v\}$
Because there exists at least one node $u \in \mathbf{C}_v$ such that $\mathbf{w}_u \neq \mathbf{w}_v$, then at least one of the sets $U^>$ or $U^<$ is not empty:
a) if $U^> \neq \emptyset$ then $(P_H 1)$ is false for v. Thus, the guard in line 2 is true and v is eligible for a move, contradicting our assumption of the safe state.
b) if $U^> = \emptyset$ then $U^< \neq \emptyset$. In this situation, any node $u \in U^<$ is eligible to make a move because $\mathbf{C}_v - \{u\} + \{v\}$ is such that $(P_H 2)$ is false for u and guard in line 6 is true. This again is contradicting our assumption of the safe state. □

Lemma 4.3. *In a safe state, for any two nodes v and u either $(\mathbf{C}_v + \{v\}) \cap (\mathbf{C}_u + \{u\}) = \emptyset$ (are disjoint) or $\mathbf{C}_v + \{v\} = \mathbf{C}_u + \{u\}$.*

Proof. (By contradiction) Assume to the contrary that there exist two nodes v and u such that $(\mathbf{C}_v + \{v\}) \cap (\mathbf{C}_u + \{u\}) \neq \emptyset$ and $\mathbf{C}_v + \{v\} \neq \mathbf{C}_u + \{u\}$. Because the intersection of $(\mathbf{C}_v + \{v\})$ and $(\mathbf{C}_u + \{u\})$ is not empty, there must exists a node z belonging to this intersection. Then, by Lemma 4.2 for v holds that for each $t \in \mathbf{C}_v$ we have $\mathbf{C}_t + \{t\} = \mathbf{C}_v + \{v\}$, thus also for z this equality must hold, thus $\mathbf{C}_z + \{z\} = \mathbf{C}_v + \{v\}$. On the other hand, from the same lemma for node u follows that $\mathbf{C}_z + \{z\} = \mathbf{C}_u + \{u\}$. Therefore, $\mathbf{C}_v + \{v\} = \mathbf{C}_z + \{z\} = \mathbf{C}_u + \{u\}$ which contradicts our assumption that the two sets $\mathbf{C}_v + \{v\}$ and $\mathbf{C}_u + \{u\}$ are not equal. □

Corollary 4.4. *In a safe state, $M = \{Q_k : \exists_{v \in V} V(Q_k) = \{v\} + \mathbf{C}_v\}$ forms a correct k-clique matching.*

Proof. From Lemma 4.3 follows that $\{v\} + \mathbf{C}_v$ of different nodes are either equal or disjoint, thus all cliques that belong to M are pairwise disjoint. □

Convergence

Theorem 4.5. *Under a fair distributed daemon, any execution of the k-clique matching protocol with an arbitrary heuristic converges to a safe state in a finite number of time steps.*

Proof. First, let us explain the terms used in the formulation of the theorem. For reasoning about the behavior of a self-stabilizing system an abstraction of a *daemon* (or scheduler) is usually used. Such a daemon fully controls how the system progresses from one state to the next. The transition from one state to the other occurs in *time steps* and a daemon decides which nodes (or in general, processes) are eligible for a move will simultaneously execute their (atomic) steps during that time step. If the algorithm executed by a node consists of more than one step, the step to be executed is determined by the node's current state. The simultaneous execution of steps is carried out according to a fixed scheme: first, all chosen nodes read the states of their neighbors; second, they perform the computation; lastly, they update their own state variables, changing the state of the system. In particular, a *distributed daemon* can choose any arbitrary subset of nodes eligible for a move. Moreover, the fact that the daemon is *fair* ensures that each node that is eligible for a move infinitely often will execute its computation (atomic) step infinitely often. We can describe an execution under a fair distributed daemon as an alternating sequence $E = (S_1, A_1, S_2, A_2, \ldots)$, where S_i is a state of a system and A_i is a set of atomic steps executed simultaneously by a subset of chosen nodes

eligible for a move in state S_i, which leads to state S_{i+1}. Because we have assumed that the step consists of the entire loop, the nodes chosen for execution in the given time step unambiguously define which computations take place.

Proof by induction on the size of the network:

Base cases: For a network of 0 nodes, P_H is always true. For any network of at least 1 and at most $k-1$ nodes, irrespective of the initial content of variables $\mathbf{C}_v$ and $\mathbf{w}_v$, each node v after single execution of the entire loop, will have $\mathbf{C}_v$ set to $\emptyset$ and $\mathbf{w}_v$ to $-\infty$. Moreover, irrespective of the heuristic, $\mathscr{P}_H(v,S)$ is 0, because H can never return a $k-1$ neighbor subset, as no such subset exists (each node has at most $k-2$ neighbors).

Induction step: Assume that for any network of $N-1$ nodes, each execution converges to a safe state in a finite number of rounds. We will prove by contradiction that for a network of N nodes, there does not exist an infinite execution. Therefore, let us assume that for a given network of N nodes there exists an infinite execution $E = (S_1, A_1, S_2, A_2, \ldots)$ in which P_H is never true.

Now, let S' be the first state before which all nodes eligible for a move at the beginning of execution E did execute the entire loop at least once. In such a state, the contents of each $\mathbf{C}_v$ is then a correct subset of $k-1$ or 0 of v'c neighbors and $\mathbf{w}_v$ is a correct weight of clique $\mathbf{C}_v + \{v\}$. From that state onwards, $\mathbf{C}_v$ contains only correctly constructed subsets of v's neighbors and value $\mathbf{w}_v$ accurately reflects the weight of $(\mathbf{C}_v + \{v\})$.

Further, let S'' be the first state after S' that contains $\mathbf{w}_{v''}$ whose value is not surpassed by any other $\mathbf{w}_u$ in this and any subsequent state. Naturally, the value of $\mathbf{w}_{v''}$ is the correct weight of the clique $Q'' = (\mathbf{C}_{v''} + \{v''\})$. In other words, we claim that there must exist state S'' in the infinite execution E and a node v'' such that in any subsequent state S''' and for any node u, $\mathbf{w}_u$ from S''' is smaller than or equal to $\mathbf{w}_{v''}$ from S''. The existence of such $\mathbf{w}_{v''}$ and its corresponding clique Q'' is guaranteed by the fact that the number of all possible cliques in the network, thus also the set of all possible clique weights assigned to variables $\mathbf{w}$ after state S', is finite.

After S'', each node u from $Q'' - \{v''\}$ is eligible for a move and after its next execution of the loop (in particular, lines 5–8) will set its values of $\mathbf{C}_u$ and $\mathbf{w}_u$ to $Q'' - \{u\}$ and $\mathbf{w}_{v''}$. Denote by $\widehat{S}$ the first state in which all nodes from Q'' have their variables set to this clique and its weight. Due to our assumption about Q'', these k nodes will never again become eligible for a move, because none of the conditions that guard assignment statements to variables $\mathbf{C}$ and $\mathbf{w}$ will ever become true in any subsequent state. Therefore, also all predicates $(P_H 0)$, $(P_H 1)$, $(P_H 2)$, and $(P_H 3a)$ will remain true for these nodes for the rest of the execution E. Note that we cannot yet claim that $(P_H 3b)$ will remain true for these nodes for the rest of the execution E as $(P_H 3b)$ depends on the state of the system, and this

can still change in E.

As a consequence, this part of the system will remain unchanged for the rest of the execution E and none of the execution steps A_i after state $\widehat{S}$ will contain any of the nodes from clique Q''. We can consider the network that consists of the remaining $N-k$ nodes in isolation from the clique Q'', as it is also impossible for any of the remaining nodes to find during any of the execution steps subsequent to $\widehat{S}$ any proper clique containing any of Q'''s members. Therefore, the rest of the execution E after state $\widehat{S}$ can be treated as an execution, $\widehat{E}$, of the k-clique matching algorithm with the same heuristic but in a network of $N-k$ nodes. This execution remains infinite and for any of the states in $\widehat{E}$ the assumption that P_H never holds is inherited from execution E in the entire network. Yet, we have assumed that for any network of size smaller than N any execution convergence in a finite number of execution steps to a state for which P_H is true. Thus, there must exist a state S^P in $\widehat{E}$ for which P_H is true for the network of the remaining $N-k$ nodes.

Because neither any of the nodes from clique Q'' nor any of the remaining $N-k$ nodes can change its state anymore, the system remains in S^P. But this also implies that $(P_H 3b)$ must be true for nodes from Q''. Thus, S^P is a state reached in a finite number of steps in which P_H is true for the entire system of N nodes. Contradiction. □

So far, our assumptions about the properties of the heuristic were minimal. Because of that, we were unable to say anything about the quality of the matching found, i.e., its stability or its ratio to the optimal solution. Moreover, although we showed that the algorithm always converges in a finite number of steps to a safe state, we could not give any upper bounds. In the next two sections we describe two types of heuristics. The first type is a subset of deterministic heuristics, while the second is a subset of randomized heuristics. The differences between the properties of these two types of heuristics will result in differences in the properties of the k-clique matching algorithm. The advantage of the first type of heuristics is that it ensures a strict upper bound on convergence time, which is only one and a half of the upper bound of the k-clique matching algorithm that uses an exact function. Its downside is that we cannot provide any estimates with regard to the quality of the solution. On the other hand, the second type of heuristics gives only an expected upper bound of convergence, but once the safe state is reached the quality of the k-clique matching is exactly the same as for the solution produced by the exact k-clique matching algorithm.

4.1.2. Attractiveness-Maximizing Deterministic Heuristics

A deterministic heuristic is an algorithm that given the same input will always produce the same output. More formally, we can define a deterministic heuristic as an algorithm that computes the value of some mathematical function for a given input and returns this value as its output. Because for a mathematical function any input value can be associated with at most one output value, a given input uniquely identifies the output of a function. In our case such a mathematical function takes as an input a quadruple of a clique size, a node, the subgraph induced by the node and its neighbor set, and current state, $(k, v, G[\{v\} + N(v)], S)$, and returns a $k-1$ subset of node v's neighbors (or an empty set).

From this family of deterministic heuristics we choose those that have the following property: Assume that for a given clique size k, node v, subgraph $G[\{v\} + N(v)]$ and system state S, the heuristic H returns subset C. Then we require that the attractiveness of C for v in state S must be equal to or larger than the attractiveness of any other output C' of heuristic H given the same input values of k, v and $G[\{v\} + N(v)]$, but different states. Thus, if $\mathscr{C}$ is the set of all H's outputs for fixed k, v and $G[\{v\} + N(v)]$, then H's output value, C, for those fixed input values and given state S should be the most attractive among the subsets in $\mathscr{C}$ for v in the given state S, i.e., $attr_v(H((k, v, G[\{v\} + N(v)], S))) = attr_v(C) \geq max\{attr_v(C') : C' \in \mathscr{C}\}$. We will refer to such heuristics as *attractiveness-maximizing deterministic heuristics*.

A simple example is a heuristic that first chooses a small subset of a node's neighbors according to some deterministic rule. Then only among the neighbors from this subset, evaluates all possible combinations to find the most attractive clique. If the size of the selected neighbor subset is at most equal to the logarithm of the network size, then each execution of the heuristic will take at most $O(|V|)$ time, as $\binom{\log_2 |V|}{k-1} < (1+1)^{\log_2 |V|} = |V|$ for any $k <= \log_2 |V| + 1$.[2] Moreover, if the network membership or edge weights do not change, then this subset also does not change for any node through the execution of the algorithm. Because in each execution of the heuristic for a given node, the same set of $k-1$ neighbor combinations is examined and the most attractive combination is returned, the property of attractiveness-maximizing is satisfied. An example of such defined attractiveness-maximizing deterministic heuristic is HEAVIESTEDGESUBSET heuristic which selects from the neighbor set of node v a fixed the number of neighbors whose edge weight connecting them to v is the highest.

[2] Actually a larger limit of subset size could be used, e.g., $|V|^{1/(k-1)}$, but choosing the logarithm of network size has a nice property of guaranteeing $O(|V|)$ execution time even when instead of one possible value of k we have a whole set K of allowed clique sizes (compare with Chapter 3.1).

Convergence with Attractiveness-Maximizing Deterministic Heuristic

Denote by $\mathcal{Q}_H(v)$ the set of all cliques that can be outputs of an attractiveness-maximizing deterministic heuristic H for clique size k, node v, $G[\{v\} + N(v)]$ and any possible system state. $\mathcal{Q}_H$ is then the union of all sets $\mathcal{Q}_H(v)$ for $v \in V$. Moreover, denote by $\mathcal{Q}_S$ the set of all cliques that come from the content of variables $\mathbf{C}_v$ in the state S of the system that at the same time do not belong to $\mathcal{Q}_H$, i.e. $\mathcal{Q}_S = \{\mathbf{C}_v + \{v\} : \mathbf{C}_v \in S \wedge \mathbf{C}_v + \{v\} \notin \mathcal{Q}_H\}$. Especially $\mathcal{Q}_{S_I}$, is such a set in the initial state. Observe that the set $\mathcal{Q}_H$ remains unchanged through the entire execution of the algorithm, while set $\mathcal{Q}_S$ may become smaller because some cliques from $\mathcal{Q}_S$ can be discarded if they become improper in some state; therefore, if state S' has been reached from state S, then $\mathcal{Q}_{S'} \subseteq \mathcal{Q}_S$. Moreover, the set of all cliques pursued by nodes in the given state S, $\{\mathbf{C}_v + \{v\} : \mathbf{C}_v \in S\}$ must be a subset of $\mathcal{Q}_H + \mathcal{Q}_S$. Therefore, the final k-clique matching M from the safe state can contain only the cliques that come from $\mathcal{Q}_H + \mathcal{Q}_{S_{safe}} \subseteq \mathcal{Q}_H + \mathcal{Q}_{S_I}$.

Theorem 4.6. *Under a fair distributed daemon, a k-clique matching protocol that uses an attractiveness-maximizing deterministic heuristic H converges in at most $3|M| + 1$ rounds.*

Proof. (By Induction) First, observe that due to our assumption that each k-clique has a unique weight, we can order all k-cliques in the graph by their weight. We can similarly order all k-cliques in the resulting matching M, $w(Q_k^1) > w(Q_k^2) > \ldots > w(Q_k^{|M|})$.
Base step: After the first three rounds at least the heaviest k-clique Q_k^1 from the final matching M is correctly matched.
At the beginning of the execution, the nodes in the system may have incorrect values of variables w and C. During the first round, each node v assigns to the variable $\mathbf{C}_v$ a correct $k-1$ neighbor combination and to variable $\mathbf{w}_v$ the corresponding weight $w(\mathbf{C}_v + \{v\})$. After the first round, when the system is in the state S, each of the cliques from the set $\mathcal{Q}_S$ is going to be considered by at least one of the nodes, just as each of the cliques from $\mathcal{Q}_H$. Moreover, the value of $\mathbf{w}_v$ of each node v is smaller than or equal to the weight of the heaviest clique Q_k from $\mathcal{Q}_S + \mathcal{Q}_H$. Thus, in the second round, at least one node, v, belonging to Q_k assigns its values of $\mathbf{C}_v$ to $Q_k - \{v\}$ and $\mathbf{w}_v$ to $w(Q_k)$. In the third round, all other nodes from Q_k read v's variables and also update their variables to point to Q_k. Because in every subsequent state S' the set $\mathcal{Q}_H + \mathcal{Q}_{S'}$ must be a subset of $\mathcal{Q}_H + \mathcal{Q}_S$ and clique Q_k is the heaviest clique that nodes can create, it is guaranteed to be the heaviest clique in the final matching M denoted by Q_k^1.
Induction step: Given that nodes from the i heaviest k-cliques from the final matching have already stabilized, only 3 rounds are needed for the nodes from the $(i+1)$-th k-clique to stabilize.

First, observe that once the nodes from the i heaviest k-cliques from the final matching create their cliques, they will never become enabled again. For every node in the j-th clique, any clique that is better that Q_k^j, has at least one node that already belongs to one of the cliques from the final matching with an index smaller than j. And with nodes that do not belong to the first $j-1$ cliques, nodes from the j-th clique can create a clique whose weight will never be higher than the weight of Q_k^j.
Thus, when the i heaviest k-cliques from the final matching have already stabilized, the nodes from the next heaviest k-clique have the first round to correct their values of w which can be higher than Q_k^{i+1}. In the next round at least one of them sets its value of $\mathbf{w}$ to $w(Q_k^{i+1})$, and all remaining nodes from Q_k^{i+1} set their values of $\mathbf{w}$ to $w(Q_k^{i+1})$ in the subsequent round.
Thus, $3 \cdot M$ rounds are needed by the nodes from cliques that belong to the final matching to stabilize. The last round is for all the remaining nodes to set their values of $\mathbf{C}$ to an empty set and the value of $\mathbf{w}$ to $-\infty$. □

4.1.3. Randomized Heuristics

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 (possibly 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.

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 [BMUN09] that VNS consistently achieved the best results in solving the heaviest k-subgraph problem. The heuristics against which Brimberg et al. tested VNS include [GP10a]: 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 the scope of this thesis.

VNS is a meta-heuristic that found its application in a vast range of combinatorial and global optimization problems [HMMP10] 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 best improvement mode 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 [CvS10].

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.) might be unrealistic. In Section 4.4 we present how this assumption can be dropped.

RANDOMSUBSET Heuristic

Another heuristic is based on a simple trick that we have also used in the HEAVIESTEDGESUBSET heuristic: 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 in a single round all $k-1$ combinations of neighbors from the subset. To construct a subset, node v can start by including into the subset the $k-1$ neighbors that are in its current clique $\mathbf{C}_v$ provided this clique is *proper*. Then, it can fill the subset to the desired size by simply picking randomly the necessary number of neighbors. Once the subset is created, the node executes the for-loop from the basic protocol (see Figure 2.4 lines 3–5) 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 4.4.

Convergence with Randomized Heuristics

Because randomized heuristic works in principle just as an exact algorithm minus the number of executions necessary to find the most attractive clique, the k-clique matching M_{RH} found by the algorithm using randomized heuristic will be exactly the same as the k-clique matching M_B returned by the basic k-clique matching algorithm. Therefore, M_{RH} will also have the same properties as M_B, that is it will be unique, stable and its total weight will be at least $1/k$ of the optimal matching's weight. The only difference is then the time necessary for the system to converge, which is much longer for the algorithm with randomized heuristic.

Theorem 4.7. *Under fair distributed daemon, k-clique matching protocol that uses a randomized heuristic RH converges in expected* $\left(2+\frac{1}{1-(1-p)^k}\right)|M|+1$ *rounds, where p is the minimum probability of examining any given clique in a single execution of RH.*[3]

Proof. (By Induction) *Base step:* The nodes from the heaviest k-clique Q_k^1 stabilize in expected $2+\frac{1}{1-(1-p)^k}$ rounds.
At the beginning of the execution, the nodes may have incorrect values of variables w and C. In such a situation nodes that belong to the heaviest clique in the graph in the first round will set values of w to less than $w(Q_k^1)$. Then, because in

[3]For example, for the RANDOMSUBSET heuristic, $p >= \frac{\binom{s-(k-1)}{k-1}}{\binom{|V|}{k-1}}$, where s is the size of the subset.

each of the following rounds the probability of discovering Q_k^1 by one of the nodes belonging to this clique is p, the probability that any of these nodes will discover Q_k^1 is $1-(1-p)^k$. Thus, the expected time for these event to occur is $\frac{1}{1-(1-p)^k}$. Once this event occurs, the node that has found Q_k^1 will inform all other nodes from this clique about it, and in the following round also these nodes will point to Q_k^1, which gives in total the expected $2+\frac{1}{1-(1-p)^k}$ rounds for nodes from Q_k^1 to assign their values of $\mathbf{w}$ to $w(Q_k^1)$.

Induction step: Given that nodes from the i heaviest k-cliques from the final matching have already stabilized, an expected $2+\frac{1}{1-(1-p)^k}$ rounds are needed for the nodes from the $(i+1)$-th k-clique to stabilize.

First, observe that once the nodes from the i heaviest k-cliques from the final matching create their cliques, they will never become enabled again. For every node in the j-th clique, any clique that is better that Q_k^j, has at least one node that already belongs to one of the cliques from the final matching with an index smaller than j. With nodes that do not belong to the first $j-1$ cliques, nodes from the j-th clique can create a clique whose weight will never be higher than the weight of Q_k^j.

Thus, when the i heaviest k-cliques from the final matching have already stabilized, the nodes from the next heaviest k-clique have the first round to correct their values of w which can be higher than Q_k^{i+1}. In the next $\frac{1}{1-(1-p)^k}$ rounds on average, at least one of the nodes from Q_k^{i+1} sets its value of $\mathbf{w}$ to $w(Q_k^{i+1})$, and in the subsequent round all other nodes will follow suit.

Thus, $(2+\frac{1}{1-(1-p)^k})\cdot M$ rounds are needed by the nodes from cliques that belong to the final matching to stabilize. The last round is for all the remaining nodes to set their values of C to an empty set and value of $\mathbf{w}$ to $-\infty$. □

If p is really small, then the bound of $(2+\frac{1}{1-(1-p)^k})\cdot M+1$ can become really large. Therefore, to improve the probabilities of finding the best clique with a randomized heuristic, we can try to limit the solution space, by discarding from consideration (pruning) neighbors which, given the current state, for sure are not part of the proper clique. We describe how it can be done in Section 4.3.

On the other hand, instead of waiting for the entire system to converge, nodes can leave the system sooner, provided that they have found the clique that is good enough in terms of weight, or that the time since the last improvement have reached some specified limit. We explore this variation of the algorithm in Section 4.2.

4.1.4. Experimental Results

General Experimental Settings

Graph topology: As in previous chapters, we run our simulations in the networks where the connections between the nodes create a complete graph. This guarantees that each maximal (in unweighted terms) k-clique matching contains exactly $\lfloor |V|/k \rfloor$ cliques. The networks used for simulations have 300, 600, 1200, and 2400 nodes.

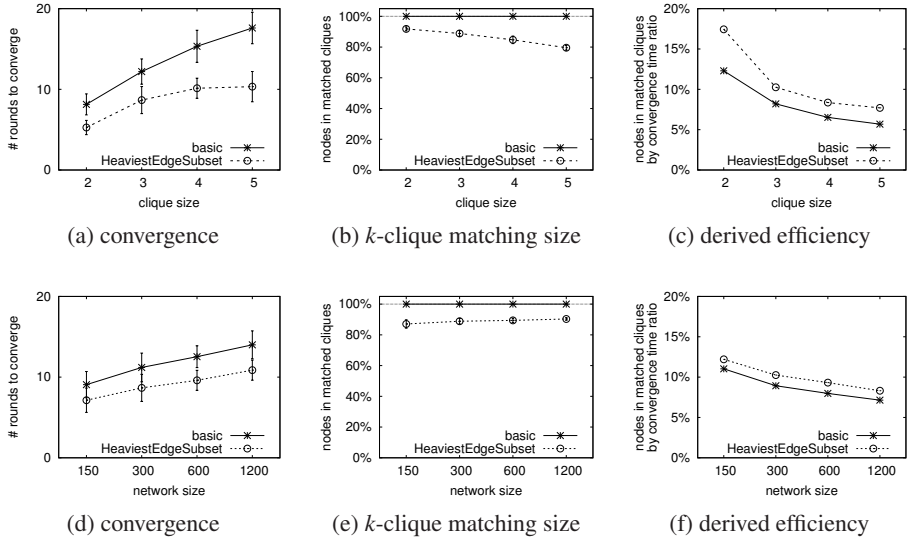
Edge Weights: To thoroughly evaluate the performance of the k-clique matching protocol employing various heuristics, we use two different distributions of the edge weights. For the first distribution, *uniform*, the weights of the edges are assigned uniformly at random from the $(0, 1)$ interval, thus, there is no correlation between the edges that share a common node. In the other distribution, *euclid-4* the edge weights are closely correlated with each other. To compute the edge weights for this distribution, we start by assigning to each node random coordinates from a 4-dimensional space[4]. These coordinates can be interpreted as measures of four different properties of each node. The weight between any two nodes is then computed as the Euclidean distance between their coordinates, therefore reflecting the similarity between the two nodes.

Clique Sizes and Weights: The performance of the protocols is examined for clique sizes ranging from 2 to 5 nodes. For both distributions, *uniform* and *euclid-4*, the weight of each clique is computed as the arithmetic average of the weights of all edges belonging to this clique. When *uniform* distribution of edge weights is used, the nodes aim at maximizing the weights of their cliques [5]. When *euclid-4* distribution is used, nodes can either try to maximize or minimize the weight of their cliques, which can be interpreted as seeking cliques in which all nodes are pairwise dissimilar or pairwise similar, respectively. We investigate both of these cases, referring to them as *euclid-4 max* and *euclid-4 min*.

Simulations execution: The simulations were performed using the PeerSim simulator. The execution of the protocols is organized in the round-based fashion. In each round, one node after another executes a single loop (a single atomic step). The node ordering is random in each round. The updated values of a node's variables are immediately available to the node's neighbors.

[4] See Appendix A for an analysis of the basic k-clique matching protocol in the networks with such distributions.

[5] Minimizing the clique weights in a graph with *uniform* edge-weight distributions would be equivalent to maximizing the clique weights in the graph in which each edge weight is equal to 1 minus the edge weight from the original graph, therefore, also having a *uniform* distribution of edge weights.



(a) convergence (b) k-clique matching size (c) derived efficiency

(d) convergence (e) k-clique matching size (f) derived efficiency

Figure 4.2: Comparison of performance between original k-clique matching algorithm (doing full search over all neighbor combinations) and k-clique matching algorithm using HEAVIESTEDGESUBSET heuristic (a)–(c) over various sizes of cliques in 300-node networks and (d)-(f) for 3-cliques over various sizes of network. The derived efficiency is computed as a ratio between the average size of the final matching and the average number of rounds necessary for convergence.

Experimental Results for Attractiveness-Maximizing Deterministic Heuristics

We evaluate the effectiveness of our k-clique matching algorithm that employs attractiveness-maximizing deterministic heuristics by observing (i) the average number of rounds necessary for the system to stabilize and (ii) the average size of the final k-clique matching, expressed as the percentage of nodes in the network that belong to one of the cliques from that matching. As an attractiveness-maximizing deterministic heuristic we use the HEAVIESTEDGESUBSET heuristic introduced in Section 4.1.2. This heuristic limits the number of neighbor combinations examined in each round by selecting s neighbors connected by heaviest edges to the node as the only potential clique partners. In our simulations, the size of s was determined by the size of the network and computed as $\lfloor log_2(|V|) \rfloor$, which amounts to 7 neighbors in an 150-node network, 8 neighbors in an 300-node network and, finally, 11 neighbors in an 2400-node network.

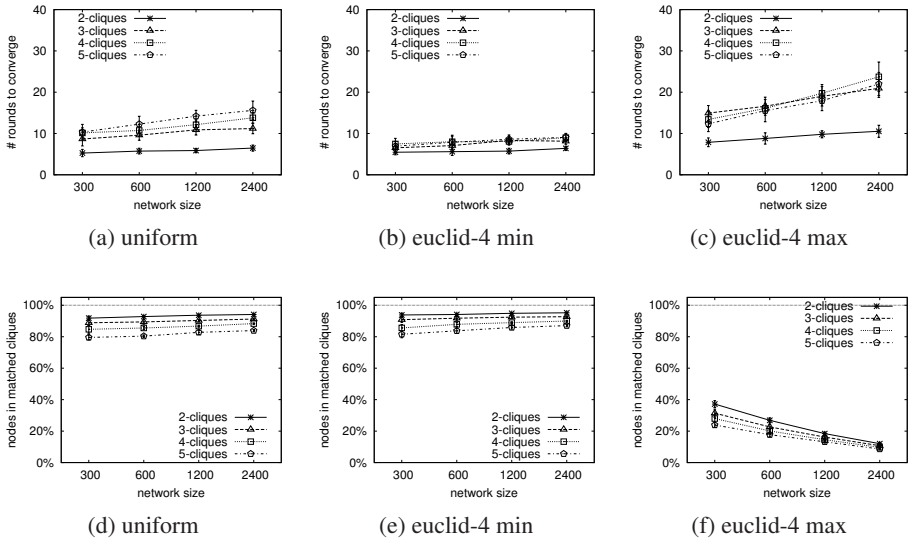
First, let us compare the performance of our k-clique matching algorithm that uses HEAVIESTEDGESUBSET to the basic k-clique matching algorithm from

Chapter 2. Figure 4.2 shows the differences in average convergence times and the average percentages of nodes forming the final k-clique matchings in graphs with uniform edge-weight distributions. The first row focuses on the results across various values of clique size, k, in a small network of 300 nodes, the second row targets the performance across various network sizes while creating 3-cliques. The first thing that we observe is that the algorithm that uses the heuristic converges faster and that this trend is consistent across various clique and network sizes (see Figures 4.2a and 4.2d). Although the worst case convergence bound is half larger for the heuristic $(3 * |M| + 1)$ than for the basic algorithm $(2 * |M| + 1)$, this does not necessarily translate onto the convergence times. In fact, in our simulations, the algorithm that used a heuristic was visibly faster. And this is without taking into account the reduced computational load of each round, as only $\binom{\log_2 |V|}{k-1}$, instead of $\binom{|V|}{k-1}$, neighbor combinations are examined by node per round.

Naturally, the tradeoff of lower computational cost of a round is in the quality of the produced matching. Not all nodes find suitable cliques, as Figures 4.2b and 4.2e show, when HEAVIESTEDGESUBSET is used, the size of the final k-clique matching never reaches 100% of nodes, although even in the worst tested case, for $k = 5$ in 300-node network, it reached almost 80% (79.5%, $s = 1.6\%$). To look at our results from a slightly different perspective, we consolidate them into one measure to express the overall efficiency of the two algorithms. To compute this new measure, we have divided the number of nodes forming the final k-clique matching by the number of rounds till full convergence, ultimately obtaining an average number of stable cliques created by the algorithm per single round. From Figures 4.2c and 4.2f presenting this new measure, the algorithm that uses HEAVIESTEDGESUBSET visibly outperforms the basic k-clique matching algorithm for each setting of parameters.

In Figure 4.3 we zoom into more details of HEAVIESTEDGESUBSET performance, examining the convergence times and the final k-clique matching sizes across various clique sizes and in various sizes of the networks with different edge-weight distributions. Our first observation is that the number of rounds necessary for the convergence of the algorithm is significantly lower than the upper bound of $3|M| + 1$ provided in the Theorem 4.6. Moreover, the convergence times show only slight increase related to the growth of the network size, which suggests good scalability properties of the algorithm combined with this heuristic. Finally, note that the differences in the convergence times for different edge-weight distributions seem to be consistent with the results obtained in analogous simulations on the basic k-clique matching algorithm in Appendix A.

The second row of graphs (Figures 4.3d, 4.3e, and 4.3f) depicting the percentage of nodes forming the final k-clique matching creates a less optimistic portrait of HEAVIESTEDGESUBSET performance as a part of the k-clique matching algo-



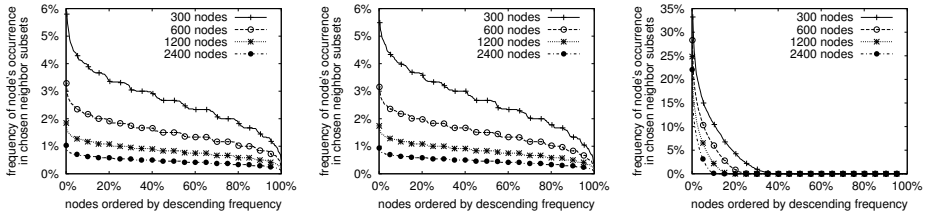
(a) uniform (b) euclid-4 min (c) euclid-4 max

(d) uniform (e) euclid-4 min (f) euclid-4 max

Figure 4.3: Performance of k-clique matching algorithm using HEAVIESTEDGE-SUBSET in three selected weight distributions: (a)–(c) number of rounds needed to converge, (d)–(f) percentage of nodes in the network forming the final k-clique matching.

rithm. On a positive note, in graphs with uniform or euclid-4 min edge-weight distributions, the final k-clique matchings consist of a majority of nodes: in a 300-node network at least 79.5% of nodes for $k = 5$ up to 91.8% of nodes for $k = 2$ with uniform distribution of edge weights (81.5% and 93.8% with euclid-4 min distribution). For larger networks these percentage increase even further. Yet, when the edge weights in the network follow euclid-4 max distribution, the average number of nodes in the final matching amounts only to 23.9% for $k = 5$ up to 37.2% for $k = 2$ in 300-node network and plunges to just 8.6% and 12.0% respectively in 2400-node network.

The explanation for such large discrepancies in final k-clique matching sizes lies in the distribution of nodes in the subsets computed with HEAVIESTEDGE-SUBSET, which we present in Figure 4.4. In each of the graphs, we depict the frequency with which any given node occurs in subsets of other nodes. In case of uniform and euclid-4 min edge-weight distributions, each of the nodes is present in roughly the same number of neighbor subsets. In case of euclid-4 max the situation looks diametrically different; the neighbor subsets are filled entirely by only 39% of all nodes in the 300-node network decreasing further to only 11% in a 2400-node network. As a result, a node that fails to create a stable clique with

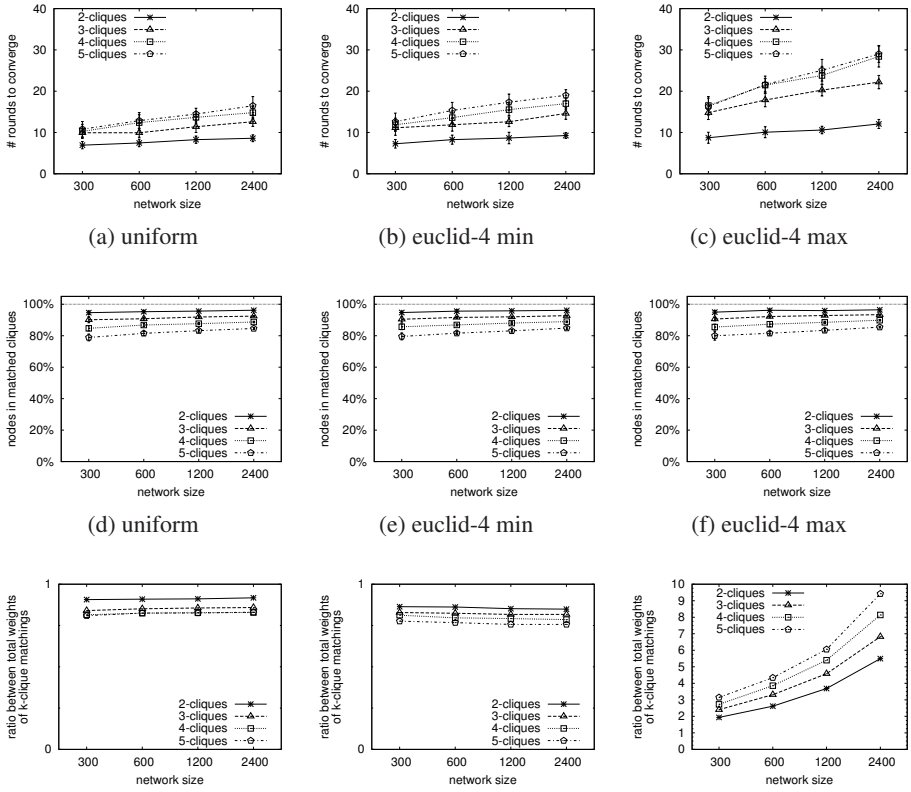


(a) uniform (b) euclid-4 min (c) euclid-4 max

Figure 4.4: Frequency of node's occurrence in the neighbor subsets used by HEAVIESTEDGESUBSET; nodes on the X-axis are ordered by their decreasing frequency.

nodes from its neighbor subsets computed by HEAVIESTEDGESUBSET, remains clique-less although there are many other nodes in a similar situation with which it could potentially form a clique.

In this light, the poor results of the k-clique matching algorithm using HEAVIESTEDGESUBSET in networks with euclid-4 max edge-weight distribution do not undermine the usefulness of employing attractiveness-maximizing deterministic heuristics in our algorithm. Instead, the results suggest that the heuristic should be tailored to the edge-weight distribution present in the system in order to provide enough variability in the examined neighbor combinations. As an example, let us see how the k-clique matching performs if the heuristic, which we will refer to as RANDOMEDGESUBSET, selects RANDOMSUBSET of neighbors which is then used by the node in each round. The simulation results are presented in Figure 4.5. First and foremost, we notice that the percentage of nodes forming a final matching shows great improvement for the euclid-4 max distribution (Figure 4.5f), in comparison with the percentage achieved by the algorithm that used HEAVIESTEDGESUBSET (Figure 4.3f). In fact, the sizes of final k-clique matchings produced by the algorithm using RANDOMEDGESUBSET are now almost identical for all tested edge-weight distributions for the respective values of k, which would suggest that this heuristic could be universally applicable to all networks irrespective of their distributions of edge weights. Moreover, for the uniform and euclid-4 min the final k-clique matching sizes are also very close to the sizes obtained by the algorithm using HEAVIESTEDGESUBSET. Therefore, while for euclid-4 max the advantage of RANDOMEDGESUBSET over HEAVIESTEDGESUBSET is undisputed, there is no clear cut between the results of these two heuristics for uniform and euclid-4 min edge-weight distributions, and we can even question whether there exist any benefits of using HEAVIESTEDGESUBSET also for these two distributions.



(a) uniform (b) euclid-4 min (c) euclid-4 max

(d) uniform (e) euclid-4 min (f) euclid-4 max

(g) uniform (h) euclid-4 min (i) euclid-4 max

Figure 4.5: Performance of k-clique matching algorithm using RANDOMEDGESUBSET in three selected weight distributions: (a)–(c) number of rounds needed to converge, (d)–(f) percentage of nodes in the network forming the final k-clique matching, (g)–(i) ratio between the weight of the total k-clique matchings produced by the algorithm using RANDOMEDGESUBSET and HEAVIESTEDGESUBSET.

To complete the discussion on the performance of attractiveness-maximizing deterministic heuristics, let us compare not only the sizes of the final k-clique matchings but their total weights. Figures 4.5g, 4.5h, and 4.5i depict the ratio between the average total weight of the final k-clique matching produced by our algorithm that uses HEAVIESTEDGESUBSET to the average total weight of the final k-clique matching produced by our algorithm that uses RANDOMEDGESUBSET. These new graphs clearly show that the HEAVIESTEDGESUBSET performs much better in uniform and euclid-4 min cases. This implies that HEAVIESTEDGESUB-

SET's rule of selecting neighbors for the subset works much better in such edge-weight distribution. This only strengthens our argument that the heuristic should be tailored to the edge-weight distribution present in the system to optimize the algorithm's performance.

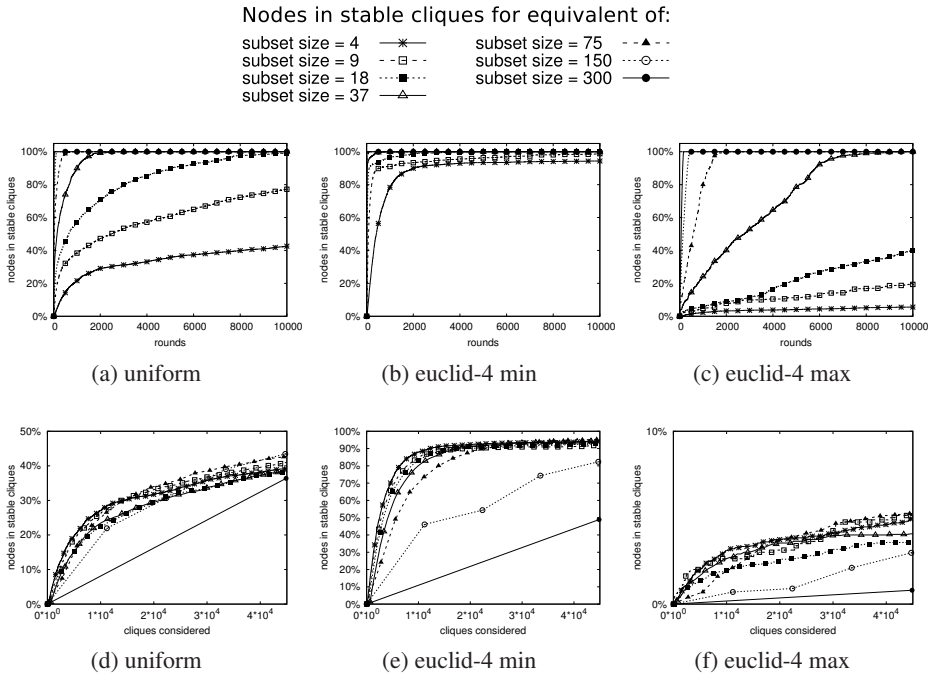
Experimental Results for Randomized Heuristics

In this section we discuss the simulation results for the k-clique matching algorithm that uses randomized heuristics. The first challenge that we faced when trying to evaluate the performance of this version of the algorithm, was tackling the fact that by observing the changes in the states of the nodes we were no longer able to determine whether the network had already stabilized. This has never been the case for any of our previous versions of the algorithm. In all of the algorithm's versions so far, we could tell with full certainty that the system had converged by comparing the state of the system over the span of one round; if there was no change in the values of variables $\mathbf{C}_V$ and $\mathbf{w}_v$ for any node v in the system since some fixed point in time (e.g., the end of last round) and if since then each node v had a chance to move, then the system must have converged. Yet, when the k-clique matching algorithm that uses randomized heuristics is considered, we can no longer make such an inference.

Another way to check if the k-clique matching algorithm that uses randomized heuristics has stabilized, is to compute the final k-clique matching before the simulation starts and then after each round compare the local state of every node to the state it should have when the final k-clique matching is formed. The disadvantage of this approach is that for even fairly small values of clique size k, such as 4 or 5, and larger sizes of network, the computation of the final k-clique matching, which takes $O(|V|^k)$, is time consuming and, thus, impractical if we want to investigate the algorithms across various sizes of networks and cliques. Therefore, we use this approach only for a single parameter setting of a 600-node network and clique size $k = 3$. In this setting we thoroughly evaluate the algorithm's performance in the three proposed edge-weight distributions for various computational loads granted to each of the two randomized heuristics: VNS and RANDOMSUBSET.

In the case of RANDOMSUBSET, the computational power granted to the heuristic can be easily controlled by adjusting the size of the neighbor subset that is searched in a single round; when the neighbor subset is of size s, we know that the heuristic will examine $\binom{s}{2}$ neighbor combinations in a single execution. For our simulations we chose values of s ranging from only 4 up to half of the network size. We will refer to the heuristics with small computational power as the thin-round heuristics, in contrast to the heuristics with large computational power, to which we refer as the fat-round heuristics.

To have a fair comparison between RANDOMSUBSET and VNS, we impose



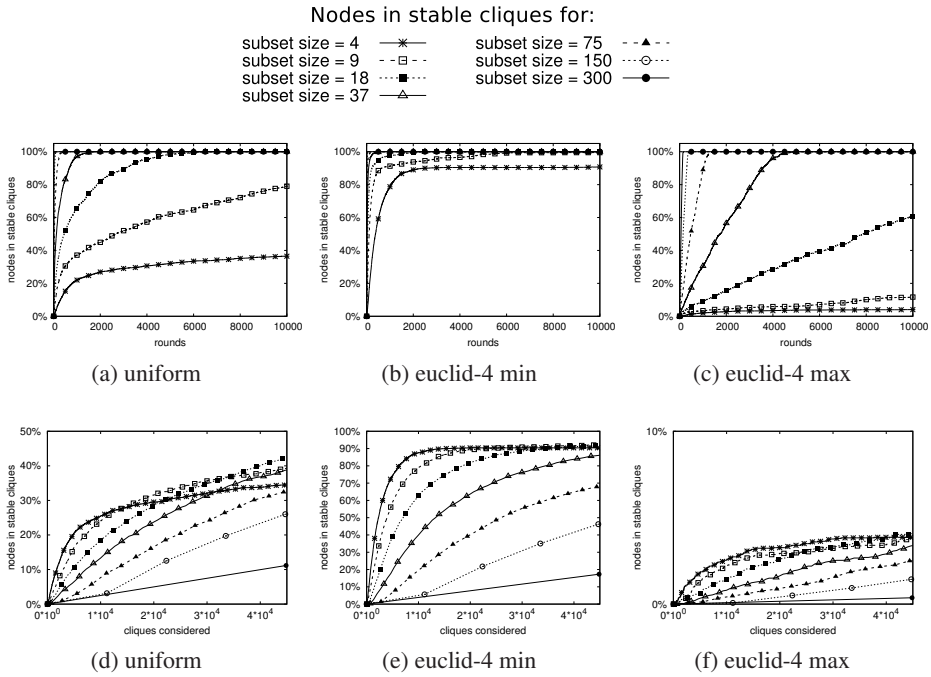
(a) uniform (b) euclid-4 min (c) euclid-4 max

(d) uniform (e) euclid-4 min (f) euclid-4 max

Figure 4.6: Performance of k-clique matching algorithm using VNS in three selected weight distributions for clique size $k = 3$ in 600-node networks: (a)–(c) stable cliques per round, (d)–(f) stable cliques per total computational load (total number of considered cliques) by a node since the beginning of a simulation.

identical constraints on the number of neighbor combinations examined during a single execution of the two heuristics. Thus, we set as a stopping condition of VNS $\binom{s}{2}$ of considered neighbor combinations for respective values of subset sizes, s, granted to RANDOMSUBSET.

Figures 4.6 and 4.7 depict the performance of our k-clique matching algorithm using VNS or RANDOMSUBSET respectively. For each simulation we have computed beforehand the final k-clique matching and we let a simulation run for 100,000 rounds. After the simulation finished, we counted how many cliques from the final k-clique matching were already formed, and maintained formed, since any given round. Each curve traces the average percentage of nodes in these cliques. To be precise, if the final k-clique matching was reached before the end of the simulation, these cliques were the stable cliques. Yet, if the simulation finished before the final k-clique matching was formed, we have no guarantee that the nodes from every of these cliques would have remained matched if the simulation had run further. Despite the uncertainty of clique stability in non-stabilized simulations, we can clearly see certain regularities in the performance of the algo-



(a) uniform (b) euclid-4 min (c) euclid-4 max

(d) uniform (e) euclid-4 min (f) euclid-4 max

Figure 4.7: Performance of k-clique matching algorithm using RANDOMSUBSET in three selected weight distributions for clique size $k = 3$ in 600-node networks: (a)–(c) stable cliques per round, (d)–(f) stable cliques per total computational workload (total number of considered cliques) by a node since the beginning of a simulation.

rithm.

A first thing that we observe consistently for both heuristics over three different edge-weight distributions is that in terms of the total number of rounds (see Figures 4.6a–4.6c and 4.7a–4.7c), the more computational power the heuristic has, the steeper the curve is, meaning, the faster the network converges. This is not surprising, because the more neighbor combinations a node can verify in a single round the larger the probability that it will find the one that is most attractive for it. Nonetheless, if we zoom into the initial stages of the simulations and compare the percentages of stable cliques formed not in terms of rounds but of the computational workload performed by nodes (see Figures 4.6d–4.6f and 4.7d–4.7f), we will observe that with the thin-round heuristics, nodes find stable cliques much faster. During just a small fraction of the time necessary for the fattest-round heuristic to execute a single round, the thinner-round heuristics find just as many or even multiple times more stable cliques. Unfortunately, the speed of forming new cliques by the thin-round heuristics decreases fast, eliminating the

discrepancies.

Secondly, we can see that out of the three tested edge weight distributions, the algorithms converge fastest in euclid-4 min and slowest in the euclid-4 max, with the results for uniform distribution falling somewhere in the middle. This is consistent with the results of attractiveness-maximizing deterministic heuristics in the previous section and also of the results of the basic k-clique matching for various edge-weight distributions presented in Appendix A, where we also explain the discrepancies in the convergence times caused by different edge-weight distributions.

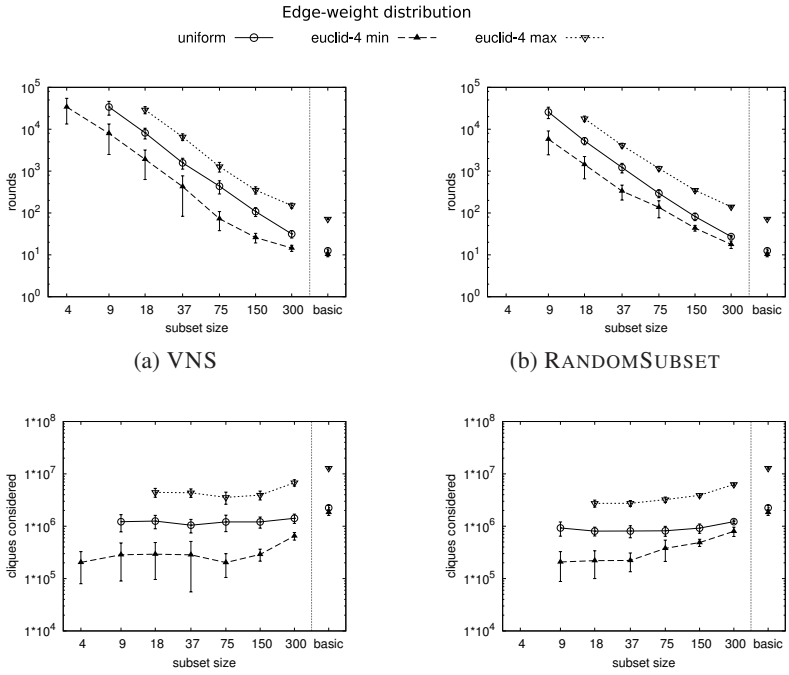
Our final observation about the results presented in Figures 4.6 and 4.7 is related to the lack of bigger differences in the performance between the two heuristics. Although as reported in [BMUN09], VNS consistently achieved the best results in solving the heaviest k-subgraph problem, beating among others also heuristics similar to RANDOMSUBSET, when used in the k-clique matching algorithm, the advantage achieved by VNS is small or virtually none. This suggests that we can successfully use the simpler one to implement randomized heuristic, and achieve equally good performance. This is especially encouraging if we consider situations in which the nodes in the system store profiles of only a small number of their neighbors at a time, because only a small subset of neighbors is required to execute RANDOMSUBSET in each round. We discuss the issue of nodes restricted views in more detail in Section 4.4.

In Figures 4.8a and 4.8b we can see more clearly the dependencies between computational powers granted to the heuristics and the average number of rounds necessary for the network to stabilize. We have selected computational powers in our simulations such that each consecutive is roughly equal to the quadruple of its predecessor, as $\binom{2s}{k-1}/\binom{s}{k-1} = \frac{2(2s-1)}{s-1}$ for $k = 3$, and we can observe that the difference between the convergence times of two consecutive computational powers is also roughly fourfold. However, if we compare the convergence times measured as the total number of cliques considered by the heuristic since the beginning of simulation for various computational powers, they turn out to be roughly the same (see Figures 4.8c and 4.8d). As a result, we can see that using a randomized heuristic does not visibly decrease the time necessary for stabilization. The benefit of using randomized heuristics lies not in speeding up the convergence time but rather in allowing nodes to find reasonably good cliques fast.

Discussion: What do these graphs teach us?

Here we summarize the most important observations on the thin-round heuristics versus fat-round heuristics:

Similar computational costs: If we consider solely the number of rounds necessary for the convergence, then the fat-round heuristics perform undoubtedly bet-



(a) VNS (b) RANDOMSUBSET

(c) VNS (d) RANDOMSUBSET

Figure 4.8: Comparison between average convergence times of k-clique matching algorithm using VNS or RANDOMSUBSET in three selected weight distributions for clique size $k = 3$ in 600-node networks. In the first row time is measured in rounds, in the second row time is measured in total number of considered cliques from the beginning of simulation.

ter than the thin-round heuristics. Yet, in the terms of the total computational costs, the differences between fat- and thin-round heuristics become much smaller, even with a slight advantage of the thinner-round heuristics. This small advantage can be attributed to the fact that the fat-round heuristics perform work in larger chunks.

Different communication costs: Although the computational costs of convergence are similar for both thin-round and fat-round heuristics, the same cannot be said about the communication costs. In fact, the computational power of a heuristic is inversely proportional to the frequency with which nodes broadcast their new clique weights, which results in thin-round heuristics requiring much more bandwidth. These costs cannot be easily neglected, and thus, the nodes should try to maximize the computational workload they perform in a single round to save on communication. Therefore, the fatter-round heuristics should clearly be preferred.

Yet, more *frequent communication has its benefits*. This is especially visible in the initial stages of our simulations. The fast progress in stable clique formation

at the very beginning of thin-round heuristic simulations can be directly attributed to the more frequent exchange of the changes in nodes' states. What happens here is that the nodes, instead of relying almost solely on their own computations to find the most attractive clique, utilize to this end in a much greater extent the information received from their fellow nodes.

Hidden waste of resources: Once the nodes find their most attractive clique and stabilize, the subsequent rounds of a simulation do not bring any change to their choice. As a result, we can look at their continued execution of the protocol as a waste of their resources. This is naturally true provided only that no transient faults occur, because the continued execution of the protocol is key to maintaining the self-stabilization properties of the protocol. Nevertheless, from the point of the nodes, once they find a clique that is good enough, they might consider leaving the system with the best clique they managed to form so far.

4.2. FORMED CLIQUES LEAVING THE SYSTEM

Taking into account the observations made in the previous section, we explore here the scenario in which nodes try to find satisfactory cliques as fast as possible by devoting to this end as little computational resources as possible. Moreover, once the nodes find a clique that is good enough, they form a consensus to leave the system. The decision about leaving the system benefits not only the leaving nodes that by doing so do not spend any more resources on execution of the protocol, but also the nodes that remain in the system, as these nodes have subsequently fewer neighbors to consider for fellow clique members.

In order to proceed with this scenario, the nodes need a method of determining whether the clique is good enough to stop looking for a better option. The bottom line is, of course, that all nodes belonging to the clique, point to this clique as their current choice. But how can the nodes figure out that the clique's weight is large enough? Naturally, if the nodes have the full information about the network with all edge weights between any two nodes and enough computational resources, they could compute the best proper clique at a time (just as the nodes do in the basic version of the algorithm), or they could even single-handedly compute the final k-clique matching to find out what their final clique would be. Yet, as we mentioned, this would require extensive computations that, as we have assumed, the nodes would prefer to avoid. One possible inexpensive approach would be for a node to set a threshold on the acceptable clique weight. Unfortunately, this approach has a large drawback, because if the threshold is set too high it might be impossible to reach, and if it is set too low, the node will settle for a mediocre clique when better options exist.

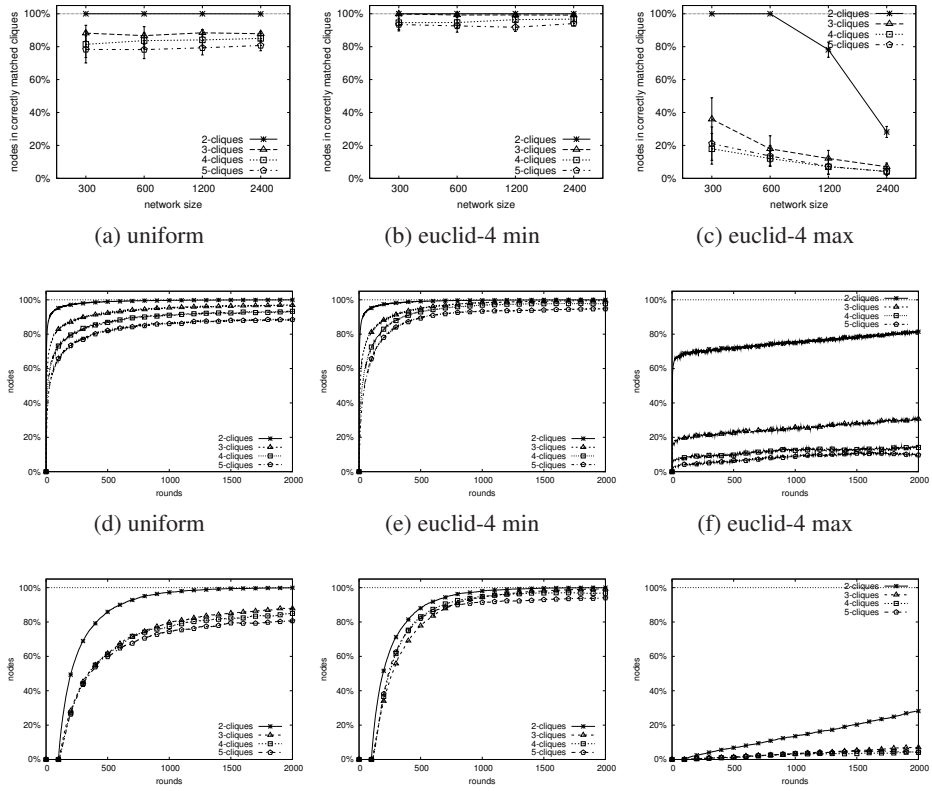
Instead, a node can try to predict whether it has chances on finding a better clique based on what has been happening to its clique choices in the previous rounds. For example, if for the past r rounds the node has been stuck with the same clique choice, and also other members of that clique have remained steadfast with the choice of that clique, a node can infer about the probability of further improvement. In short, the larger the value of r, the smaller the probability that there exists a better clique option for the node. And this probability encompasses not only the effectiveness of the node's heuristic in finding a more attractive clique, but also the effectiveness of heuristics from all of the node's neighbors. Therefore, by setting the lower bound on the number of rounds after which a node is willing to stop the search and settle for the clique that has persisted as its choice throughout this bound, the node is avoiding staying in the system when any further improvement is unlikely. Moreover, by leaving the nodes are also helping their neighbors who are still remaining in the system, because the remaining nodes' neighborhoods shrink, which also decreases the number of all node combinations among which the heuristic has to search for the best clique.

Naturally, before the nodes sharing the same clique leave the system, they should reach a unanimous agreement on doing so. As our deliberations here are of purely theoretical nature and our main goal is to investigate the impact of the proposed approach on the convergence properties of the protocol, we abstract here from any particular consensus algorithm that the nodes could use.

4.2.1. Experimental Results

The general experimental settings with regard to graph topologies, edge weights, clique sizes and weights, and simulations are identical to the ones for the experiments with heuristics detailed at the beginning of Section 4.1.4. Moreover, we run the experiments for both randomized heuristics discussed in this chapter. For RANDOMSUBSET, the size of the subset, s, depends on the size of the network according to the formula: $s = \lfloor |V|/k \rfloor$, which for our selected network sizes of 300, 600, 1200, and 2400 gives 8, 9, 10, and 11 respectively. The size of the subset directly determines the number of neighbor combinations examined by RANDOMSUBSET in a single execution, which amounts to $\binom{s}{k-1}$. Thus, to keep the comparison between RANDOMSUBSET and VNS fair, we set the stopping condition for VNS also to $\binom{s}{k-1}$ of neighbor combinations per execution.

Before we discuss the results of the simulations when nodes are allowed to leave the system, let us first take another look at the results of simulations of our k-clique matching algorithm with randomized heuristics, but this time let us expand our view to various clique and network sizes. Figures 4.9 and 4.10 present results for VNS and RANDOMSUBSET from three different perspectives. The graphs from the first row depict the percentages of nodes that remained in the same correct



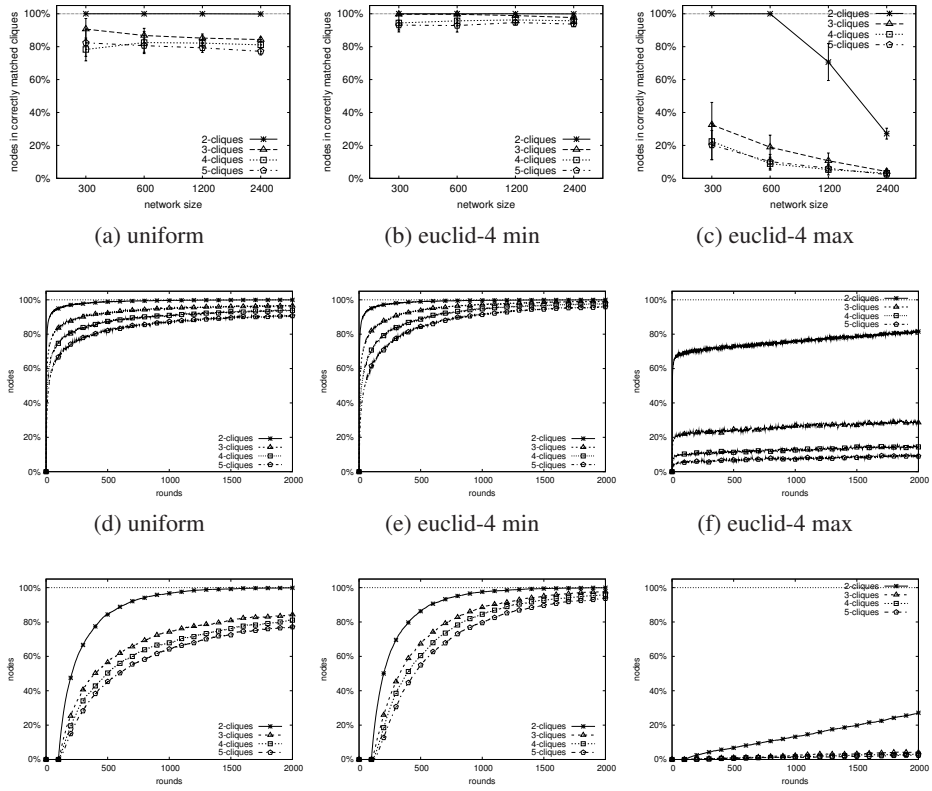
(a) uniform (b) euclid-4 min (c) euclid-4 max

(d) uniform (e) euclid-4 min (f) euclid-4 max

(g) uniform (h) euclid-4 min (i) euclid-4 max

Figure 4.9: Performance of the k-clique matching algorithm using VNS in three selected weight distributions: (a)–(c) percentage of nodes that remained in the same correct cliques for the last 100 rounds of simulations, (d)–(f) percentage of nodes in correctly matched cliques by round in 2400-node networks, (g)–(i) percentage of nodes that have remained in the same correct cliques for the previous 100 rounds.

cliques for the last 100 (out of 2000) rounds of simulations. We can see that the situation after 1900 rounds for $k > 2$ is still fairly dynamic and that some nodes (and in the case of euclid-4 max, the majority of the nodes) are still looking for new cliques and switching their choices, which is a clear indication that the stable state has not been reached. On the other hand, the graphs from the second row show that the nodes in 2400-node networks do not have any major problems with forming correct cliques, and that at the beginning of the simulations in less than 100 rounds the percentage of nodes forming correctly matched cliques increases



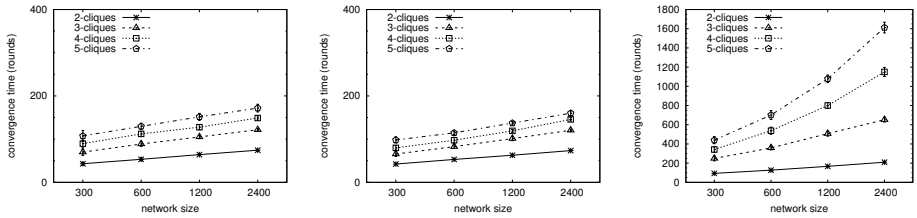
(a) uniform (b) euclid-4 min (c) euclid-4 max

(d) uniform (e) euclid-4 min (f) euclid-4 max

(g) uniform (h) euclid-4 min (i) euclid-4 max

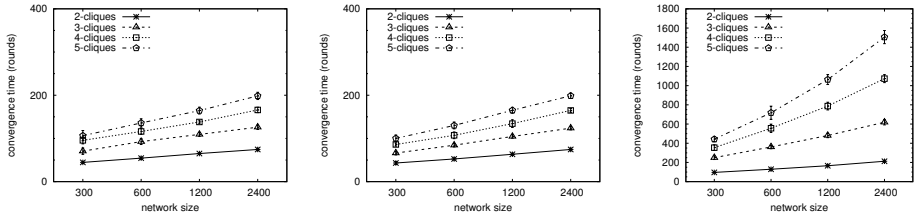
Figure 4.10: Performance of the k-clique matching algorithm using RANDOM-SUBSET in three selected weight distributions: (a)–(c) percentage of nodes that remained in the same correct cliques for the last 100 rounds of simulations, (d)–(f) percentage of nodes in correctly matched cliques by round in 2400-node networks, (g)–(i) percentage of nodes that have remained in the same correct cliques for the previous 100 rounds.

fast to the level that in the remaining rounds improves only slightly. Moreover, if we take a closer look at the stability of the correctly matched cliques, as depicted by graphs from the third row, we observe that a number of nodes which remain in the same cliques for at least 100 consecutive rounds increases almost just as fast. This suggests that nodes should not have problems with fulfilling the criteria that would allow them to leave the system. Moreover, although for the euclid-4 max distribution the percentages of nodes in correctly matched cliques are significantly lower than in the other two distributions, and the percentages of nodes remaining



(a) uniform (b) euclid-4 min (c) euclid-4 max

Figure 4.11: Convergence times for the k-clique matching algorithm using VNS when nodes leave after they are for at least 10 rounds in the same clique.

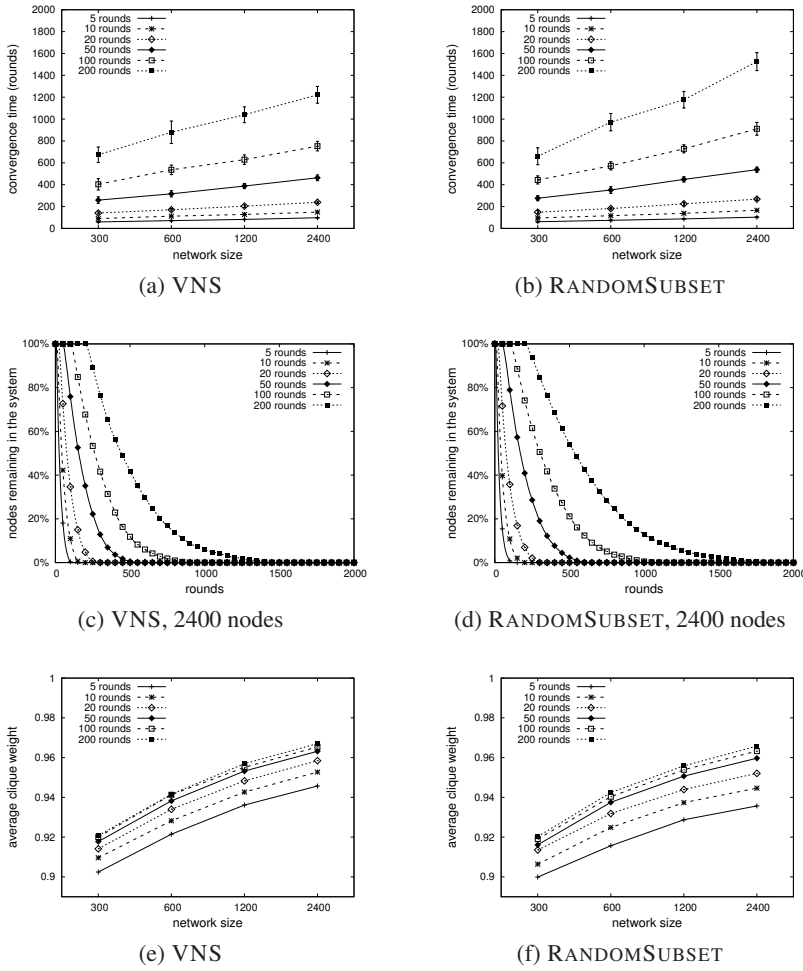


(a) uniform (b) euclid-4 min (c) euclid-4 max

Figure 4.12: Convergence times for the of k-clique matching algorithm using RANDOMSUBSET when nodes leave after they are for at least 10 rounds in the same clique.

in such cliques for more than 100 rounds in case of $k > 2$ do not exceed 10% throughout the entire 2000-round simulations, we will soon see that even in such cases our approach works and can largely improve the convergence of the system.

We evaluate the performance of the algorithm with nodes leaving after r rounds based on the convergence time, which is measured in the total number of rounds from the beginning of the simulation till the last change in the variables of any of the nodes. Thus, the last nodes leave the system precisely r rounds after the convergence. Figures 4.11 and 4.12 provide the comparison between the average convergence times across various clique sizes, network sizes and edge-weight distributions. For uniform and euclid-4 min distributions nodes need only up to 200 rounds to converge even for the largest network size of 2400 nodes and for the biggest clique size of 5. Moreover, the increase in the convergence time is less than linear (note the logarithmic scale of the x-axis), which implies that our approach scales well. The convergence times for euclid-4 max distribution are much larger and amount to around 1600 rounds for $k = 5$ in 2400-node networks. Yet, the degradation in the performance in this distribution is obvious if we recall the



(a) VNS (b) RANDOMSUBSET

(c) VNS, 2400 nodes (d) RANDOMSUBSET, 2400 nodes

(e) VNS (f) RANDOMSUBSET

Figure 4.13: Performance comparison of k-clique matching algorithm using VNS and RANDOMSUBSET for $k = 4$ when nodes leave after they are for at least r rounds in the same clique for various values of r in graphs with uniform edge-weight distribution.

differences in the performance between euclid-4 max and the other two distributions for VNS and RANDOMSUBSET depicted in Figures 4.9 and 4.10. In fact, the results for the euclid-4 max distribution when nodes are allowed to leave are surprisingly good if we take into account that without nodes leaving the percentage of nodes in correctly matched cliques stayed well under 20% throughout the entire duration of the simulations for $k = 4$ and $k = 5$.

In Figure 4.13 we take a closer look at the differences in the convergence times when different values of r are chosen. As expected, the increase in the value of r results in the increase of the convergence time (see Figures 4.13a and 4.13b). The larger the value of r, the longer the nodes wait before they leave the system, thus the larger the chances that they will find in the meantime a better clique and therefore stay in the system longer. Yet, irrespective of the value of r we can see that the convergence times scale nicely with the doubling of the network sizes. The explanation for this is depicted in Figures 4.13c and 4.13d where we can see for each round the percentages of nodes still remaining in the system. After the initial r rounds, when no node is allowed to leave, we can see rapid flow of the nodes from the system, such that in no time the number of nodes remaining in the system is reduced by half. Finally, Figures 4.13e and 4.13f show the impact of the value of parameter r on the average weight of a clique in the converged state. Indeed, the graphs support our claim that the longer the nodes have to wait before they can leave the system, the larger is the probability that at least one of them will find a better clique. Moreover, we can see that these chances for individual improvements have a visible impact on the total weight of the final k-clique matching. Although the improvement of the total weight of the final k-clique matching is encouraging, we should realize that the dependency between this weight and the value of r is not bound to occur, because if one node uses the additional rounds to find a better clique, its change of the decision can result in the deterioration of the state of other nodes in the network.

4.3. PRUNING

To improve the performance of the randomized heuristics, a node can try to exploit its knowledge about the network. For example, if based on the available information, node v could assess for each neighbor whether there is a chance to form with this neighbor a proper clique, it could temporarily ignore the neighbors that by no means could be part of a viable solution. This way, a node could pass to the heuristic a truncated set of neighbors, in the hope of increasing the chances of the heuristic to find the most attractive clique. From the global perspective, this could result in the improvement of the convergence speed.

Information that a node could exploit to this end is, for example, the fact that edge weights are bounded. In such a case, node v can compute for each of its neighbors the maximum weight of a clique that these two nodes could create irrespective of other nodes that would fill the remaining places in such a clique. To do that, v only needs to know 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:

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

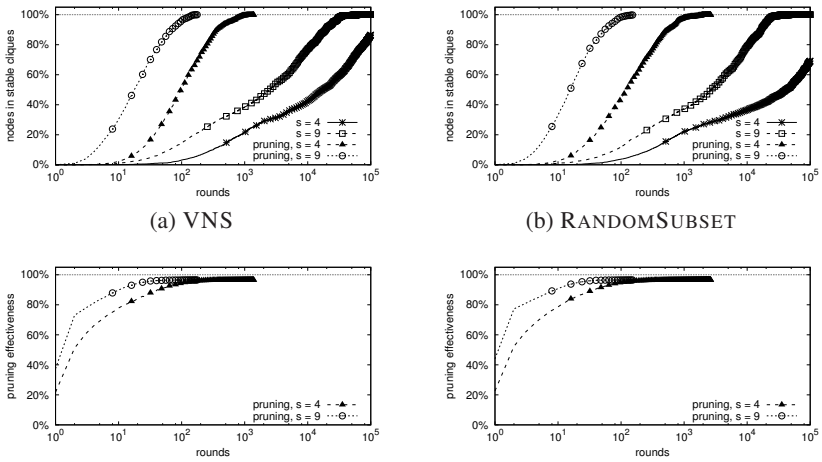
The value w' can then be further used to judge whether neighbor u has any chances to become a partner in the newly recomputed clique. If w' is smaller than either $\mathbf{w}_v$ or $\mathbf{w}_u$, node v can safely ignore this neighbor for one of two reasons. First, when $w' < \mathbf{w}_v$ and $\mathbf{C}_v$ is *proper*, 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.

It is important to realize that when a node prunes its neighbors based on such criteria, it will never prune a neighbor with which it could create a proper clique, thus no attractive cliques are lost. Therefore, the final k-clique matching created by the algorithm that uses a heuristic in combination with pruning and the k-clique matching created by the algorithm that uses only the heuristic will be identical. What is different is, that by pruning the neighbor set before passing it to the heuristic, the node limits the solution space available to the heuristic, which can possibly increase the chances that the heuristic returns a better clique, and this in turn will lead to the shortening of the convergence time. 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, we should expect that pruning will bring the best results only for small values of k.

Note that this pruning technique can be also applied to the basic version of the k-clique matching algorithm, in which case the benefit will not be related to the convergence time which will remain unchanged, but to the decrease in the computational load of a single round.

4.3.1. Experimental Results

Let us start our evaluation of pruning with a small example of $k = 3$ in a 600-node network with uniform edge-weight distribution. In Figures 4.14a and 4.14b we captured the process of creation of stable cliques for each of the randomized heuristics VNS and RANDOMSUBSET given the two computational powers granted to the heuristic amounting to $\binom{s}{2}$ neighbor combinations per round for $s = 4$ and $s = 9$. The difference between the number of rounds that the algorithms



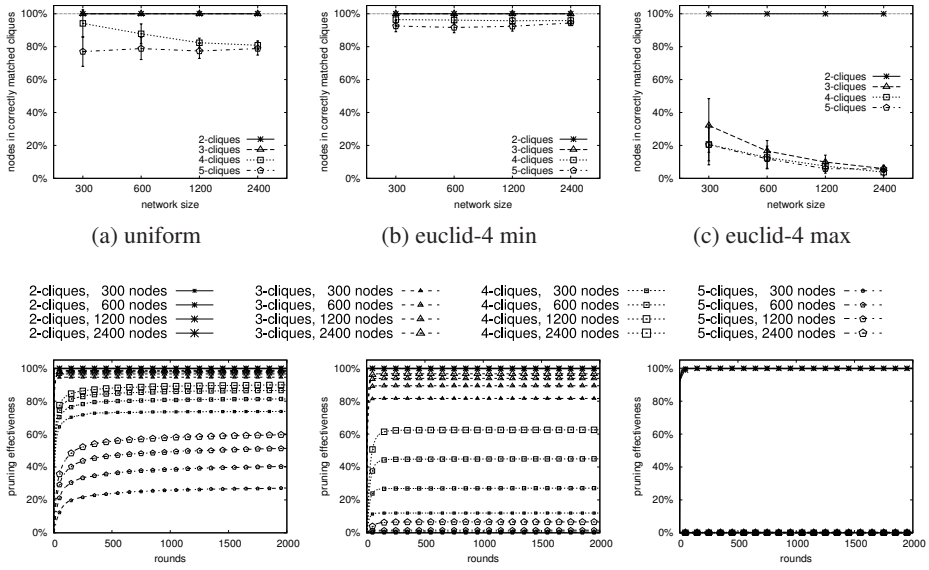
(a) VNS (b) RANDOMSUBSET

(c) VNS (d) RANDOMSUBSET

Figure 4.14: Performance of k-clique matching algorithm using VNS and RANDOMSUBSET with pruning for $k = 3$ in 600 node network: (a)–(b) percentage of stable cliques per round, (c)–(d) effectiveness of pruning as an average percentage of neighbors pruned.

need whether they use pruning or not is in several orders of magnitude (note the logarithmic scale of the horizontal axis). The reason of these differences can be ascribed directly to the effective pruning of the neighbors in each round. As Figures 4.14c and 4.14d depict, the average percentage of neighbors that is ignored reaches quickly over 90%, which means that less than 60 neighbors are passed to the heuristic in each round, decreasing the solution space from $\binom{599}{2}$ to less than $\binom{60}{2}$, a hundredfold reduction.

Figures 4.15d–4.15f and 4.16d–4.16f show the effectiveness of pruning for various clique sizes and network sizes. As we expected the average number of neighbors that each node is capable of pruning is inversely proportional to the clique size. We can observe, for example, that in the uniform distribution the pruning effectiveness for $k = 2$ is close to 100%, for $k = 3$ it is above 90%, while for $k = 4$ it ranges from 70% to 90% and for $k = 5$ from 25% to 60%. As a result, 1900 rounds of simulations prove insufficient for $k = 4$ and $k = 5$ in order for the networks to converge, and we can see that in the subsequent 100 rounds there are still many nodes (even over 20%) actively seeking for a clique (see Figures 4.15a and 4.16a). The situation appears more stable for the euclid-4 min distribution, although the pruning effectiveness is much lower. Yet, this does not have to be surprising, as we have seen in Section 4.1.4 that the heuristics VNS and RANDOMSUBSET performed better in euclid-4 min distribution than with a uniform



(a) uniform (b) euclid-4 min (c) euclid-4 max

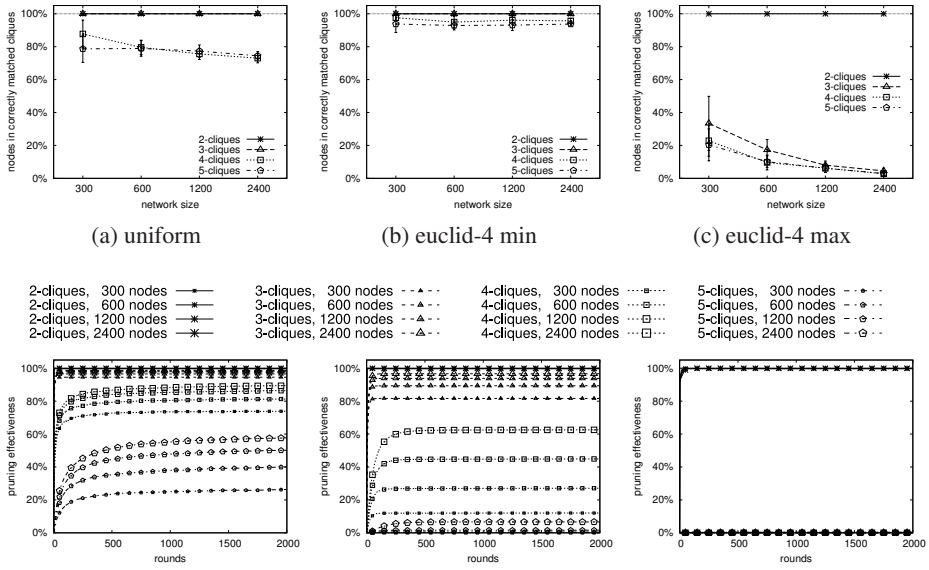
(d) uniform (e) euclid-4 min (f) euclid-4 max

Figure 4.15: Performance of k-clique matching algorithm using VNS with pruning in three selected weight distributions: (a)–(c) percentage of nodes that remained in the same correct cliques for the last 100 rounds of simulations, (d)–(f) effectiveness of pruning as an average percentage of neighbors pruned.

distribution, also without the pruning.

Even though pruning was not sufficient for nodes in our simulations to fully converge within 2000 rounds when $k = 4$ or $k = 5$, this does not mean that pruning has no effect on the convergence for $k >= 4$ with uniform or euclid-4 min distributions. In each round that a node prunes even a small portion of its neighbors, the smaller set of neighbors is passed to the heuristic, increasing the chances of the heuristic finding a better clique option.

Finally, we can see that apart from $k = 2$ pruning appears to have no impact on the reduction of the neighbor sets in euclid-4 max distributions. The reasons for this ineffectiveness can be sought directly in the distribution of edge weights. The main cause lies in the fact that the clique weights that nodes form are too small to allow nodes to prune their neighbors effectively. In Figure 4.17 we can see the average weights of cliques that are correctly matched at the end of each round. Taking into account the standard deviation of these weights, we expect the great majority of them to be beyond 0.66. This means that when node v computes $w' = (w\{v,u\} + 2w_{max})/3 = (w\{v,u\} + 2)/3 \geq 2/3$ for each of its neighbors, u,



(a) uniform (b) euclid-4 min (c) euclid-4 max

(d) uniform (e) euclid-4 min (f) euclid-4 max

Figure 4.16: Performance of k-clique matching algorithm using RANDOMSUBSET with pruning in three selected weight distributions: (a)–(c) percentage of nodes that remained in the same correct cliques for the last 100 rounds of simulations, (d)–(f) effectiveness of pruning as an average percentage of neighbors pruned.

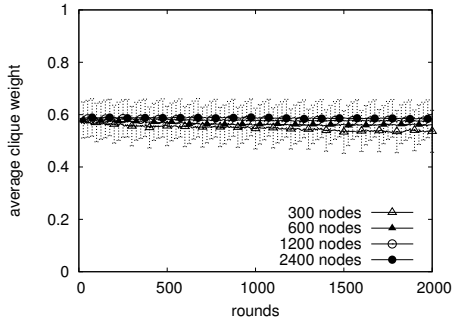


Figure 4.17: Average weight of correctly matched cliques created by k-clique matching algorithm with VNS and pruning for $k = 3$ in euclid-4 max distribution.

most probably w' is higher than both $\mathbf{w}_v$ and $\mathbf{w}_u$, and thus u is not pruned. For larger values of k, w' also grow, while the average clique weights stay on the similar level as for $k = 3$.

4.4. PARTIAL VIEWS

When any two nodes in the network can be neighbors, then a node's neighbor set 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 of finding the best clique high.

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 heuristic for finding the best k-clique — in each round a node has a small subset of all neighbors available, and 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 4.1, and the efficiency of clique matching will largely depend on the policies for modifying and using 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 [KvS07] is a simple, lightweight and robust type of peer-to-peer protocol that found its application in information dissemination, distributed computations, resource management, and most importantly from our perspective, peer sampling and topology construction. The framework of a gossiping protocol used for creation and maintenance of various overlays is summarized in Figure 4.18. Here, each node keeps a list of nodes of a specific small size c. We refer to this list as a partial view. 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. Here, we show how we can use two existing implementations of a gossiping protocol, CYCLON [VGvS05] and VICINITY [VvSI07] to achieve two different behaviors of our k-clique matching algorithm. For reference, an overview of the implementations of these two protocols is provided in Figure 4.19.

The idea behind using CYCLON, is to provide enough variety of the partial view contents for the subsequent executions of our k-clique matching protocol. CYCLON, apart from maintaining a connected overlay, can provide the upper-layer protocol with nodes uniformly selected from the entire network. This enables the k-clique matching protocol to discover part by part all nodes in the network and explore all possible neighbor combinations. In fact, by polling from time to time

Active thread loop:

```
u ← selectNode()
buffer_out ←
    selectItemsToSend()
send buffer_out to u
receive buffer_in from u
selectItemsToKeep()
```

Passive thread loop:

```
receive buffer_in from v
buffer_out ←
    selectItemsToSend()
send buffer_out to v
selectItemsToKeep()
```

Figure 4.18: Gossiping protocol framework.

the partial view maintained by CYCLON, the k-clique matching protocol should behave as if it was using RANDOMSUBSET to look for cliques. As a result, we could substitute the requirement of the full knowledge of all nodes in the network with a requirement to know only a small, constantly changing subset of the nodes, without any loss in the performance.

At the same time, the drawback of the above approach is that the convergence of the k-clique matching can take extremely long. Conversely, the goal of using VICINITY is to quickly collect from the entire network nodes that have the heaviest edge connecting them to the node owning the partial view. As VICINITY used on its own can lead to the partitioning of the network, it is recommended to run it on top of another protocol, such as CYCLON, (see Figure 4.20) to keep the network connected, and also to feed VICINITY with random nodes that can help in the convergence of VICINITY. Once VICINITY converges, its partial views become fixed. From that moment, from the perspective of k-clique matching operating on top of VICINITY, the examination of best neighbors drawn from VICINITY's partial views becomes equivalent to the HEAVIESTEDGESUBSET heuristic.

4.4.1. Experimental Results

CYCLON partial views

In our first experiments with partial views, we compare the performance of our k-clique matching algorithm that obtains small neighbor subsets from the CYCLON layer to the performance of our algorithm that uses the RANDOMSUBSET heuristic. In these experiments we have chosen relatively small values of parameters: $k = 3$ and a network size of 600 nodes, which are consistent with the parameters chosen for experiments on RANDOMSUBSET presented in Figure 4.14d. For the CYCLON protocol, we have set the partial view capacity, c, to 18 ($= 2 * \lfloor log_2(|V|) \rfloor$) nodes and the exchange buffer size of 9. The k-clique

In both protocols, items exchanged by nodes are tuples containing:
- contact information of the item's creator, node P,
- application specific information on node P,
- age of the item.

(a) Syntax of exchanged items

Hook	Description
`selectNode()`	increase the ages of all items in the partial view by 1 and select the item with the highest age
`selectItemsToSend()`	
active thread:	randomly select $b-1$ items from the protocol's view and add own item with age 0
passive thread:	randomly select b items from the protocol's view
`selectItemsToKeep()`	keep all b received items, removing (if necessary) the items selected to send (and in case of *active thread*, the item returned by `selectNode()`); if two items have the same creator, keep only the item with smaller age

(b) Cyclon specification

Hook	Description
`selectNode()`	increase the ages of all items in the partial view by 1 and select the item with the highest age
`selectItemsToSend()`	combine items from the partial views of CYCLON and VICINITY and add own item with age 0 into one view; from this view select b items of nodes that have the heaviest edge connecting them to the recipient
`selectItemsToKeep()`	out of all items from the received buffer and current views of CYCLON and VICINITY, keep c items of nodes that have the heaviest edge connecting them to the node;if two items have the same creator, keep the item with smaller age

(c) Vicinity specification

Figure 4.19: Implementation details of CYCLON and VICINITY protocols.

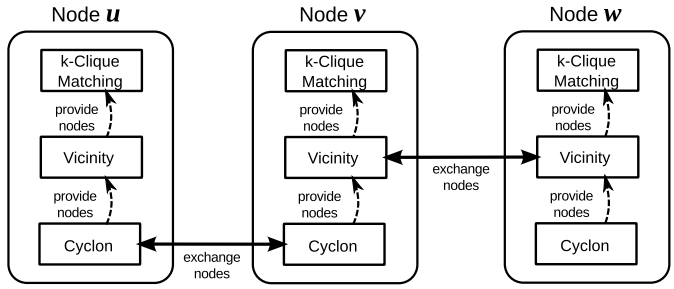


Figure 4.20: The layered framework.

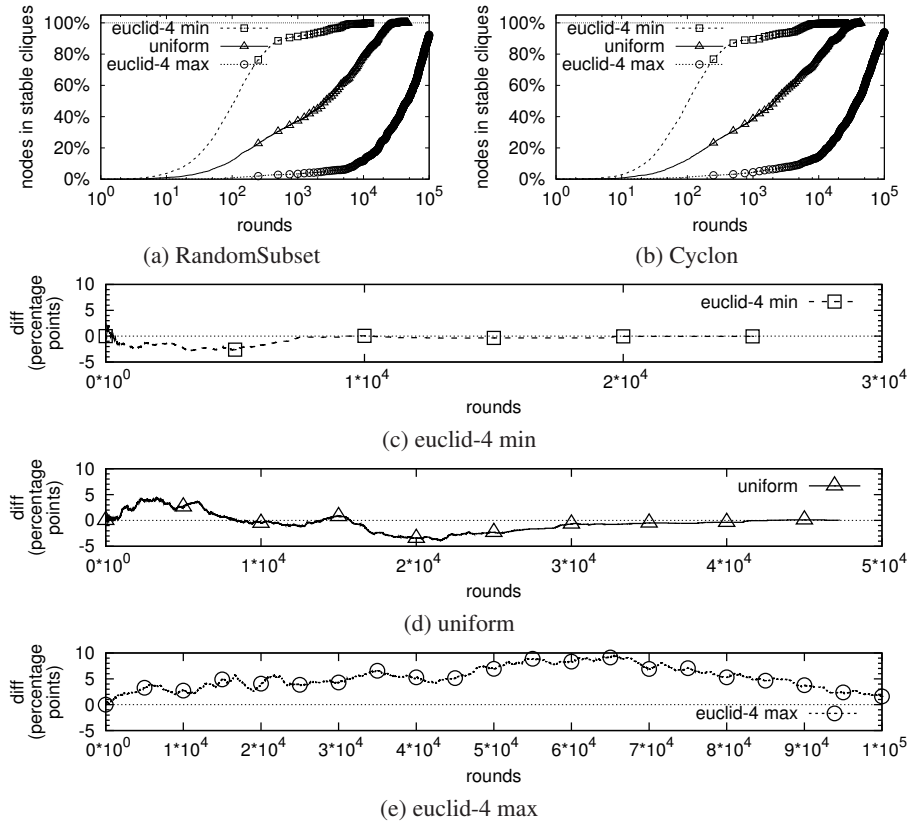


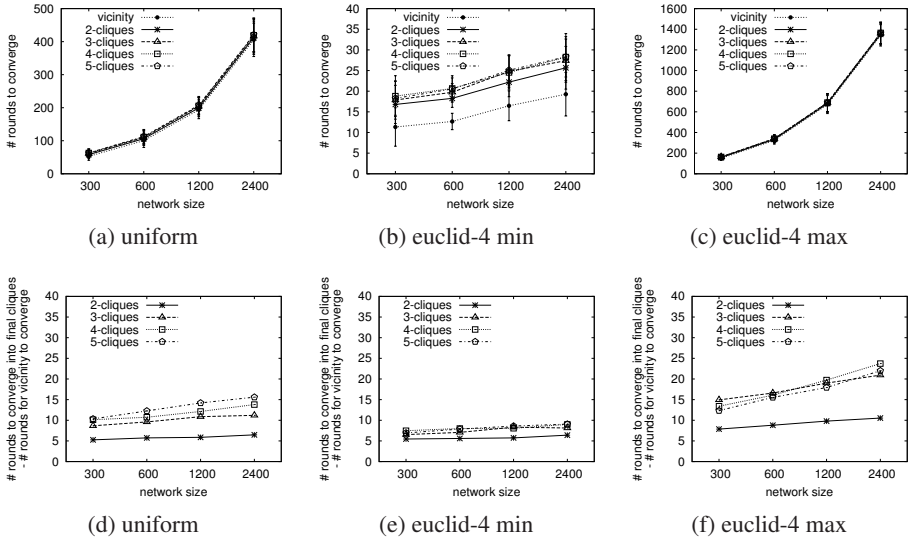
Figure 4.21: Stable cliques for $k = 3$ in 600-node network.

matching protocol and CYCLON run in parallel, but the frequency of CYCLON is five times higher allowing for almost complete replacement of cache contents with new items between two executions of the k-clique matching protocol. Each time

the *k*-clique matching protocol is executed the CYCLON partial view is polled for up to 9 random nodes to construct a subset of 9 nodes to be examined in the same manner as the RANDOMSUBSET heuristic constructs its subset from the entire node set.

Figures 4.21a and 4.21b present side-by-side the progression of stable cliques creation for RANDOMSUBSET and CYCLON-based *k*-clique matching protocols. We observe that for any of the three edge-weight distributions, there are no major discrepancies in the percentages of nodes in stable cliques formed by the two versions of the protocol. In Figures 4.21c–4.21e we zoom into the differences between the percentages of stable cliques of the CYCLON-based version of the protocol and the RANDOMSUBSET one. We see that for the uniform and euclid-4 min distributions, the differences between the two protocols do not exceed 5 percentage points (pp). For the euclid-4 max distribution, the percentage of stable cliques formed by the CYCLON-based version of the protocol exceeds for the major part of experiments 5pp, rising even up to 10pp, but we are inclined to attribute this large difference to a small sample of simulations (15) rather than claim a superior performance of the CYCLON-based version of the protocol. The behavior of any of the two protocols is highly unpredictable and, especially at the beginning of a simulation, it is difficult to predict how fast the number of stable cliques will grow. On the other hand, once one of simulations gets ahead, it will keep the lead for some time before other simulations catch up. This is what might have happened in our simulation. Naturally, at the later stage of simulations, as the number of stable cliques in all simulations approaches 100%, the difference becomes smaller until it reaches 0 once all simulations converge.

All of the graphs in Figures 4.21 confirm our claim that we can replace the full knowledge of the network with only a partial view of it without loss of performance for the *k*-clique matching. This means that from our protocol's perspective, there is no apparent difference between using RANDOMSUBSET which draws the nodes from the entire set of neighbors and drawing the same number of nodes from the partial view provided by CYCLON. Furthermore, this implies that we can use next to CYCLON partial views, the same techniques that we used in previous parts of this chapter to improve the performance of RANDOMSUBSET. For example, we can let the nodes leave the system once they have found a clique that is good enough, and let CYCLON handle the removal of leaving nodes from partial views. Additionally, when selecting nodes from the partial view, a node can first prune from the view all the nodes that do not provide a chance of creating a prospective clique. The neighbor subset for the further examination is then created only from the remaining nodes.



(a) uniform (b) euclid-4 min (c) euclid-4 max

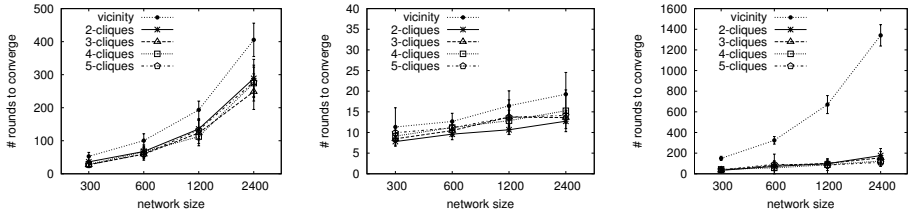
(d) uniform (e) euclid-4 min (f) euclid-4 max

Figure 4.22: Convergence times of VICINITY and of k-clique matching using converged VICINITY's partial views as the only source of neighbors.

VICINITY partial views

For our experiments with the VICINITY protocol we deploy the parameter settings as follows. For CYCLON we set the partial view size, c, to $\lfloor log_2(|V|) \rfloor$ and the exchange buffer size, b, to $\lfloor \frac{1}{2}c \rfloor$. Respectively for VICINITY, we set $c = 2 * \lfloor log_2(|V|) \rfloor$ and $b = \frac{1}{2}c$. These particular parameter settings are based on reported experiences with gossip-based protocols (e.g., see [BGFvS09] for the calculation of the optimal size of the exchange buffer). Lastly, we allow our k-clique matching protocol to retrieve from VICINITY's partial view only $s = \lfloor log_2(|V|) \rfloor$ nodes whose edges connecting them to the given node are highest. Moreover, while presenting the result of the experiments we express the convergence of the VICINITY protocol in the terms of the k-clique matching protocol; we consider a node's VICINITY partial view converged if it contains s most optimal node's neighbors from the entire network. As a result, once the VICINITY's partial view is converged, the k-clique matching protocol will always retrieve the same set of nodes from VICINITY's view. Along the same line, the level of VICINITY's convergence is measured by counting how many of the s most optimal node's neighbors have made it to the node's VICINITY partial view and by providing the average percentage.

In our first experiment, of which the results are presented in Figure 4.22, we present a preliminary investigation of the convergence times of VICINITY and of



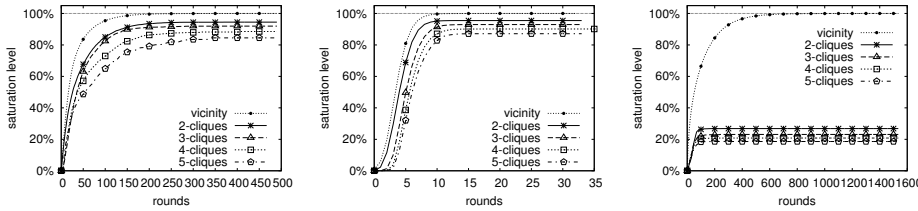
(a) uniform (b) euclid-4 min (c) euclid-4 max

Figure 4.23: Comparison between convergence times of VICINITY and of k-clique matching when they run concurrently.

our k-clique matching protocol running on top of it. In these experiments the execution of the two protocols is fully separated; the k-clique matching protocol starts its execution only when VICINITY has fully converged. Firstly, we see that the convergence times of VICINITY vary largely among the three distributions (see Figures 4.22a–4.22c). The best results in terms of individual convergence times as well as scalability properties, is achieved by VICINITY for the euclid-4 min edge-weight distribution. This should hardly be a surprise, as VICINITY has been designed for very specific topologies. These topologies are characterized by a *transitivity* property, which simply refers to a situation where if node v_2 is a "good fit" for v_1 and node v_3 is a "good fit" for v_2, then v_3 tends to be a "good fit" for v_1, too. Among the three distributions only euclid-4 min exhibits this property.

Yet, a more important observation here is that the k-clique matching protocol convergence times are relatively small in comparison to the convergence times of VICINITY. In fact, once VICINITY is converged, the k-clique matching protocol converges in less than 25 rounds (see Figures 4.22d–4.22f). These convergence results are very consistent with the results we have obtained for the HEAVIESTEDGESUBSET heuristic. This is actually what we expected from the beginning as the k-clique matching protocol running on top of VICINITY transforms to HEAVIESTEDGESUBSET once the VICINITY converges.

Although considering VICINITY and our k-clique matching protocol separately gave us a good insight into the differences between their respective convergence times, a more realistic scenario is when the two protocols run concurrently. In Figure 4.23 we present the comparison between the convergence times of the two protocols in such a scenario. The most interesting observation is that the average convergence times of the k-clique matching protocol are consistently smaller than the average convergence times of VICINITY. Naturally, in the worst case, we expect the k-clique matching protocol to converge after VICINITY converges, because the changes to VICINITY's partial views can result in changes in the cliques



(a) uniform (b) euclid-4 min (c) euclid-4 max

Figure 4.24: Performance of VICINITY and k-clique matching in 2400-node network. In the case of VICINITY the curve depicts the average saturation level of partial views with respective node's $s = 11$ best neighbors, while in the case of k-clique matching the curves depict the percentage of nodes in cliques which will remain stable.

formed by the k-clique matching protocol. Yet, in many cases it seems to be sufficient for VICINITY to be nearly converged for the k-clique matching protocol to reach its final state. This point is illustrated in Figure 4.24, which shows the progress in the convergence round by round. Firstly, we can see that the average percentage of optimal nodes contained in partial views raises very quickly (in relation to the respective convergence time) to almost 100%. This suggests that the large portion of VICINITY's convergence time is spent on looking for the last missing nodes. Secondly, the increase in the level of VICINITY's convergence is closely followed by the increase in the number of stable cliques, which quickly approaches to its final level. Thus, what can be happening here is that the last missing nodes necessary for VICINITY to converge, once discovered, happen to have no impact on the clique formation.

Nonetheless, the most important lesson from the experiments with VICINITY lies in the differences in convergence times of the protocols for various edge-weight distributions. Once again, euclid-4 max turns out to pose the biggest challenge not only to the k-clique matching protocol but also for VICINITY, while in the network with a uniform distribution VICINITY takes also relatively long to converge. The best performance of k-clique matching protocol and VICINITY combination we have achieved in the euclid-4 min edge-weight distributions. This gives us a good indication of the network properties for which the combination of these two protocols will be most efficient.

4.5. 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 $|V|^2$.

Therefore, it might be beneficial to consider other methods of disseminating weights of pursued cliques. One alternative is to 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, however, 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 would replace the '**send** $\mathbf{w}_v$ **to** all $u \in N(v)$' statement from the active thread of the k-clique matching protocol and the '**receive** $\mathbf{w}_u$ **from** any $u \in N(v)$' statement from its passive thread.

This protocol follows the same framework from Figure 4.18 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 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 4.25 we present details of possible rules to incorporate into the three core functions of the gossiping protocol. In Section 4.5.1, 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. Here, we confine ourselves only to the evaluation of the former approach, leaving the evaluation of the latter idea to future research.

Hook	Description
`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 4.25: Implementation details of gossiping protocol for clique weights dissemination.

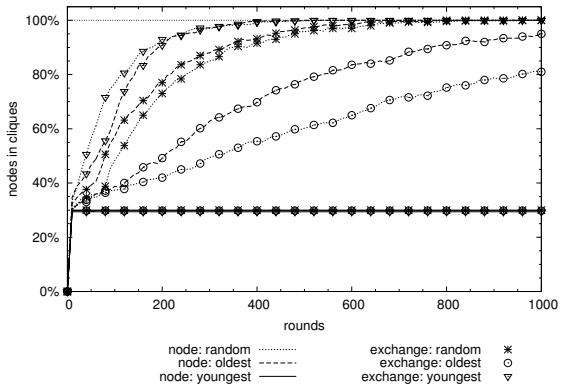


Figure 4.26: Comparison between various implementations of clique weight gossiping core functions.

4.5.1. Experimental Results

We start our evaluation of the gossiping protocol for clique-weight dissemination by comparing the effectiveness of various implementations of the two core methods `selectNode()` and `selectItemsToSend()`. In Figure 4.26 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

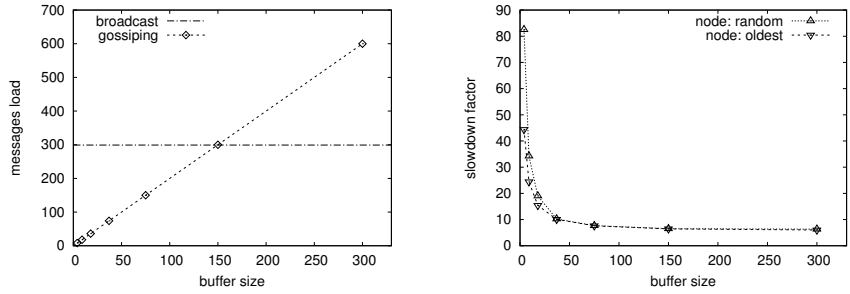
this information. Each iteration of the k-clique matching protocol loop is followed by a single execution of the active thread loop of the gossiping protocol. The simulations have been performed on a network of 300 nodes and for $k = 3$. The size of the buffer for the gossiping protocol has been set to 10. Each curve corresponds to a different combination of `selectNode()` and `selectItemsToSend()` implementations, as detailed in Section 4.5.

In Figure 4.26 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 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.

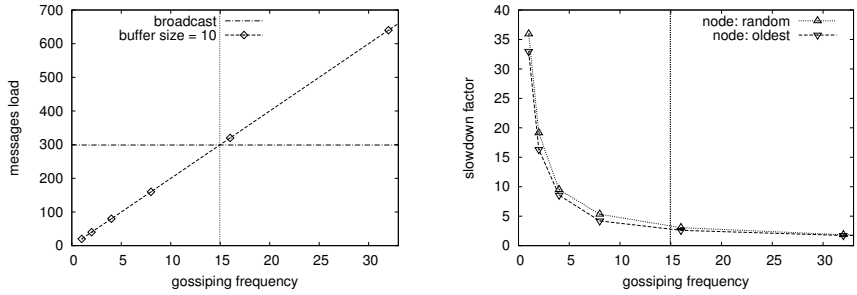
Even with the most efficient combination of `selectNode()` (*oldest* and *random*) and `selectItemsToSend()` (*youngest*) implementations, k-clique matching using gossiping to disseminate clique weight information is significantly (over 30 times) slower than its counterpart that uses broadcast. Nonetheless, the convergence can be improved by increasing the number of items exchanged by nodes between any two executions of the k-clique matching protocol loop. This can be done by either increasing the number of items exchanged in every gossiping communication (Figure 4.27) or by increasing the frequency of the gossiping protocol relatively to the k-clique matching protocol (Figure 4.28).

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 on average



(a) (b)

Figure 4.27: Comparison between broadcasting of clique weights and clique-weight gossiping with various buffer sizes: 4.28a average number of clique-weight items sent inbetween two executions of k-clique matching algorithm, 4.28b ratio between convergence times of k-clique matching algorithm with clique-weight gossiping versus clique-weight broadcast.



(a) (b)

Figure 4.28: Comparison between broadcasting of clique weights and various frequencies of clique-weight gossiping: 4.28a average number of clique-weight items sent inbetween two executions of k-clique matching algorithm, 4.28b ratio between convergence times of k-clique matching algorithm with clique-weight gossiping versus clique-weight broadcast.

the number of items equal to twice the size of the gossiping buffer. The differences between the message load of broadcast and gossiping with various sizes of the exchange buffer for a 300-node network are depicted in Figure 4.27a, while Figure 4.27b shows the factor by which the convergence time of k-clique matching algorithm with clique-weight gossiping is larger than with broadcasting. We can observe that for very small buffer sizes, this factor is fairly large, and that also

the difference between selecting *oldest* or *random* node in `selectNode()` is the biggest. For example, for buffer size equal to 4, it takes over 40 (in the case of *oldest*) and over 80 (in the case of *random*) times longer for the *k*-clique matching algorithm to converge. The difference in performance between two modes of `selectNode()` quickly diminishes with the increase of buffer size. Similarly, the ratio between convergence times of *k*-clique matching algorithm with gossiping or broadcasting becomes smaller, falling to a little above 10 for buffer size of 37 and stabilizing just above 6 for buffer sizes larger than 150.

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 4.28a: the network size equals 300 and the size of the gossiping buffer is set to 10. Furthermore, in Figure 4.28b we can see that the convergence speed of the *k*-clique matching algorithm with gossiping of clique weights improves exponentially, and for the frequency above 8 the difference in convergence time between using gossiping and broadcasting falls below factor of 5.

To sum up the comparison of the two methods for improving the clique-weight gossiping, we can clearly see that, while sending the same number of clique-weight items, increasing the frequency of gossiping rounds wins over increasing the buffer size by factor of 2. This can be attributed to the fact that with the increasing frequency, a gossiping node contacts growing number of neighbors, and the communication pattern gets closer to broadcasting. Moreover, consider only message load below 300 clique-weight items, which is equivalent to the message load for broadcasting. In such a case, neither of the gossiping-improvement approaches managed to achieve the same convergence for *k*-clique matching algorithm as broadcasting did. This is understandable, because with gossiping under such constraints there will always exist a delay in delivery of new information to majority of the nodes in the network.

Lastly, it is worth pointing out how easy it would be to combine the functionality of partial views management and clique weights dissemination into one protocol. If we look at Figure 4.25 with the various modes of implementing a gossiping protocol for clique-weight dissemination, we notice that choosing the oldest node in `selectNode()` and random nodes in `selectItemsToSend()` are the strategies also used by CYCLON, although in CYCLON they are applied to a partial view which is restricted in size. As the curves in Figure 4.26 suggest, the combination of these two strategies does not provide the most effective regime for clique-weight dissemination, yet it is very close to the top, right after the other two regimes that always select the youngest information to exchange. Therefore, it seems reasonable to let CYCLON handle both tasks. This way, CYCLON can

single handedly serve the k-clique matching algorithm both as a discovery service of potential clique partners and as a dissemination medium of pursued clique weights.

4.6. CONCLUDING REMARKS

The two previous chapters focused mainly on the theoretical aspects of our k-clique matching algorithm. In turn, in this chapter we looked at our algorithm from a more practical perspective. We decided to push our algorithm to its limits and assumed a scenario in which the network forms a complete graph, i.e., the neighbor set of each node consists of all other nodes in the network. This scenario instantly creates challenges in terms of: (i) high computational complexity of a single round, (ii) necessity of algorithm-specific data management (neighbor sets management), and (iii) high communication costs.

We proposed to tackle the first problem of high computational complexity of a single round by replacing the function that returns the optimal (most attractive) clique with a heuristic. We have provided the proof that irrespective of what heuristic is used, our modified k-clique matching algorithm is bound to converge to a correct k-clique matching. This enables us to set our own limit on the computational complexity of a single round, for example, to bring it down to $O(|V|)$. Nonetheless, there are tradeoffs related to using this approach, and the nature of these tradeoffs depends on the particular heuristic used. We have discussed two types of heuristics. The first type, which we referred to as attractiveness-maximizing deterministic heuristics, traded the smaller computational complexity of a single round for a 50% increase in the upper bound of convergence time of the k-clique matching algorithm and for the loss of quality in the final k-clique matching. The use of the second type of heuristics, i.e., randomized heuristics, does not modify the final k-clique matching, but there is no strict upper bound on the convergence and the expected convergence time can easily become polynomial to the size of the network.

This large expected convergence time of the k-clique matching algorithm combined with the randomized heuristic lead us to the consideration of two possible improvements. In the first improvement, we allow nodes to leave the system if their k-clique choice has remained unchanged for some given number of rounds. The idea behind it is that the longer this waiting period is, the smaller the chances that there exists a better clique for any of these k nodes. By deciding to leave the system, nodes not only save their own computational resources, but also help in this respect the remaining nodes in the system whose neighbor sets become smaller. The second improvement, pruning, aims to increase the efficiency of the

randomized heuristic by limiting a node's neighbor set to only those neighbors whose edge weight connecting them to the node is large enough to suspect that this neighbor is a potential fellow clique member. As a result the expected convergence time of k-clique matching can become shorter. Lastly, nothing stands in the way of combining these two improvements to increase the chances for the nodes to find better cliques even faster.

Our second challenge stemmed from the fact that in a single round of our basic k-clique matching algorithm a node had to consider all neighbor combinations and compute their respective clique weights. This entailed that the node needed to have on a per-round-basis access to the information about its entire neighbor set, especially neighbor-profile information necessary to compute the clique weights. Additionally, this information needed to be kept up-to-date accounting for any changes in neighbors' profiles or their membership in the system. With the introduction of heuristics, the necessity of examining all neighbor combinations was alleviated. Moreover, some heuristics, such as HEAVIESTEDGESUBSET or RANDOMSUBSET, required access to only a small subset of the entire neighbor set per round. Yet, for the proper operation of these heuristics, these small subsets had to meet certain constraints. This is where gossiping protocols came into the picture. We showed how we can use protocols such as CYCLON or VICINITY to provide subsets that do meet these constraints. As a result nodes had to maintain only a partial view of the entire neighbor set, but at the same time these partial views gave the impression that our k-clique matching executed HEAVIESTEDGESUBSET or RANDOMSUBSET as if it had access to the entire neighbor set. Additionally, CYCLON and VICINITY seamlessly took care of facilitating the discovery of changes in neighbor profiles as well as handling neighbors joining or leaving the system.

In the light of utilizing CYCLON to achieve the same behavior of our k-clique matching as if it was executing RANDOMSUBSET, it seems especially important to recall the performance comparison between RANDOMSUBSET and another randomized heuristic, VNS. VNS, according to [BMUN09], is a state-of-the-art heuristic outperforming many other heuristics in finding heaviest k-cliques. Nonetheless, we did not notice major differences in the performance of the two heuristics. This does not undermine the effectiveness of VNS in general, but rather shows that in our particular situation where a heuristic is executed repeatedly and in parallel by many nodes, a simple heuristic, such as RANDOMSUBSET can perform just as well as VNS. And this is obviously very good news for using partial views operated by CYCLON to provide neighbor subsets for our k-clique matching algorithm.

The last challenge revolved around the high communication costs. The fast convergence of our algorithm relies on the swift dissemination of updates on each node's clique weights to all its neighbors. In each round a node sends and receives

a number of messages equal to the size of its neighbor set. Yet, when neighbor sets contain all nodes in the network, this means that there are $|V|^2$ messages to be delivered per round. To decrease the communication costs, we proposed to use a gossiping protocol to facilitate the dissemination of updates on clique choices. We considered various implementations of such a gossiping protocol and we showed how, with carefully selected modes of gossiping-protocol operation, we can minimize the negative impact of delays in providing information on clique choices on the performance of our k-clique matching algorithm.

Our last remark on the research presented in this chapter is related to differences in the performance of our algorithms depending on the edge-weight distribution in the network. While evaluating the heuristics and the partial views we have reported their convergence times and final k-clique matchings qualities for three different edge-weight distributions: uniform, euclid-4 min and euclid-4 max. Consistently with euclid-4 max our k-clique matching algorithms achieved worst results, while the results in euclid-4 min distributions were consistently superior to those of the other two distributions. The awareness of these differences can prove very useful in predicting the effectiveness of our algorithm when its new application is considered.

CHAPTER 5

Decentralized Brand Alliance Formation

In this chapter we investigate how our k-clique matching algorithm can be applied in the realms of brand alliances formation. Brand alliance is a strategy of partnering among brands that can be used by a brand to achieve a variety of objectives. Examples include enhancing consumer attitudes toward a brand or positively influencing quality perceptions about a jointly branded product. Yet, for a brand alliance to be successful, the partners must be carefully chosen taking into account their suitability in terms of consistency of brand images and congruence of functional product attributes. We show how we can combine the theoretical models on brand and product fits, automated extraction of consumer perceptions about brands from the Web, and our k-clique matching algorithm to create a decentralized system that allows brands to discover and assess potential partners and to form tentative brand alliances.

5.1. BACKGROUND

Brand alliance is a collaboration between two or more brands created with a purpose of presenting them to consumers in some combined form which can involve various levels of integration between their products [RR94; SR98; WTP04]. A good example of a brand alliance is the partnership between Nike and Apple. The close collaboration between the two companies resulted in the announcement in 2006 of *Nike+iPod Sport kit* — a two-piece sensor-based device for runners that allows them to track their workout data, such as elapsed time, pace and distance covered as they are running. Such a brand alliance in which a single product is associated with two (or more) brands is commonly referred to as *co-*

branding [LCDL96]. Similar products of co-branding partnerships include the Senseo coffee-maker, co-branded by Philips and Douwe Egberts, the Smart car, co-branded by Mercedes and Swatch, or Dell's computers with Intel's processors inside.

A very different type of brand alliance is *product bundling* which involves selling two or more different products together for one price. Product bundling is, for example, a core business strategy of Humble Bundle, Inc., which periodically engages in partnerships with game developers (and occasionally also music creators or book authors) to create product bundles (also referred to as Humble Bundles) which are then available for purchase during a two-week period on a pay what you want basis.

The brand alliance of Nike and Apple and the brand alliances created by Humble Bundle differ in almost every aspect. First of all, the number of companies forming the partnership is different, the first alliance consists of just two companies, while in the case of Humble Bundle usually there are at least 3-5 game developers contributing to a single bundle. Secondly, the different nature of the product integration determines different levels and time spans of collaboration between alliance partners. In the case of Nike and Apple, their collaboration was very close and lasted years, which was necessary to develop a completely new product. On the other hand, Humble Bundles are created of already existing products that are not integrated in any way and the entire sales period lasts only two weeks. Lastly, the two alliances deploy different marketing channels with Nike and Apple using standard media, e.g., TV commercials and Humble Bundle relying on virality of the word-of-mouth on the Internet, e.g., encouraging customers to announce their purchase on Twitter.

Although brand alliances can come in all different shapes and flavors their common denominator is the important role of the brands that to a large extent determines the success of the alliance. For example, as Rao and Ruekert point out in [RR94; RQR99], brand names can convey information about the quality of the jointly branded product when the product's quality cannot be readily assessed by buyers prior to purchase. This makes brand alliance attractive from the point of view of a lesser known brand which can benefit from a well-known brand ally that becomes the voucher for the product's quality. For instance, in our example of Humble Bundles, a not-widely recognized independent game developer can benefit from the good reputation of Humble Bundle (and other bundle participants). On the other hand, a well-known brand can also benefit from an alliance with other well-known brands. For example, if the brand is not associated with some of the attributes of a new product, the quality of these attributes can be guaranteed by another brand [RR94]. In the case of Nike and Apple, for instance, the two companies leveraged their complementary expertise and resources, to create a

new product which is positioned at the intersection of their competencies: Nike's experience in the design of sportswear and understanding of runners' needs and behavior, and Apple's know-how of digital and software technology. So in the case of Nike and Apple we expect that their brand alliance will bring some synergistic effect where the sum is greater than the parts.

Yet, as Simonin and Ruth [SR98] point out, the consumer judgements about the brand alliance are likely to be affected not only by preexisting attitudes toward each of the brands in the partnership, but also by perceived fit of the brands and perceived fit of the products. The *product fit* refers only to the products offered by a brand alliance and is focused on the relatedness between functional attributes of these products. Thus, for example, brand alliances that combine products that are complementary with regard to their functionality, e.g., by creating a product bundle of a computer and a printer, or by including a branded microprocessor into a branded computer, have a high degree of product fit. As Simonin and Ruth show in [SR98], having a relatively high degree of product fit enhances the attitudes towards the brand alliance. Although Simonin and Ruth do not consider combining substitutes, other research (e.g., [BB99; BB00; VM09]) shows that bundling together products whose functionality or intended use is essentially identical can also be a viable strategy for a company. And as we see in the example of Humble Bundle, whose bundles consist usually of the same types of products, such as only computer games or only ebooks, brand alliances that offer product bundles consisting of substitute products can indeed be successful.

The second type of fit, *brand fit*, refers to the cohesiveness between the brand images of each partner. The definition of brand image used here was given by Keller in [Kel93]. According to this definition, a brand image is a collection of perceptions about a brand that are held in consumers' memory as brand associations. Thus, if these brand images are consistent, the degree of the brand fit is high, which should work in favor of the brand alliance. For example, as an article in Bloomberg Businessweek [HH06] mentions "Both [Nike and Apple] are iconic brands that appeal to a consumer market that is young and considers itself hip and cool", which suggests that the two brands rank high on brand fit. Conversely, one of the bundles created by Humble Bundle with THQ, which is a major game publisher, could be perceived as a poor fit between the two brands. As an article on arstechnica.com [Orl12] points out "the [Humble] Bundle built a name for itself by promoting lesser-known, quality independent games that the creators believed in — games that deserved a wider audience that no traditional set-price sales schemes or limited indie marketing budgets would allow" and "using the bundle as further promotion for already successful, big-budget games is the antithesis of 'humble,' and it dilutes the power and impact of what it means to be part of a Humble Bundle in the first place." Most of the research on different

types of brand alliances (e.g., [LKS03; LGH04]) seems to agree that the fit of the brands is determined by the similarity of their brand images. Nonetheless, Park et al. [PJS96], for example, sees the potential of combining into a brand alliance brands whose brand images are not similar but complementary.

To sum up, brand alliances can be beneficial for both well-known and little known brands, and the factors influencing their success are related to (i) the relation between the product categories, and (ii) the relation between brand images.

From our perspective it is interesting to observe how Internet-related technologies influence brand alliances formation. Chatterjee in [Cha02] acknowledges how online presence of brands and especially online retailing poses many challenges related to unpredictability of consumer demand, novel and untested business models, necessity to stay abreast with technology and to compete for consumers' attention. As Chatterjee and researchers point out ([Cha02; DBOB00; LKS03]), the network nature of the Web fosters the creation of online collaborations between brands as a way to tackle these challenges and achieve greater market share, endorse a brand's reputation or gain mutual benefit. For example, Delgado-Ballester and Hernández-Espallardo in [DBHE08], showcase how by forming an alliance with a well-known brand, a little-known online brand can mitigate the problem of the lack of established reputation and gain consumers' trust. In this context, Humble Bundle, and other similar initiatives, serve as a good example of how easy it is to create successful online brand alliances among digital goods producers. Digital goods, such as ebooks, music files, and software, are characterized by negligible marginal cost of production and usually low cost of their storage and delivery. Also bundling of digital goods, as it does not require any integration of the products, does not impose large time or monetary costs. This invites to bundling digital goods even in large bundles of hundreds of products, which can create "economies of aggregation" [BB99; BB00].

Finally, we should take a look at brand alliances from a wider perspective by zooming out from our considerations of a few companies pondering on creating a brand alliance into the world of companies interacting with each other. Such a wider perspective, has been gaining more and more attention with the ideas of *business ecosystems* [Moo93], *network economy* [Kel99] or *business webs* [TTL00] which draw attention to the structural organization of enterprises and markets, and the changes such organizations are undergoing. There is an irrevocable shift of focus from a single self-sufficient company or traditional value chains to the networks of interconnected companies. Looking into the future, van Heck et al. [HV07] carve a scenario in which such networks become more agile. They foresee that companies operate in what they call a *pick, plug, and play* fashion, dynamically creating linkages with each other to provide complex, bundled goods and services. Digital technology plays a pivotal role in facilitating this

constant re-organization of business connections.

5.2. SYSTEM MODEL

The presented research shows brand alliances as an appealing strategy that can benefit brands in traditional as well as online settings. At the same time, the importance of selecting suitable partners by assessing brand and product fit is emphasized. Ironically, the advances in ICT and Internet have made the discovery of best possible partners more difficult. The current communication technology makes it potentially possible to team up with companies from all over the world. However, at the same time this means that the choice is much greater, which in many cases makes the decision process much more complex. Moreover, the Internet also contributes to the increasingly rapid growth of consumer expectations, which forces brands to react quickly to a changing situation. By collaborating together brands can achieve results faster and, thus, gain the competitive advantage over competitors. Yet, to make such a collaboration effective, the process of alliance creation cannot be too time-consuming, as the competitors may seize the opportunity sooner.

To aid brands with their search for brand alliance partners, we envision a system in which a large collection of brands looking for brand alliance partners interacts with each other. Such a system can be seen as a form of co-opetition [NB97] in which system participants who in the real world are to a greater or lesser extent competitors cooperate with each other. The system is organized as a network of nodes, each representing a profile of one brand. The nodes exchange among each other information about brands present in the system and form prospective alliances. The operation of the system is fully decentralized, the decisions on which nodes to partner with are made by nodes locally based on the nodes' knowledge about the state of the system. The fact of system's decentralization alleviates many problems related to the existence of centralized components. For example, a concentration of decision making in the hands of a single actor can lead to arbitrarily large costs of using the system or possible bias in decisions made by the controlling party.

This leads us to the following outline of the requirements for a *partnership formation service*:

- **Clarity** The rules by which the potential partners are selected should be clear and not too complicated.
- **Fairness** The service, while matching prospective partners, should respect the interests of all the companies that are represented by nodes in the system.

- **Minimal Resource Requirements** Use of the service should not pose high computational or memory requirements on the nodes.
- **Ease of Deployment** Companies should be able to add their nodes to the system at any time without using complicated bootstrapping mechanisms.
- **Robustness** The impact of nodes entering or leaving the system should be minimal. Moreover, the service should be able to recover gracefully from the situations of many nodes leaving the service at the same time.
- **Scalability** The performance of the service should not be heavily affected by a growing number of nodes.
- **Automation** The whole process should require (almost) no manual intervention; this means that profile creation for participants as well as the discovery of potential partners and formation of prospective partnerships should be fully automated.
- **Flexibility** It is possible to look for partnerships based on various criteria.

We will address most of these requirements and to the others we will provide a guidance for a plausible implementation.

5.2.1. Node Profile Model

Each brand that is looking for brand alliance partners is represented in our system by a single node. The nodes are then responsible for discovering potential partners, assessing their suitability and trying to form the most promising partnership. In order to realize these tasks, each node needs to have information about the brand it is representing, which it can share with other nodes and which it can use to easily assess the potential of partnering with any other node. As we have mentioned before, the important factors in the brand alliance partners assessment are brand fit, related to the consistency of brand images, and product fit, related to the compatibility of functional product-related attributes. Thus, the profile of a node should consist of two parts representing respectively brand image and functional image of a given brand. We propose to build such a node profile on the model[1] presented by Vermeulen in [Ver07].

The brand image puts the brand in the context of symbolic attributes. After Vermeulen we compose the list of symbolic attributes out of the main brand personality traits, which have been selected by Aaker [Aak97]. These personality traits are organized into groups of four creating five dimensions of brand personality. The exact list of traits used in our simulations is presented in Table 5.1. For each brand, the brand image profile is a vector of the individual measurements of the strengths with which the brand name is associated with any of these traits.

[1] Our profile adopts only a part of Vermeulen's model. In particular, we omit evaluative terms (e.g., "good", "bad") and Osgood's semantic differential terms (e.g., "active", "passive").

Sincerity	Excitement	Competence	Sophistication	Ruggedness
domestic	daring	reliable	glamorous	tough
honest	spirited	responsible	pretentious	strong
genuine	imaginative	dependable	charming	outdoorsy
cheerful	up-to-date	efficient	romantic	rugged

Table 5.1: Brand personality traits.

The functional image in a node's profile should help in assessing the product fit with other brands. As Vermeulen points out, creating a list of functional attributes which would be relevant to all possible product categories poses a challenge, especially because only a small part of the terms in such a list would be relevant to a single brand. Instead, he proposes to use associations of a given brand name with other brand names, which has been proved to work well in determining the level of competition between brands. As shown in [VB06], strong association between two brand names (extracted from the Web) indicates strong competition. Thus, the full functional image can be encoded as a vector of values corresponding to the strength of associations between a given brand and other brands in the system. The advantage of such an approach is that it is very generic and can be applied to a situation where all the brands fall into the same category as well as to a situation with a large variety of product categories. The disadvantage of this approach is that the size of a functional image can still be relatively large, as it grows linearly with the number of brands present in the system.

The important point to make here is that an entire node profile can be constructed in a fully automated way by retrieving relevant data from the Web. Traditionally, measuring brand images or assessing brand and product fit involves surveying a large group of consumers. Unfortunately, such an approach is impractical due to its long time and monetary costs. Vermeulen shows in [Ver07] how both brand image and competitive image, which are further used to assess brand and product fit, can be extracted from the World Wide Web. We can see how the Web, especially since the advent of Web 2.0, has become an excellent source of consumer opinions voiced on various forums, discussion groups, blogs, or websites of online retailers. Vermeulen proposes thus to retrieve the information about the strength of consumers associations between brand names and relevant terms and to construct out of them brand and competitive images. His method is based on the techniques used also in computational linguistics which relate the degree of association between two notions to their frequency of co-occurrence in text corpora [CH90].

Furthermore, instead of crawling the Web, Vermeulen suggests using an existing Web search engine to elicit frequency of terms co-occurrence on the Web.

The advantage of using a Web search engine is that it saves on the computational effort, as Web search engines maintain information about Web documents in already indexed form which allows for easy search of relevant Web documents. Moreover, the major search engines apart from providing a list of relevant documents for given search terms, also provide an estimated number of matching pages, commonly referred to as hit count. Based on the hit counts of: (i) first term, t_1, (ii) second term, t_2, and (iii) t_1 with t_2 together, the degree of the two terms' co-occurrence can be computed, for example, by using the Jaccard association coefficient:

$$J(t_1,t_2) = \frac{|t_1 \text{ and } t_2|}{|t_1 \text{ or } t_2|} = \frac{\text{hit\_count}(t_1 \text{ and } t_2)}{\text{hit\_count}(t_1) + \text{hit\_count}(t_2) - \text{hit\_count}(t_1 \text{ and } t_2)}$$

which measures similarity between two sets, in our case sets containing documents with term t_1 or term t_2.

Using Web search engines has its drawbacks. Firstly, Web search engines do not index all documents on the Web. Secondly, the hit counts provided usually do not express the exact number of pages that contain a given term but only an estimation [Uya09]. Finally, it remains an open question how the brand associations extracted from the Web stand in relation to the brand associations that real consumers hold. Nonetheless, we find this approach fully sufficient for our purposes which are more exploratory than prescriptive, and where use of a Web search engine to automatically create node profiles constitutes only a part of our proof-of-concept solution.

5.2.2. Co-Branding Potential Functions

By comparing their brand images, or functional images, any two nodes can assess their mutual brand, or product, fit. To this end nodes can interpret their images as association networks and measure their structural equivalence (*SE*) to find the degree of similarity between them. The function usually used for this purpose is the Pearson product-moment correlation coefficient whose values range from -1 (meaning two images are complete opposites) to 1 (meaning two images are identical). As already mentioned, similar brand images are advantageous for brand alliance partners. Thus, the higher the value of the coefficient, *SE*, for two brand images, the more promising the pairing of two brands seems.

On the other hand, the overlap between the functional attributes should be moderate, thus favoring the moderate values of the coefficient for functional images. To account for that, Vermeulen proposes to consider that two brands have a high product fit if *SE* of their functional images is close to the average *SE* of product images in the entire set of brand alliances considered. Accordingly, if *SE*'s value of functional images is far from the average *SE*, the two brands are

considered a poor fit in terms of functional compatibility. Vermeulen formalizes this relation as a function called structural complementarity (*SC*).

By averaging *SE* of brand images and *SC* of functional images, the *co-brand potential* of two brands is computed. This *co-brand potential* can play a role of an edge weight for our *k*-clique matching protocol. Furthermore, the co-brand potential of the larger group (clique) of brands can be computed by, for example, averaging the pairwise co-brand potentials. Yet, the *co-brand potential* as proposed by Vermeulen has some drawbacks which are related to the use of the *SC* function. *SC* computation depends on the average value of *SE*. As a consequence, a given brand *A* is not able to assess which of any two brands, *B* or *C*, is a better partner in terms of functional fit in isolation from the entire set of brands. Moreover, if the set of brands changes, the average value of functional images *SE*'s may also change leading to a situation in which a brand *A*'s preference of, say, brand *B* over brand *C* may become reversed. Therefore, in our simulations we have decided to use only brand fit as a measure of co-branding potential. Another solution, not presented in this dissertation, would be to follow the other line of research presented earlier that suggests that bundling substitutes can also be beneficial. This would mean that instead of using *SC*, we could rely solely on *SE* for functional images. Yet, devising a *co-brand potential* function falls out of scope of our research.

5.2.3. Protocol's Layered Architecture

To facilitate the discovery of suitable partners and the formation of promising alliances, we propose a protocol framework that draws on the main ideas from the previous chapter. The framework consists of four protocols. The main functionality related to evaluation and formation of alliances is provided by a version of our *k*-clique matching protocol. To support the operation of this protocol three other protocols run in parallel to it. Two of them are responsible for the discovery of suitable partners. The other protocol disseminates the updated weights of cliques chosen by nodes. The information flow between these four protocols is depicted in Figure 5.1.

The version of the *k*-clique matching protocol used in our framework follows the implementation detailed in Figure 5.2. The general mode of operation of this protocol is identical to the one of our *k*-clique matching algorithm adapted for the use of heuristics (see Figure 4.1), which is well suited for situations in which a node has knowledge only about a fraction of the system participants at a time. In each round, the protocol reevaluates whether the clique chosen in the previous round is still *proper*, i.e., whether it is still a viable choice (lines 2–4). Then, the clique offers, which were sent to inform the node that it is a member of another node's clique, are examined to check if any of these cliques is a better (more

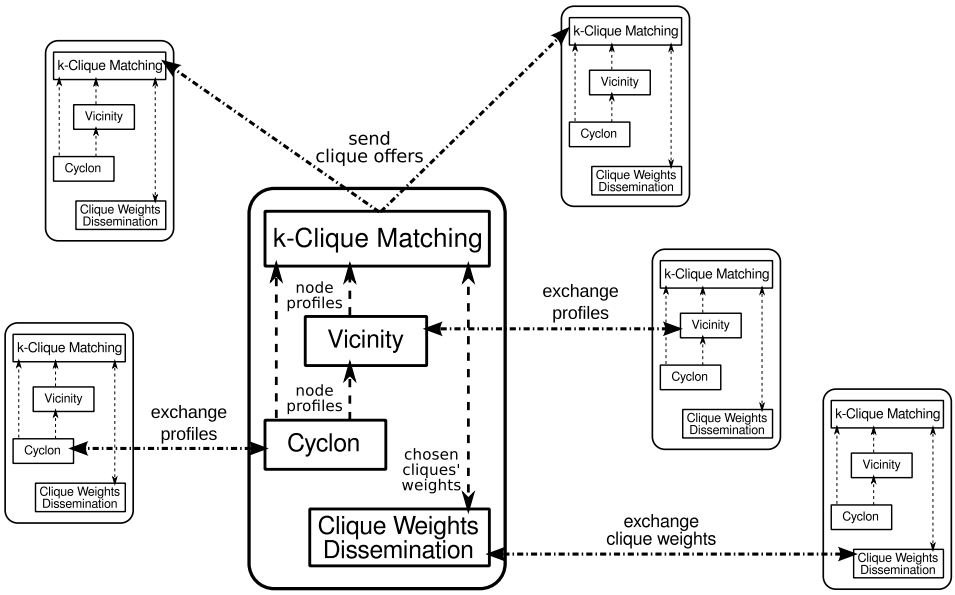


Figure 5.1: Flow of information between protocols facilitating formation of brand alliances.

attractive) choice (lines 5–8). Subsequently, the protocol executes the MOSTATTRACTIVECLIQUEINVIEW function. The function retrieves the list of all node profiles stored in the views of *partner discovery protocols* and supplements it with profiles of the most attractive clique found so far in this round. Then, from the elements of this list all possible combinations of $k-1$ nodes are generated and the most attractive one among them is returned as the function's result. As a consequence, the most attractive k-clique considered during the given round is set as the current choice (lines 10–12). The chosen clique's weight is provided to the protocol responsible for clique weights dissemination to all of v's neighbors (line 13). Furthermore, the node additionally sends to the members of the chosen clique a clique offer (lines 14–15) which contains the information about that clique's members (including their profiles).

Partner-discovery protocols are implemented as two gossiping protocols, VICINITY and CYCLON, stacked one upon the other. The implementation details for these two protocols are summarized in the previous chapter in Figure 4.19. Both protocols are responsible for supplying neighbor profiles to the k-clique matching protocol. CYCLON provides a random selection of neighbors, while VICINITY provides neighbors with highest co-brand potentials.

To check whether a clique is *proper* or estimate a clique's *attractiveness*, the k-clique matching protocol needs up-to-date information from other members of

MAIN PROTOCOL: **Active thread:**

```
1:  loop
2:    if not proper(v, C_v) then
3:      C_v ← {}
4:      w_v ← −∞
5:    for all (u, C_u) ∈ received offers do
6:      if attr_v(C_u + {u} − {v}) > attr_v(C_v) then
7:        C_v ← C_u + {u} − {v}
8:        w_v ← w({v} + C_v)
9:    C ←MOSTATTRACTIVECLIQUEINVIEW(k, v, C_v)
10:   if attr_v(C) > attr_v(C_v) then
11:     C_v ← C
12:     w_v ← w({v} + C_v)
13:   send w_v to all u ∈ N(v)
14:   offer ← (v, C_v)
15:   send offer to all u ∈ C_v
```

Passive thread:

```
1:  loop
2:    receive w_u (or an offer) from any u ∈ N(v)
3:    store w_u (or an offer) locally
```

MOSTATTRACTIVECLIQUEINVIEW$(k, v, \mathbf{C}_v)$:

```
N ← v's view of VICINITY + v's view of CYCLON + members of C_v
C ← {}
for all U ≡ {u_1, ..., u_{k−1}} ⊆ S do
  if attr_v(U) > attr_v(C) then
    C ← U
return C
```

Figure 5.2: k-Clique matching protocol facilitating brand alliances formation.

this clique on the weights of their chosen cliques. This information is provided by a protocol responsible for the clique weights dissemination in the system. In Section 4.5 of the previous chapter, we investigated various implementations of gossiping protocols for this purpose, and the most effective of them is used here in our framework. For this protocol there is no limit on the size of the view, as it is impossible to predict which nodes will be evaluated by the k-clique matching protocol as potential clique partners in any given round. Each item of the view consists of a node's id, contact information, chosen clique weight and the age of this item. In each round, the ages of all items are increased and a random node

from the view is selected. The two nodes exchange b items: one item contains the info on the sender's own clique weight with age 0 and the remaining $b-1$ items are the youngest items from the sender's view. Upon receipt of the items, each of the two nodes updates the information about the nodes related to each of these items keeping the clique weight with the smaller age. Such gossiping exchanges are sufficient for the clique weight updates to quickly spread across the entire network. Nonetheless, in our framework there exists another source of clique weight updates, which can be exploited by the protocol. Fresh information on chosen cliques comes also from the clique offers sent by the k-clique matching protocol running at other nodes. This information is passed to the clique-weight dissemination protocol and used to update its view.

The above framework can be viewed as a part of a larger system in which many instances of the framework are executed separately. Each such instance can have its own co-brand potential function. Thus, in different instances different partners will be seen as more suitable, and as a result, in different instances different cliques will be created. Any node in the system can initiate such an instance with a co-brand potential of the node's personal preference, and any other node can decide whether it wants to participate in this instance. Many different instances can run simultaneously, each having a distinctive id. If by chance any two such instances have identical co-brand potential functions they can be combined into one by adopting the smaller id. To spread the invitations to participate in various instances, a simple gossiping protocol can be run by all the nodes.

Such an approach makes the system highly flexible. Nodes can make individual decisions about which instances they want to participate in, and if no instance seems satisfactory, they can create their own with a co-brand potential function suited for their individual needs. As a result, nodes can discover cliques best suited for their various purposes.

5.3. EVALUATION

In this section, we take a look at the performance of our proposed framework. The simulations presented in this section have been carried out using PeerSim, an open source simulator for P2P protocols [MJ09].

To make our simulations relatable to the real world, we used real-world data to construct node profiles. Moreover, as the number of partners in a single brand alliance is usually small (with brand alliance of two partners being the most common), we limited ourselves to creating 2- and 3-cliques.

5.3.1. Experimental Data Acquisition

In his initial research Vermeulen used 50 brand names from the Businessweek's list of top brands of 2005 and 30 brand image attributes. The estimates on the number of (co-)occurrences of these terms on the Web were acquired by sending queries to the Google Web Search API. We have redone the data acquisition in a similar fashion. We combined lists of Fortune's top 1000 USA companies [For10a] and top 500 global companies [For10b] for 2010, which yielded 1366 company names in total, and we used 20 brand personality traits listed in Table 5.1. To retrieve the estimated number of co-occurrences of these terms, we performed our queries using the University Research Program for Google Search. This Program provided academic researchers with high-volume programmatic access to Google Search results and was officially shut down on January 15, 2012. All our data was gathered between October 13, 2011 and January 7, 2012. The total of 961001 queries were sent:

- 1366 queries with a company name as a search term (if the name consisted of more than one word it was taken into quotation marks),
- 20 queries with a brand personality trait,
- 1366*20 queries with a pair of company name and a brand personality trait as a search term,
- (1366*1365)/2 queries with a pair of two company names (acquired in order to compute product fit values, but not used in here as explained in Section 5.2.2).

From the replies, we saved only the values of the total number of hits.

5.3.2. Node Profile

We used the collected data to construct node profiles. Each profile consists only of a brand image, which is stored as a vector of 20 values with each of the values representing the strength of relation between node's brand and a particular trait. The strengths of these relations are computed using the Jaccard association coefficient. A node evaluates each of its peers by calculating the structural equivalence, *SE*, between their respective brand images using Pearson product-moment correlation coefficient. Further, to assess the prospective value of the brand alliance of size 3 (or more), the node calculates an algorithmic average of respective pairwise *SE* values.

5.3.3. Protocols Comparison Setup

We created our framework by choosing as the starting point our basic k-clique matching algorithm from Chapter 2. We modified it by incorporating partial views

		view size	# items per message	avg. # items sent per round	# item available to k-clique matching algorithm
FV		1365	——	——	1365
PV	VICINITY	20	10	$2 \cdot 10$	20+20
	CYCLON	20	10	$2 \cdot 10$	

Table 5.2: Parameter settings of *FV* and *PV*.

	view size	# items per message	avg. # items sent per round	# item available to k-clique matching algorithm
br	1365	1	$1365 \cdot 1$	1365
gs	$40^* \rightarrow 1365$	20	$2 \cdot 20$	$40^* \rightarrow 1365$

*40 is the initial size of *gs* cache.

Table 5.3: Parameter settings of *br* and *gs*.

and clique-weight gossiping approaches from Chapter 4. Thus, it seems justified to evaluate our framework's performance against the performance of the protocols, which became its constituent building blocks. To this end, we use three other protocol configurations.

The choice of these configurations has been dictated by the distinct types of information that need to be provided to our k-clique matching algorithm. There are two types of such information: (i) information on the profiles of other nodes in the system, and (ii) information on the current weights of other nodes' clique choices. For each of these two types of information we consider here two possible sources.

For the profile information the first option is to assume that each node has an upfront knowledge about profiles of all nodes in the network and that in each round the k-clique matching algorithm has access to this information in its entirety, hence each node has a full view (*FV*) on other nodes' profiles. Another source of information provides to the k-clique matching algorithm only a limited number of node profiles present in the entire network. This partial view (*PV*) is composed out of VICINITY and CYCLON views and can change from round to round.

For the clique weight information, the possible sources are provided either via broadcasting (*br*) or gossiping (*gs*) protocols. Regardless of the communication method, both protocols maintain a cache with the information about the clique weights of all the nodes in the network. In the case of gossiping, this cache is

not full at the beginning of a simulation but rather is built up gradually as the gossiping algorithm discovers new nodes.

The two profile information sources with the two clique weight information sources give rise to four distinct combinations of algorithms:

- *FV+br* creates our basic *k*-clique matching algorithm as described in Chapter 2,
- *FV+gs* produces an algorithm from Chapter 4.5,
- *PV+br* is an algorithm very similar to algorithms from Chapter 4.4, but now both CYCLON and VICINITY views are used by *k*-clique matching algorithm
- *PV+gs* is our framework for brand alliance formation.

In order to simplify the analysis of the performance of the four configurations, we impose an equal round length for all component protocols. Thus, once every T time units each simulated protocol is executed by each node exactly once. For example, for our framework *PV+gs* during every T time units each node initiates one gossip exchange of clique weights (*gs*), one gossip exchange of CYCLON and one of VICINITY (*PV*), and one execution of *k*-clique matching algorithm.

The details of parameter settings as used in our simulations for each of four information sources are summarized in Tables 5.2 and 5.3.

5.3.4. Convergence Analysis

Similarly to previous chapters, we examine the performance of our framework in terms of the convergence speed. Apart from measuring convergence in the classic way as the number of rounds, we look also at the convergence from the point of computational load and of the communicational load.

Before we proceed with an analysis of individual convergence measurements, observe that both for $k = 2$ (see Figure 5.3) and $k = 3$ (see Figure 5.4) the dependencies between the results of the four selected configurations follow the same pattern, with only difference being the length of the simulations. Therefore for the simplicity, in our analysis we focus solely on results for $k = 3$.

In terms of rounds, configuration *FV+br* is the fastest (see Figures 5.4a and 5.4b). For this configuration it takes nodes on average only 20 rounds to reach a stable state. Exchanging either *FV* or *br* to a different protocol results in the new configuration, *FV+gs* or *PV+br*, to be about two magnitudes slower than *FV+br*. Additionally, *PV+br* takes almost twice as long to converge as *FV+gs*, although the dynamics of the two configurations at the beginning of the simulation would suggest quite the opposite. When looking closer at the graph in Figure 5.4a, we discover that with *PV+br* stable cliques form very quickly in the initial stages of the simulations and within 20 rounds over 84% of nodes manage to find cliques in which they will remain till stabilization of entire systems (we refer to such cliques

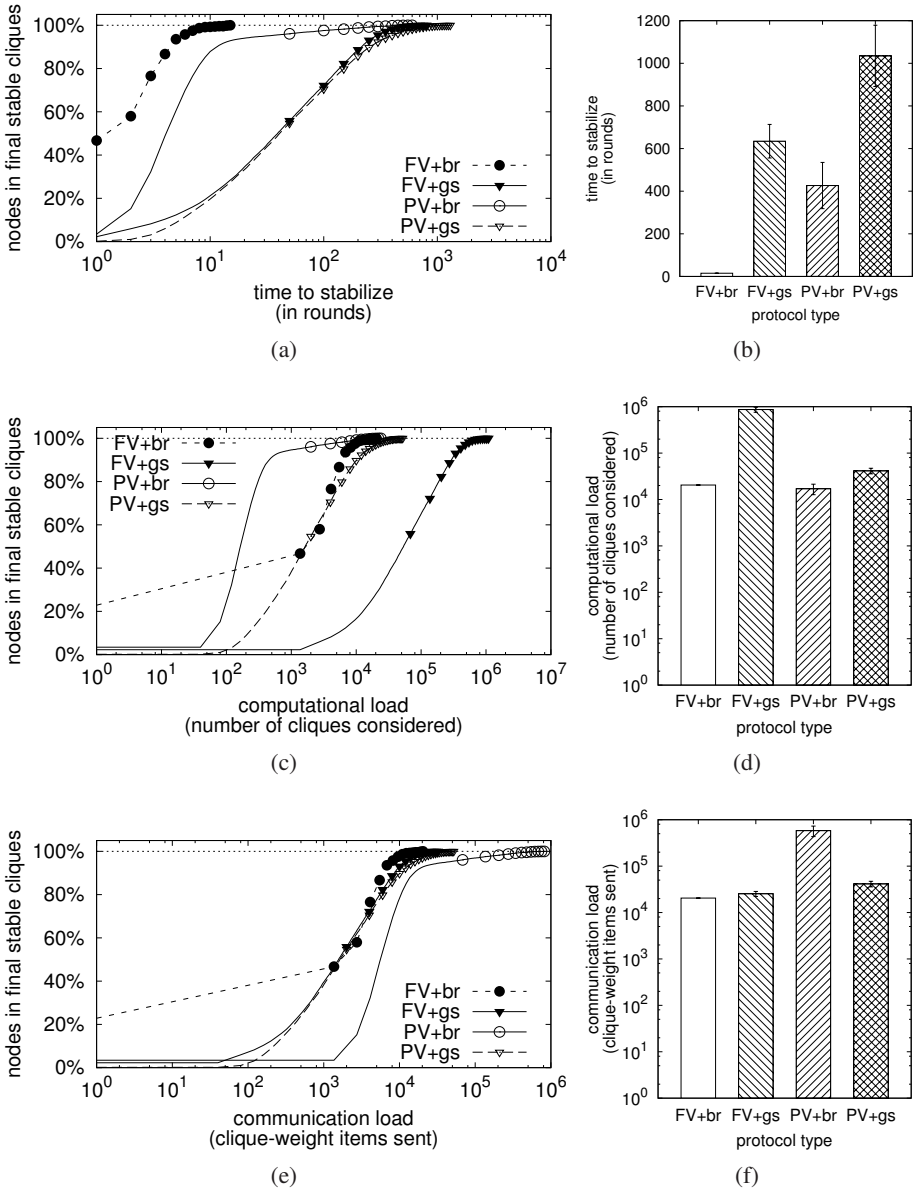


Figure 5.3: Performance of k-clique matching protocols framework (PV+g) for brand alliance formation in comparison with three other versions of the k-clique matching protocol; $k = 2$.

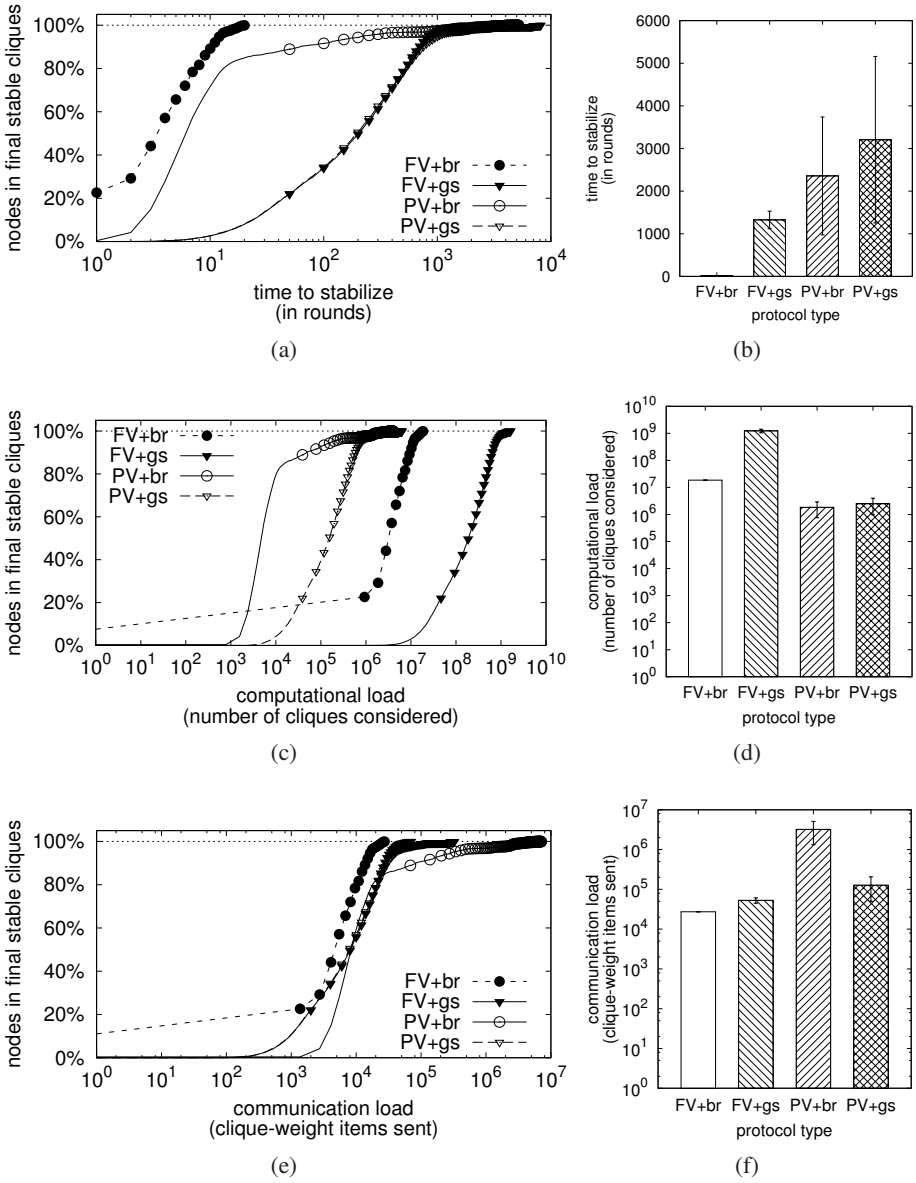


Figure 5.4: Performance of k-clique matching protocols framework (PV+g) for brand alliance formation in comparison with three other versions of the k-clique matching protocol; $k = 3$.

as final). Yet, as the simulation progresses the tempo of final cliques formation significantly decreases. On the other hand, with configuration *FV+gs* the number of final cliques increases more gradually, and it takes more than 500 rounds to form final cliques by 80% nodes. This shows how significant is the impact of up-to-date information of node's chosen clique weights in the initial stages of the *k*-clique matching protocol. Our framework's configuration *PV+gs* follows closely the dynamics of *FV+gs*, but converges much slower. The total time of convergence for *PV+gs* is 1.4 times longer than for and *PV+br* 2.4 times longer than for *FV+gs*, making it the slowest of the four protocols.

Yet, if instead of measuring convergence in terms of elapsed rounds, we measure it in terms of total computational load incurred by our *k*-clique matching algorithm, a very different picture of the four configurations arises. As can be seen in Figure 5.4d (note the logarithmic scale of the Y-axis), *FV+br* is no longer the fastest configuration anymore. Instead, the slowest configurations in terms of rounds, *PV+br* and *PV+gs*, turn out to be most effective in terms of computational resources. This suggests that the *PV* layer which combines CYCLON and VICINITY works well when in comes to providing the most promising subsets of neighbors to *k*-clique matching protocol. As a result, although the two configurations need more rounds to converge, thanks to the fact that in each such a round node examines only a small fraction of all possible cliques, puts these configurations ahead of *FV+br* in terms of total computational load imposed on *k*-clique matching algorithm.

In our last comparison of the four configurations, we take a closer look at the relation between clique weight dissemination method, *br* or *gs* and the convergence of our *k*-clique matching protocol (see Figure 5.4f and note the logarithmic scale of the Y-axis). When clique weights are broadcasted, in each round the number of messages with clique weights in the system amounts to the square of the network size with each node receiving $|V| - 1$. In comparison, when gossiping is used to disseminate clique weights, each node engages on average in two gossiping exchanges sending 20 items and receiving 20 items in each exchange. Due to this discrepancy, the difference between *FV+br* and *FV+gs* diminishes, when the total communicational load of clique-weight gossiping is compared instead of total number of rounds. Moreover, for *PV+br* and *PV+gs*, the latter is more efficient in terms of clique-weight communication necessary for system's convergence.

To summarize, the presented results show that the framework we proposed for brand alliance formation, *PV+gs*, can be a viable contender. It performs worse in terms of rounds to its predecessors *FV+gs* and *PV+br*, but when taking into account computational and communicational loads, it manages these two resources much more effectively than *FV+gs* or *PV+br*.

We will now zoom into more detail of our framework's configuration and

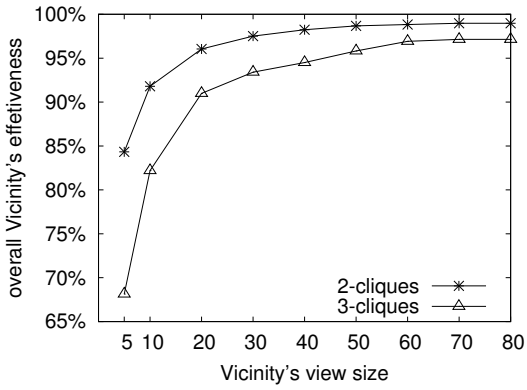


Figure 5.5: Effectiveness of the Vicinity layer from the point of view of the final k-clique matching for $k = 2$ and $k = 3$.

influence of its component protocols on the performance of our framework. In particular, we examine in more depth the effectiveness of VICINITY as a source of neighbor profiles, and the impact of clique-weight gossiping on the tempo of frameworks convergence.

5.3.5. Partial Views Performance

In Chapter 4.4, we analyzed for the first time how two gossiping protocols, CYCLON and VICINITY, can be used as a source of node profiles for our k-clique matching algorithm. We considered two cases, (i) running CYCLON, (ii) running both CYCLON and VICINITY and providing to our algorithm node profiles only from VICINITY. In the second case, the network converged fast but some nodes remained without cliques as their prospective clique partners were not included in VICINITY's views. In the first case, all nodes eventually formed cliques, but the convergence took considerably longer. In this chapter for a change, the k-clique matching algorithm is provided with the nodes from both CYCLON and VICINITY. As a result, the benefits of the two protocols get combined. VICINITY allows the majority of nodes to quickly find their final cliques, while the less fortunate remaining nodes can rely on CYCLON to discover their final cliques.

In Chapter 4.4 we have examined the impact of edge-weight distributions on the efficiency with which VICINITY can act as a source of most promising nodes. Now we take a closer look how this efficiency is influenced by the size of the VICINITY view. To measure the contribution of the VICINITY view in the creation of the final k-clique matching matching, M_{final}, we take each clique from M_{final} and check if there exists a node in this clique such that all its partners from the clique belong to its VICINITY view. The percentage of the network covered by

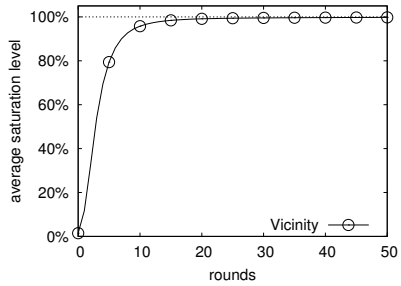


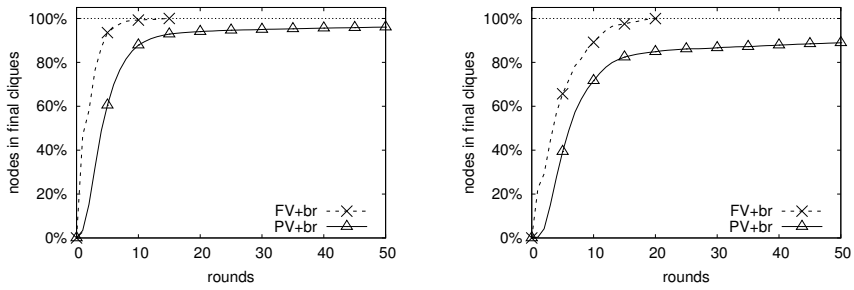
Figure 5.6: Convergence of VICINITY running on top of CYCLON.

such cliques is a good indication of VICINITY *effectiveness* in supplying promising node profiles to the *k*-clique matching algorithm. In Figure 5.5 we depict such defined VICINITY effectiveness across various view sizes separately for creation of 2- and 3-cliques. As expected, by increasing the size of VICINITY's view, we increase the chances for each node that all its partners from the final clique will fall into its view. For both clique sizes we observe that the VICINITY effectiveness rises quickly as its view size increases from 5 to 20, reaching 96% for 2-cliques and 91% for 3-cliques. As the view size further increases the growth rate of effectiveness slows down but it will keep growing monotonically until it reaches 100%.

Already for the view size of 20, which we have chosen for our simulations, VICINITY helps over 90% of nodes to create their final cliques. Moreover, the two-layered architecture of CYCLON and VICINITY allows for the fast convergence of the latter. In Figure 5.6 we see that also within the first 20 rounds an average VICINITY view becomes filled with 99% of the 20 most suitable nodes. This in turn contributes to the high speed with which the final cliques form. Figures 5.7a and 5.7a depict the initial rounds of simulation for *FV+br* and *PV+br*. We observe that switching from a full view of the network to the partial view leaves the majority of the nodes only slightly affected. Over 80% of the nodes still manage to find their final 3-cliques during the first 20 rounds, and for 2-cliques this amount rises to 90%.

5.3.6. Clique-Weight Distribution Performance

As we have already mentioned while discussing convergence of the four configurations, up-to-date information on the weights of cliques chosen by the nodes has a significant impact on the tempo of final cliques formation. When comparing in Figure 5.4a convergence of *FV+br* against *FV+gs* and *PV+br* against *PV+gs*, we observed that the tempo of final cliques formation is much smaller



(a) 2-cliques (b) 3-cliques

Figure 5.7: Performance of four configurations in the initial rounds of simulation.

for configurations using gossiping of clique weights (*gs*) than for their counterpart configurations broadcasting clique weights (*br*). It took *FV+br* only 8 rounds on average to form final cliques by at least 80% of nodes, while it took *FV+gs* over 520 rounds to do the same. Similarly, it took *PV+br* only 14 rounds on average to form final cliques by at least 80% of nodes (only 6 more than for *FV+br*), and it took *PV+gs* again over 520 rounds to do the same.

One of the methods to improve the formation of final cliques for configurations using *gs* is to increase the amount of information on nodes' clique weights by either increasing the number of exchanges relative to k-clique matching executions or the sizes of exchanged messages. We have examined both of these techniques in Chapter 4.5. Both of these techniques would improve the freshness on clique weight information and, consequently, shorten the convergence of the k-clique matching algorithm in the *FV+gs* or *PV+gs* configurations, yet at the same time both of these approaches would increase the bandwidth consumption of the *gs* protocol.

Now, we investigate if we can improve the tempo of final cliques formation not by increasing the volume of exchanged information but by improving the relevance of exchanged clique weight information. In particular, we will focus our efforts on improving our proposed brand alliance formation framework, *PV+gs*. We know that for configurations with partial views, the nodes present in *PV*'s view are the ones that will be considered as potential clique partners by our k-clique matching algorithm. Thus, the up-to-date information on these nodes' chosen clique weights would be especially useful. By coordinating partial views with clique weight gossiping in such a way that nodes present in *PV*'s view are also the nodes with most recent information on their chosen clique weights, a node can improve its own decision on clique partners, leading to a faster convergence of entire system.

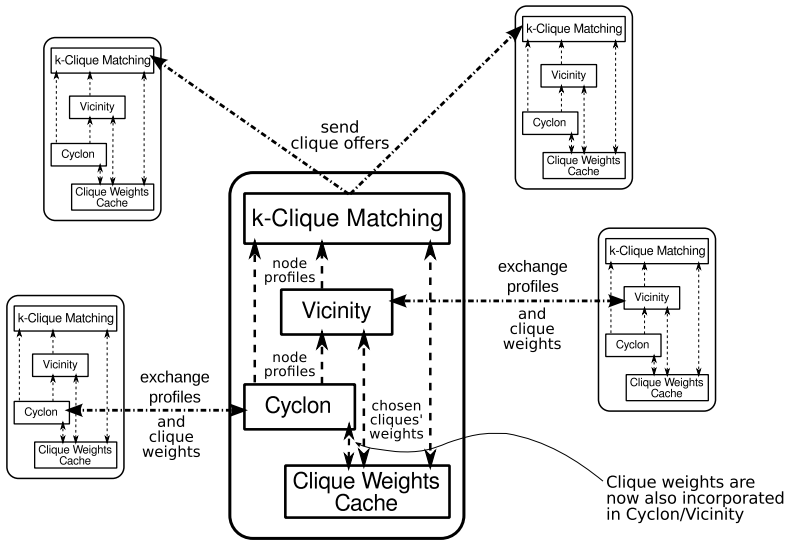
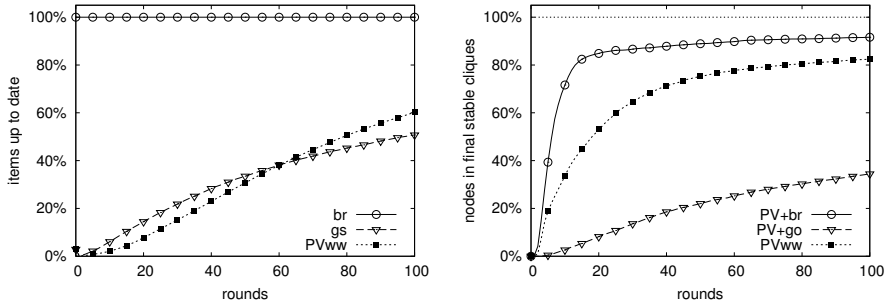


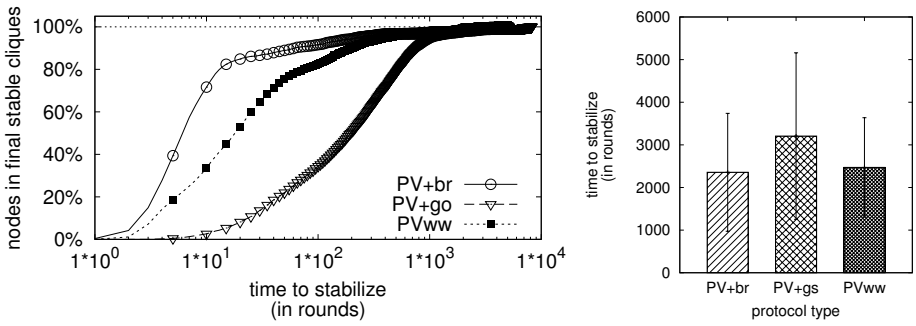
Figure 5.8: Improved framework for brand alliances formation.



(a) state of clique weights cache (b) formation of *k*-cliques

Figure 5.9: Performance of *PVww* in comparison to *PV+br* and *PV+gs*.

The coordination of partial views and clique weight gossiping can be implemented as follows. We can put the responsibility of exchanging node profiles and their clique weights entirely on *PV* protocols. CYCLON and VICINITY will operate in most part as usual. Yet, before sending items to the selected node they will append each of the items with the clique weight of the node associated with this item accompanied by the age of this clique weight information. Then, after exchange, the node will update its clique weight cache retaining the clique weight with the smallest age for any node present in received items. A scheme of information flow in this modified framework is depicted in Figure 5.8 and we dub the new framework *PVww* (*PV* with weights).



(a) (b)

Figure 5.10: Performance of *PVww* in comparison to *PV+br* and *PV+gs*.

In Figures 5.9 and 5.10 we compare the performance of *PVww* with the performance of *PV+br* and *PV+gs*. Figures 5.9 depicts the first 100 rounds of simulations. From 5.9b we see that *PVww* achieves much better performance than *PV+gs* with regard to the tempo of final cliques formation. It took *PVww* on average 80 rounds to form final cliques by at least 80% of nodes. In comparison, it took *FV+br* only 14 (5.7 times less) rounds on average to do the same, but it also required 34 times more items with clique weight information to be delivered per round. The performance of *PVww* is a large improvement over *PV+gs* which, with the same communication load, needed on average over 520 rounds to reach 80% of nodes matched in final cliques.

When we measure for each cache what percentage of nodes in the network have a corresponding item in this cache with an up-to-date weight, we see (as depicted in Figure 5.9a) that on average in the initial rounds *PVww* does not provide nodes with more up-to-date information than *PV+gs*. Thus, the improved performance of *PVww* does not stem from the more accurate information on clique weights, but rather from the preference in updating weights for the nodes that comprise *PV*'s view.

The performance of *PVww* in the initial rounds, translates to its overall performance. Figure 5.9a presents the tempo of final cliques creation throughout the entire simulation, showing that *PVww* stays ahead of *PV+gs* for the entire duration of the simulation. Finally, in Figure 5.9a we see that *PVww* takes only slightly longer to converge than *PV+br*, thus our new way of disseminating clique weights seems almost as good as broadcasting and at the same time is much more bandwidth efficient.

5.4. CONCLUSIONS

In this chapter we demonstrated the potential of using our k-clique matching algorithm in a real-life application. Our application of choice was brand alliance formation, on a large scale. Instead of looking at the problem of finding best partners for an alliance from the point of view of an individual brand, we looked at a pool of brands as a system, in which every brand is evaluating distinct characteristics of other players seeking for best-suited partners.

Our algorithm seems well suited for the task. It is grounded on the premise that nodes, or in this particular case, brands are mainly concerned with their own goals and prefer to make their decisions individually instead of yielding the control over to a third party. Additionally, our algorithm allows for a creation of cliques (brand alliances) that are mutually disjoint, thus avoiding potential conflicts of interests. On the other hand, the available research on brands and brand alliances, provided us with the methods for quantifying the fitness of any pair of brands which nodes could use to compute weights of any clique and pick the best one of those available.

We introduced an initial framework, *PV+gs*, for decentralized brand alliance formation, which combined the research presented in the previous chapters of this dissertation. We investigated the system's behavior through experimentation, analyzing its strong and weak points. Our analysis led us to modification of an underperforming clique-weight dissemination layer, and proposing a new framework, *PVww*. We showed that this improved framework forms prospective brand alliances fast, in a totally decentralized and self-organized manner.

CHAPTER 6

Conclusions

In this dissertation we explored how the problem of forming small-size partnerships in a large pool of nodes can be solved in a fully decentralized fashion. The approach we adopted for this research was to focus first on exploring theoretical properties of our proposed solution and next to look more closely at practical issues associated with real-life deployment of our algorithms. We dedicate this last chapter to describing our accomplishments and discussing the challenges encountered and lessons learned along the way. We conclude by drafting directions for future research.

6.1. DISCUSSION

In Chapters 2 and 3, we formalized the problem of small-size partnerships formation as a weighted k-clique matching problem, where the objective is to find in a given graph $G = (V, E)$ a set of non-overlapping k-cliques such that the total weight of the cliques is maximized. We consider also three generalizations of this problem: (i) weighted K-clique matching, in which the size of cliques, k, can be any of the values present in set K, (ii) weighted k-clique b-matching, where the condition that each node can belong to at most one clique is relaxed to at most b cliques per node, (iii) weighted K-clique b-matching which combines modifications introduced by (i) and (ii).

To find a decentralized solution for these problems we sifted through existing distributed algorithms that solve similar problems. Our attention was caught by a self-stabilizing algorithm that finds a 1/2-approximation of the weighted matching problem — a special case of the weighted k-clique matching problem for $k = 2$ — proposed by Manne and Mjelde [MM07]. We discovered that this algorithm can be smoothly extended to find approximate solutions for the general weighted k-

clique matching problem. The approximation factor did decrease in the process to $1/k$ but all other properties of the initial algorithm were preserved. Furthermore, to our satisfaction we found out that we can modify the algorithm even further to solve our three subsequent (i)–(iii) generalizations of the problem, and that both the approximation factor of $1/k$ and other properties remain in force.

To recap, first and foremost, we proved that all four algorithms are self-stabilizing, i.e. they are able to recover from transient failures and always achieve a stable state. Secondly, we showed that the solution produced by the k-clique matching algorithm is correct, unique, stable and that its total weight is at least $1/k$ of the optimal solution's weight. Moreover, these properties pertain also to the solutions produced by the remaining three algorithms, with the caveat, that in the case of K-clique (b-)matching algorithms the approximation factor is $1/\max\{k \in K\}$. Finally, we showed that all four algorithms converge in the number of rounds that is linearly proportional to the number of cliques in the final matching ($2|M| + 1$ rounds to be exact, where M is the final k-clique matching), which means that it is also linearly proportional to the number of nodes.

The result for the convergence speed is especially satisfying if we take into account the large difference in the complexity of weighted k-clique matching for $k = 2$ and $k \geq 3$. The complexity of finding the optimal solution for the former is polynomial with respect to number of nodes in the graph, while the complexity of finding the optimal solution for the latter falls into the realm of NP-hard problems. Furthermore, Preis's algorithm [Pre99] which is the sequential algorithm for finding $1/2$-approximation of weighted matching problem on which Manne and Mjelde's distributed algorithm is based, has complexity of $O(|E|)$ (worst case for the complete graph this yields $O(|V|^2)$). The analogous version of this algorithm for weighted k-clique matching has complexity of $O(k|S_k|)$ where S_k is the set of all k-cliques in the graph, thus in the worst case this gives $O(|V|^k)$. Yet, when we compare the convergence times of Manne and Mjelde's algorithm and our algorithm, we realize that both algorithms need the number of rounds which depends linearly on the number of nodes $O(|V|)$. This is indeed a very promising result, especially considering that in our simulations the convergence speed tended to be much lower than the upper bound.

However, the two distributed algorithms are not equal in terms of their complexity. The difference between them did not vanish when we moved from sequential versions to distributed ones, but it hid in the complexity of computations performed by each node in the network. In Manne and Mjelde's algorithm, each node v performs $O(|N_v|)$ edge (2-cliques) evaluations per round, while in our algorithm for weighted k-clique matching each node performs $O(|N_v|^{k-1})$ k-clique evaluations.

6.1.1. Scalability and Computational Costs

Thus, the big challenge we faced was the scalability issue of the algorithm. The culprit of this issue was not lying in the sheer number of nodes in the network, but rather it was closely tied to the number of neighbors (potential clique partners) of each node. As long as the number of neighbors per node stays relatively small, then the computational cost of the algorithm remains manageable even if the number of nodes in the network grows substantially. As an illustration, consider a network where the number of neighbors is always in the order of the logarithm of the network size ($O(\log |V|)$). Then, the number of rounds necessary for convergence is, as we already mentioned, in the order of $O(|V|)$. More importantly, the computational cost of a single round for each node v is also only in the order of $O(|V|)$ as $\binom{\log(|V|)}{k-1} \leq |V|$. Yet, what to do if the number of neighbors is not constrained by anything apart from the number of nodes in the network?

We addressed this issue in Chapter 4. Our starting point was to replace the main loop of the algorithm in which a node checks all the possible combinations of its neighbors with a heuristic. By doing so, we made it possible for nodes to control how much computational resources they are willing to spend per round. We explored two types of heuristics. The first type, which we named the attractiveness-maximizing deterministic heuristic, limited for each node the set of neighbor combinations that the node was allowed to be evaluating as potential cliques throughout the execution of the algorithm, for example by limiting the number of evaluated neighbors to some predefined constant. The second type of heuristics also limited the number of neighbor combinations evaluated by each node in each round, but at the same time guaranteed that each node is going to evaluate each possible neighbor combination infinitely often during the algorithm's execution. This can be achieved, for example, when in each round a node selects at random a subset of neighbors to evaluate.

As we learned, these two types of heuristics when incorporated into our basic k-clique matching algorithm produced algorithms of very different properties, and none of them could be called superior to the other. Instead, there are possible tradeoffs to be made depending on what is more crucial for a given application. One would choose the first type of a heuristic if preserving the small upper bound on the number of rounds necessary for convergence is more important than the quality of the final solution in which some of the nodes might remain without cliques. The choice of the second type produces the solution identical to the original 1/k-approximation one but the number of rounds until convergence is no longer bounded in a deterministic way, but instead has a significantly larger probabilistic estimate.

6.1.2. Scalability and Communication Costs

Bandwidth consumption is another scalability issue related to the number of neighbors. To verify that particular neighbors would be interested in forming a joint clique, a node needs to have accurate information whether any of these neighbors is not pursuing at the moment a higher-valued clique. Thus, when a node is choosing the best available clique in a given round, it is important for this node to have the most current information on the state of all its neighbors. To ensure this, in our basic algorithms from Chapters 2 and 3 each node broadcasts at the end of each round the weight of its currently chosen clique. This means, that in the networks where each two nodes are neighbors, a total of $O(|V|^2)$ messages is going to be delivered in each round. Even if efficient broadcasting algorithms are used, still each node has to receive $O(|V|)$ messages per round. Granting that each such message is really small and consists only of a node's identifier (128bit in size) and a clique weight (e.g. 32bit float number), in large networks (with millions of nodes) this amounts to 160 Mbits per round received by each node, which can pose a considerable burden on the nodes downstream bandwidth.

We proposed in Section 4.5 to replace the broadcasting of clique weights with gossiping. In each round, instead of broadcasting its own clique weight to all its neighbors, a node would choose one of its neighbors and send its own clique weight together with clique weights of a small number of selected neighbors. In exchange, the node receives from the chosen neighbor a same-size set of other neighbors' clique weights. Using gossiping instead of broadcasting comes at a price. With gossiping it is possible that the information that a node has on some of its neighbors is outdated. Therefore, we investigated best strategies for choosing a communication partner and selecting neighbors whose clique weights are to be sent. We achieved relatively good results in terms of convergence of k-clique matching if nodes chose a communication partner at random or the one with whom they have not communicated the longest, and exchanged the freshest information, which ensured that the new information spreads fast in the network. Yet, even better results were achieved when we tied clique weights dissemination closely with the exchange of neighbors that are directly used in computation of best cliques in the succeeding round. This approach led to putting more emphasis not on the freshness of the information but its usefulness to the node, thus improving the relevance of messages received by the nodes.

One last thing worth to mention about clique-weight dissemination is that we focused only on the communication costs, and we completely ignored the costs of storage. In our algorithms all nodes maintain a cache of all their neighbors' clique weights. Thus, the necessary storage space grows linearly with the growth of the network size. However, as the costs of storage are much lower than the communication costs, this is not an issue. Assuming the sizes of node's identifiers

and clique weights as mentioned above and adding additional 32 bits for storing the age of the information, we are still facing a need of only 24 MB to store all neighbors' clique weights in a network of 1 million nodes. Currently, such a memory requirement is negligible.

6.1.3. Multitasking Gossiping Protocols

Gossiping protocols are a simple, lightweight and robust type of peer-to-peer protocol. At the same time they are very versatile. The extensive research on these protocols shows that they can be applied to multiple means including information dissemination, distributed computations, peer sampling, topology construction, and resource management. In our own research we exploited gossiping protocols as a peer-sampling service to provide nodes executing our randomized heuristic-based k-clique matching algorithm with a subset of neighbors randomly selected from the entire network. We also used the ability of gossiping protocols to create specific topologies. In our particular case, the paradigm for the topology construction was for the nodes to find best-fitting neighbors, which were further used by our k-clique matching protocol based on the second type of investigated heuristics. In both of these cases, the main goal of using gossiping protocols, was to feed the k-clique matching algorithm with neighbor subsets of specific characteristics, and consequently limit the single-round computational costs. Our chosen gossiping protocols came with additional benefits, such as relieving the k-clique matching algorithm from the necessity of storing profile information about all neighbors and maintaining this information up-to-date, or discovering new nodes that joined the network. Yet, there were also drawbacks such as the loss of self-stabilization properties, because with the use of gossiping protocols the network may become partitioned.

Finally, we also used gossiping for clique-weight dissemination, and in Chapter 5, we created a framework which made use of all three functionalities. At first, we kept the protocols providing random neighbor selection and best-fitting neighbor discovery separate from the protocol responsible for clique-weight dissemination. Separation of these two concerns, of neighbor-profile supply and of clique-weight updates, seemed natural. Yet, when we combined all the gossiping protocols together, we realized that our k-clique matching algorithm operating on top of the gossiping protocols performs much better than when the protocols are separated. As we discovered, it was not because the timeliness of clique-weight updates improved, but rather the correlation between updated clique weights and neighbor profiles used for computations increased. Examining such synergistic effects of gossiping algorithms seems worthy of further investigation, yet unfortunately it falls out of scope of this dissertation.

6.1.4. Decentralized vs. Centralized Solution

Amongst all the deliberations about the properties of our algorithms and their engineering aspects, we should not forget our initial decision to strive for the decentralized solution. Let us now take a look at our findings and see how they stand against a possible centralized alternative.

Our biggest concern related to settling for a centralized solution involved the issues of availability and trust. For example, in an ad hoc situation there might not exists a dedicated entity that could perform the entire computation. Naturally, one of the nodes participating in the matching could take on itself finding the k-cliques for all other nodes, but such a node would be tempted to influence the solution to its own advantage, leading to other nodes questioning the fairness of the found solution. Moreover, even if a third party that can act as a broker is present, its computation still could be influenced by some participating nodes, especially if we take into account that the k-clique matching is computationally expensive and a broker might want a compensation for its services. In such a case, it is easy to imagine that there might exist nodes that would be willing to pay more to gain preferential treatment in the matching process.

In contrast, in our algorithms all nodes are treated equally. Moreover, there is no way for the nodes to create a clique better than the one dictated by the stability of the final k-clique matching. Naturally, malicious nodes could lie about their profile information, but this is possible also in a centralized approach. Yet, there is no advantage for the nodes in lying about their chosen clique weights. Consider a node v spreading the information that its current clique weight is of weight $\mathbf{w}_v$. This information would only directly influence those v's neighbors that could potentially create a clique with v. If there are neighbors that can create with v a clique whose weight is higher than $\mathbf{w}_v$, then by sticking to $\mathbf{w}_v$, node v is only sabotaging itself, as it is passing on a better offer. On the other hand, if there is no neighbor that could offer v a clique with weight better than $\mathbf{w}_v$, but there are neighbors that could create a clique with v with a smaller weight, than v's neighbors will form other cliques and v will be left without any clique partners, which again puts v in a lost position. Thus, the only harm done by such a tactic is the disruption of the algorithm's performance as some nodes might not be able to form cliques due to malicious behavior of other nodes. Even in such a case we can actually see this as a completely legitimate way for the node to inform its neighbors that the minimal clique weight they are interested in is $\mathbf{w}_v$.

Now consider those algorithms that, apart from sending clique-weight updates, also send separately clique offers to other members of the chosen clique. A malicious node v may also try to tamper with the offer message to trick other nodes into forming a particular clique. For example, when trying to form a clique with nodes u and t, node v can modify u's profile that is included in the offer sent

to t, such that clique $\{t,u,v\}$ in the offer has weight higher than the original profiles of t, u, and v indicate. Similarly, v can modify the profile of t when sending an offer to u. Yet, such a move can be quickly discovered by nodes t and u. As soon as these nodes fall for the trap, they will send to each other their own offer messages. In these messages, the profiles of t and u will be correct. Thus, upon receiving the offer from u, node t will have the correct profile of u and the weight recomputation of clique $\{t,u,v\}$ will yield the correct value. The same will happen at node u. In the end, v's plot will be thwarted. The intrinsic redundancy in computations and messages, makes it almost impossible for a node to deceive its neighbors into unfavorable clique choices. Naturally, v may try a more advanced approach by intercepting the communication between t and u. To counteract such attempts, all offer messages (as well as all other messages in the system) can be digitally signed, although such a solution would require some sort of a public key infrastructure to be in place.

Apart from the trust issues, the centralized solution has other drawbacks. In our algorithms each node performs its own part of computation. In contrast, the centralized entity immediately becomes a potential bottleneck, as it must perform the entire computation on its own. Additionally, the centralized component is also a single point of failure and with its failure, the entire computation is lost and must be redone by another centralized entity. In a decentralized case, no such single point of failure exists. In case of a node failure and the discovery of this fact by its neighbors, the neighbors only need to update their own neighbor sets and potentially readjust their clique choices, if those involved the failed node. Apart from that the computation of k-clique matching carries on undisrupted. Thus our algorithms are very robust, largely thanks to their self-stabilization property.

Naturally, the centralized solution also has some advantages. In a decentralized case, many nodes consider the same clique simultaneously during a single round. Additionally, the same cliques might be reconsidered over and over again by the same node in subsequent rounds. A centralized solution having the full control over the computation, can easily avoid duplication of effort. Moreover, the centralized component can make the entire computation atomic. Once it arrives at the solution, the formed cliques are final. Conversely, in our algorithms nodes may be changing their decision on a clique choice multiple times and their decisions do not have to agree, resulting in the system being in a partially inconsistent state before it finally converges. Further, as our algorithms are based on self-stabilization, their computation never really ends, thus to form final cliques, the nodes have to resort to separate consensus protocols such as two-phase commit before they can finalize their choice.

Weighing in the advantages and drawbacks of both approaches, we feel that with the robustness and intrinsic trust enforcement our algorithms make a strong

case in favor of a decentralized solution.

6.1.5. Discussion Summary and Conclusions

At the beginning of this dissertation we put forward four research goals that we aimed to achieve while devising a distributed K-clique b-matching algorithms. Those goals were pervasive throughout the chapters, thus we recap here shortly our approaches to address each of the goals.

Our first goal revolved around the fairness of proposed algorithms, which we achieved in two steps. First, the fairness of all our algorithms stems directly from the fact that all nodes execute the same code, with no node playing a specific role that could allow it to gain a preferential position in the system. Second, although the decisions made by the nodes are selfish in nature, with each node trying to maximize the weight of its clique, the algorithms impose that nodes also respect their neighbors' choices. This is governed by the rule present in all our algorithms that a node should choose in each round the best available clique among the *proper* ones as defined in Section 2.2.2. This leads nodes (in most versions of our algorithms) to the formation of a *stable* clique matching, in which there is no group of nodes that would prefer to form a clique together outside of this matching instead of staying in their current situation. In some versions of our algorithms reaching a stable clique matching is abandoned in the light of other objectives. For instance, in the algorithms using an attractiveness-maximizing deterministic heuristic (from Section 4.1.2) and algorithms in which nodes can leave the system once they found a satisfactory clique before full convergence is reached (described in Section 4.2) reaching a stable clique matching is traded for a swift formation of final cliques and a fast convergence of the system. Yet, even in those algorithms all nodes follow the rule to select the best clique out of the proper ones which lets them reach an agreement on which cliques to form.

Our second goal pertained to the quality of the clique matching constructed by our algorithms. For the algorithms in Chapters 2 and 3 we proved that the total weight is at most k times worse than the optimal k-clique matching. Although this approximation factor is not high, we did not try to improve it, as for applications in which nodes are inherently selfish, and thus preoccupied with maximizing their own choices rather then the global state, the quality of the total clique matching seems less important then ensuring fairness of the algorithms and the stability of the final solution.

Our third goal revolved around scalability issues. Although our initial algorithms from Chapters 2 and 3 have very quick convergence times (linear to the number of cliques in the final clique matching), their weak points are computational and communication costs that grow quickly with the increasing number of neighbors per node. To address scalability issues related to computational costs of

our algorithms we proposed various heuristics, each coming with different trade-offs ranging from a decreased quality of a solution (for attractiveness-maximizing deterministic heuristics) to larger communication costs and potentially smaller convergence speed (for randomized heuristics). Further, we showed how some of these heuristics (HEAVIESTEDGESUBSET and RANDOMSUBSET heuristics in particular) can be combined with gossiping protocols. These protocols can provide nodes with partial views of the network having desired properties, e.g. a subset of most suitable or random neighbors. Such use of gossiping protocols additionally alleviates another scalability issue related to the necessity of nodes to possess full knowledge about their closest neighborhood. We also utilize gossiping protocols in one other way: as a replacement for broadcasting of messages, which decreases the communication costs of our algorithms.

Our fourth and final goal is related to robustness. The robustness of our algorithms from Chapters 2 and 3 is ensured by their self-stabilizing property. By definition, this guarantees that these algorithms can gracefully recover from any transient errors such as nodes joining or leaving the system, or messages getting lost or becoming corrupted. By extending these initial algorithms with gossiping protocols we lost the self-stabilization property. Nonetheless, due to the self-healing properties of gossiping protocols, the robustness of our system has been largely preserved.

Summarizing, we proposed self-stabilizing approximation algorithms that solve the weighted k-clique matching problem and a range of its generalizations. Our basic versions of the algorithms have very efficient convergence speeds in terms of the overall number of rounds with a theoretical upper bound of $2|V|/k+1$, and as our simulations revealed, the number of rounds till convergence is much lower in practice. Nonetheless, the computational and communication loads do not scale well for these algorithms when the number of neighbors is relatively large in comparison to the network size. We addressed computational performance issues by introducing heuristics, and we suggested gossiping instead of broadcasting to circumvent bandwidth considerations. Additionally, when exploring heuristic-based solutions, we also played with the idea of executing them on top of partial views maintained by selected gossiping protocols such as CYCLON and VICINITY.

In the end, we combined all of the above findings into one coherent framework, and created an efficient, fully distributed k-clique matching service. In comparison with a centralized approach, this service has several important advantages: it is robust in case of nodes joining or leaving the network, all nodes are equal and in full control over their decisions, and yet at the same time, the service intrinsically imposes non-malicious behavior. As such, we feel that this service is well suited for many applications that demand creating non-overlapping (or overlapping limited number of times) groups of nodes.

6.2. FUTURE DIRECTIONS

In this dissertation we fulfilled our main research goal of devising a robust decentralized algorithm for small partnership formation. Nonetheless, our work does not exhaust the topic, and we can derive from it further research questions to complement and extend this research.

First and foremost, in our dissertation we abstracted from the discussions on the sizes of k, mentioning only that it should be relatively small. In fact, the correctness of our algorithms from Chapters 2 and 3 or the properties of the produced solution are not affected in any way by the value of k. Additionally, the number of rounds necessary for the convergence of these algorithms actually decreases with the increase of k's value. Therefore, the only place where the value of k can have a negative impact is related to the computational load of a single round, where each node v has to choose the best available clique among $\binom{|N_v|}{k-1}$ possibilities. Thus, the most computationally expensive are values of k closest to $(|N_v|+2)/2$. This impact further translates to the performance of heuristic-based algorithms presented in Chapter 4, where for example the value of k will directly influence the estimate convergence time of the k-clique matching algorithm that uses a randomized heuristic, or where the high value of k will significantly limit the usefulness of small partial views. For this reason, a logical next step for this research could be to analyze the relation between the value of k and the performance of the algorithm employing various heuristics. We have already made a step into that direction, by running simulations for k-cliques up to size 5, but a more comprehensive study would be beneficial.

In our current algorithms, the nodes are greedily searching for better cliques. Even if the clique weight improvement is minute they will switch from a slightly worse clique to a better one. Because of that any changes to the network (caused by the node churn, changes in edge weights, etc) could result in a snowball effect of many nodes readjusting their choices. A possible research project could analyze the impact of such phenomena. Subsequently, we could try to prevent such volatile behavior. For example, we could let the nodes have a δ value specifying how much better a clique must be for them to consider a change. In such a scenario, nodes will change an old clique for a new one only if the new clique would result in a δ improvement over the old clique for all new clique members. As a result, the final k-clique matching might not be stable anymore (rather it would be stable with regard to the value of δ). Moreover, the final solution might also not be unique, as two separate runs of the algorithm, due to its chaotic behavior, can produce two different k-clique matchings. Thus, it would be interesting to investigate what impact the value of δ would have on the average weight of cliques in the final matching and how sensitive this value can be from one algorithm's execution to

the other. For example, it might be possible that with δ, although some nodes passed on better cliques, the average weight of cliques in the final matching might be higher than the average in the basic algorithm from Chapter 2.

In this thesis, we geared our considerations on the performance of algorithms towards computational and communication scalability questions in situations where any two nodes in the system are neighbors. Such systems are most likely to occur in wired networks, where available infrastructure alleviates to a great extent any constraints regarding node connectivity. In contrast, in wireless networks we expect that radio ranges will create a natural restriction on node neighborhoods, with two nodes being potential clique partners only if a two-way communication can be established between them. In such a setting, as long as the network density is kept in check, the growth of the network size will not pose scalability issues with regard to computational load of a single round. Additionally, wireless communication is inherently broadcast-based, which also acts to our advantage, as it relieves us from devising special protocols to efficiently reach all the neighbors with clique weight updates. Yet, wireless environment come with their own set of intrinsic challenges. For example, one possible research direction would be to investigate to what extent nodes situated at the verge of the network are susceptible to being left without a clique, and further to explore if it is possible to increase their chances of forming a clique, for instance, by incorporating into a node's profile information about its position in the network.

Although we have shown that malicious nodes cannot gain a better clique by faking their clique weight updates or clique offers, their wrong-doing still can disrupt the algorithm's performance. Therefore, another possible research direction can be related to the examining of the impact malicious nodes can have on the algorithms and devising mechanisms to discover and potentially remove malicious nodes from the system.

Lastly, the major strength of our algorithms lies in the fact that they are in principle application independent. We have complete freedom in the choice of profile structure or clique weight function. Therefore, to establish a baseline of our algorithms performance, we carried out simulations with varied node profile characteristics resulting in different edge weight distributions. Our findings show that both convergence speed and the average clique weight are influenced by those factors. Cross-examining these results against potential applications could reveal new areas in which our algorithms could be improved in general or by tailoring them for particular application scenarios.

APPENDIX A

Various Edge Weight Distributions

All of the simulations presented in Chapter 2 were conducted in networks where the weight between any pair of nodes is assigned uniformly at random from the $(0, 1)$ interval. This, we believe, constitutes a good base line due to its statistical properties; the mean edge weight and mean clique weight (irrespective of its size) is expected to be close to 0.5 in any graph with such generated edge weights. At the same time such distributions are unlikely to occur in real life applications, in which some correlation between edge weights is to be expected. Therefore, it is interesting to compare the performance of the protocol with uniformly at random assigned weights to different schemes for edge weights assignment.

A.1. CHOICE OF WEIGHT DISTRIBUTIONS

In Section 2.1 we mentioned that each node has its own profile. Such a profile should contain all information about the node's qualities and capabilities meaningful from the point of view of the application for which the clique matching is to be created. With regard to technical aspects of profile specification, there is a great freedom in choosing the technology most suitable for the particular application. The only constraint is that it must be possible to derive from the profiles of nodes, the weight of the clique formed by these nodes either directly or by computing the pairwise edge weights first. We foresee that for many applications it will be sufficient to simply encode the properties of nodes into finite-length numeric vectors whose value at the *i*th position corresponds to the *i*th characteristic, capability, or resource possession of a node. This manner of modelling node profiles is often encountered in multi-agent systems, e.g. [SK98].

Such defined profiles allow us to map all the nodes into a multi-dimensional space, where the number of dimensions is equal to the length of the vector. Then, in principle, any symmetric function (whose value does not depend on the order of arguments) that takes k d-dimensional points and returns a single (real) value can be used as a clique weight measure. If we want to derive edge weights first, we can use two symmetric functions: first one which takes two d-dimensional points and returns an edge weight, second one which takes $k(k-1)/2$ edge weights to produce the clique weight. To demonstrate the operation of our k-clique matching algorithm under edge-weight distributions different from the uniform distribution, we adopt the second approach and focus on the following two test cases:
Similarity Test Case: Node profiles are vectors of d properties and the goal of nodes is to form cliques based on the pairwise similarity of the profiles. As a measure of similarity between any two profiles (edge weight), the Euclidean distance is used. A clique weight is computed as an average of edge weights.
Dissimilarity Test Case: The settings for this test case are identical as for the similarity test case, but now, the nodes try to maximize their clique weights.

A.2. EXPERIMENTAL EVALUATION

We test the performance of our k-clique matching protocol for networks in which nodes have profiles represented as vectors of length d. For each node the values of its vector coordinates are drawn uniformly at random from the $(0,1)$ interval. The edge weight between two nodes v and u is computed as the Euclidean distance between their respective vectors of properties $(x_i, x_2, \ldots, x_d)$ and $(y_i, y_2, \ldots, y_d)$, which are treated here as d-dimensional coordinates:

$$w(v,u) = \sqrt{\sum_{1 \le i \le d} (x_i - y_i)^2},$$

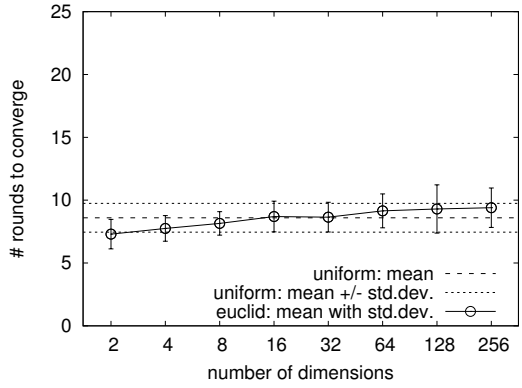
and the weight of a clique $Q = \{v_1, \ldots, v_k\}$ is the average of the edge weights between all clique members:

$$w(Q) = \frac{2 \sum_{1 \le i \le k} \sum_{i < j \le k} w(v_i, v_j)}{k(k-1)}.$$

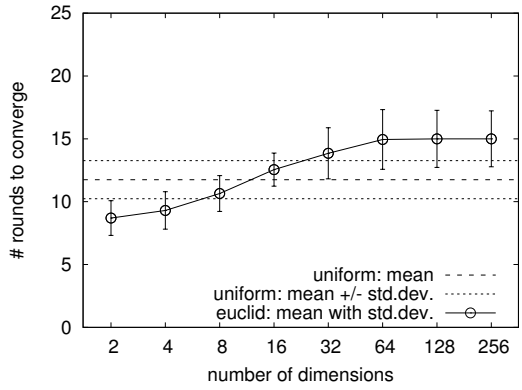
We compare the obtained simulation results to the base case in which each edge weight is assigned uniformly at random.

A.2.1. Similarity Test Case

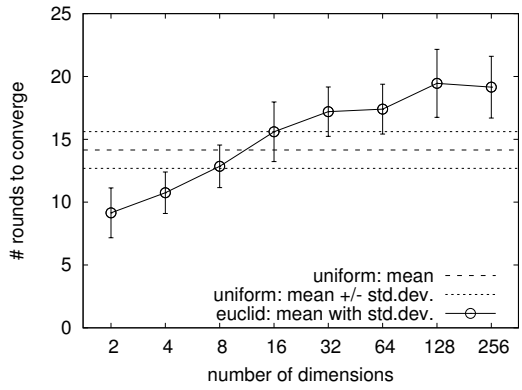
Figure A.1 depicts the average convergence times (in rounds) for networks of 240 nodes. Each graph is dedicated to a different clique size k and in every graph the



(a) $k = 2$

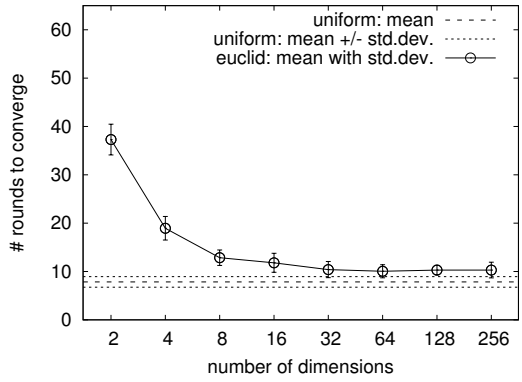


(b) $k = 3$

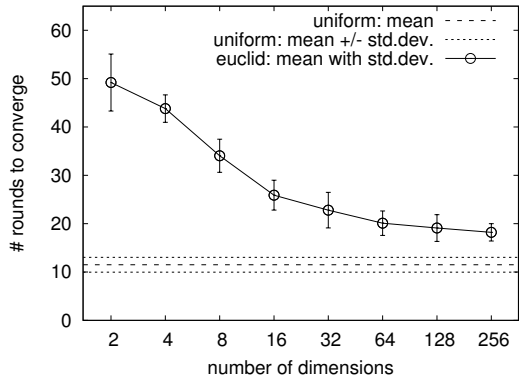


(c) $k = 4$

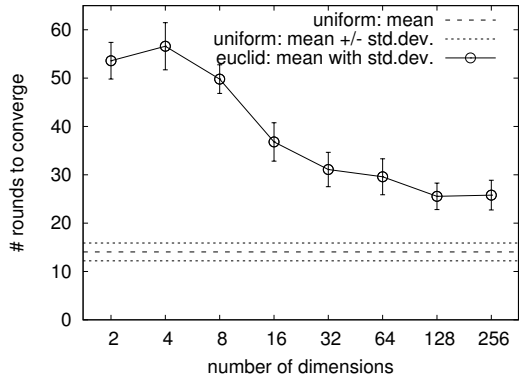
Figure A.1: Similarity Test Cases: Convergence under d-dimensional Euclidean edge metric in 240-node network.



(a) $k = 2$



(b) $k = 3$



(c) $k = 4$

Figure A.2: Dissimilarity test case: Convergence under d-dimensional Euclidean edge metric in 240-node network.

Figure A.3: Example of an 8-node network with profiles as vectors of length 1.

results are plotted for the number of dimensions increasing from 2 to 256. We can clearly see the pattern. With the increase in the number of dimensions, the convergence time also increases. In particular, for a small number of dimensions, such as 2 or 4, the average convergence time tends to be lower than the convergence time in the network with edge weights assigned uniformly at random. While for large number of dimensions, the average number of rounds necessary for the network to converge tends to be larger than in the base case. Moreover, the differences between the convergence times for a particular number of dimensions and the base case become more prominent with the increase of clique size k from 2 to 4. Nonetheless, the convergence times for any number of considered dimensions tend to be close to the convergence times in the base case, which suggest that our base case can serve as a good representative of k-clique matching algorithm performance also for graphs defined in accordance to Similarity Test Cases.

A.2.2. Dissimilarity Test Case

Figure A.2 depicts the average convergence times for the same simulation parameters as used in the examination of Similarity Test Case. Thus, we can clearly see the difference in the k-clique matching performance when the objective of the nodes changes. First, we observe that the average convergence times for all tested numbers of dimensions are larger than the average convergence time for the base case. Moreover, these times decrease with the increase in the number of dimensions.

A.2.3. Analysis of Two Cases

Here we will try to provide an explanation for the differences in the convergence times of the k-clique matching algorithm between the two presented cases. Consider a small network of 8 nodes, whose profiles are vectors of length 1 (see Figure A.3). In such a case, nodes can be mapped into a 1-dimensional space (straight line) and the weight of an edge between any two nodes directly corresponds to the distance between these two nodes on the line. In this network, nodes execute the 2-clique matching protocol.

Let us first assume that nodes are trying to form the cliques with the smallest weights possible (Similarity Test Case). Then, we can see that for each node

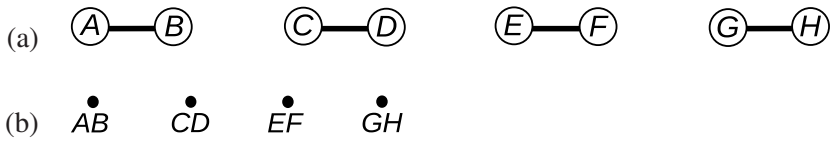


Figure A.4: Similarity test case: (a) final 2-clique matching, (b) graph of stability dependencies between cliques from the final 2-clique matching.

the lightest possible clique is also one of the locally lightest cliques of the entire network. Therefore, the final 2-clique matching emerges in the network almost instantaneously (see Figure A.4(a)).

The situation looks very different if nodes are trying to form cliques with the highest weights possible (Dissimilarity Test Case). Then, we can see that only for nodes A and H, the heaviest possible clique AH is also the locally heaviest clique of the entire network. For all other nodes, their heaviest possible clique contains either A or H and is lighter than the locally heaviest clique AH. Therefore, nodes A and H become matched almost immediately (see Figure A.5(a)), while the other nodes are trying to create a clique with either A or H until they learn that they cannot create a proper clique with these nodes. Only once AH is created, are nodes B and G able to create a stable clique BG (Figure A.5(b)), followed by nodes C and F abandoning their attempts to become matched with either G or B and forming clique CF (Figure A.5(c)). Finally, nodes D and E give up on creating heavier cliques and settle down on forming clique DE (Figure A.5(d)) and the network stabilizes.

The presented scenarios of convergence are consistent with our simulation results. Let us take a closer look at our results for $k = 2$ in the networks of 240 nodes with profiles as vectors of length 2 and plot in Figure A.6 the percentages of nodes that are correctly matched in a given round along with the percentages of nodes that are in stable cliques, that is, in cliques that will remain correctly matched permanently and, thus, will be part of the final k-clique matching. We can observe that in the similarity test case the percentage of nodes in stable cliques is only slightly lower than the percentage of nodes in cliques that are correctly matched but might not be stable. Moreover, in each round the number of nodes that create stable cliques is significant and the network converges in just 8 rounds on average. On the other hand, in the dissimilarity test case the number of nodes that are correctly matched largely differs from the number of nodes in stable cliques, which means that a lot of cliques that are created in some given round will soon fall apart, while the number of stable cliques grows only by a few in each round.

To better understand the principle behind the differences in the convergence

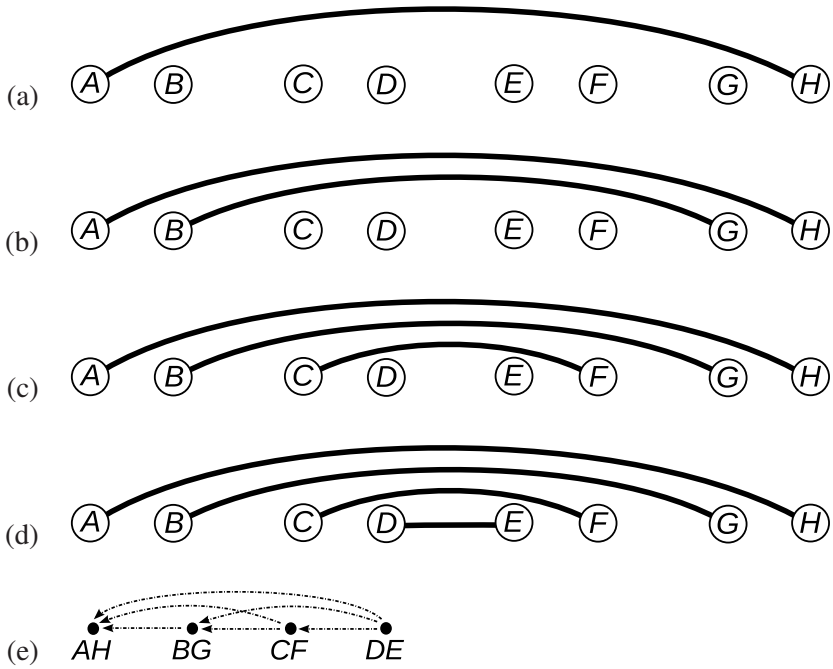


Figure A.5: Dissimilarity test case: (a)-(d) projected steps of stable cliques formation until the final 2-clique matching emerges, (e) graph of stability dependencies between cliques from the final 2-clique matching.

between the two test cases, let us define the *stability dependency* between the non-overlapping cliques in the graph. We will say that one clique Q_i depends on another non-overlapping clique Q_j in terms of stability if there exists a node v in Q_i that would prefer to form some clique Q_k whose (at least one) other node is in clique Q_j. For example, in Figure A.7, $w(BC) < w(BD) < w(AD)$, and we can observe that clique BC depends on clique AD in terms of stability in the Dissimilarity Test Case, because B would prefer to create the heavier clique BD over the lighter clique BC. The situation is reversed in similarity test case, where AD depends on BC in terms of stability, because D prefers the lighter clique BD over AD. Figure A.4(b) and Figure A.5(e) show all stability dependencies between the cliques from final matching in the respective test cases. In Similarity Test Case example there are no clique dependencies, while in Dissimilarity Test Case we can see that these relations create chains of dependencies, the longest of which contains all cliques from the final matching.

The intuition is that the longer the chains of stability dependencies between the cliques from the final matching, the longer the convergence takes. The reasoning

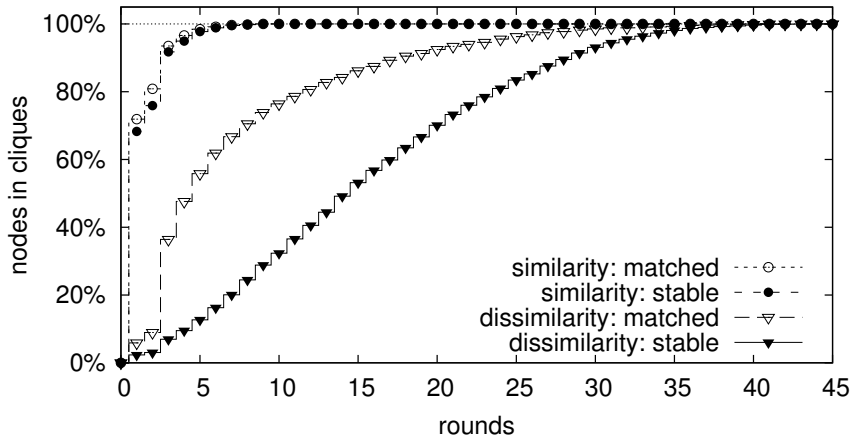


Figure A.6: Percentage of nodes in stable 2-cliques over time (in rounds) in networks of 240 nodes with profiles as vectors of length 2.

Figure A.7: Dissimilarity Test Case: clique BC depends on clique AD in terms of stability; Similarity Test Case: clique AD depends on clique BC in terms of stability.

behind it is as follows. Consider the nodes that will belong to two cliques Q_i and Q_j in the final k-clique matching such that Q_i depends in terms of stability on Q_j. For instance, consider nodes B, C, F, G from the Dissimilarity Test Case example in Figure A.5(d). These nodes form in the final matching cliques BG and CF. Moreover CF depends on BG, because C prefers clique CG over CF and F prefers clique BF over CF. Then, as long as nodes B and G are not stably matched together into clique BG but are still in the process of looking for their most preferred clique, their values of variables $\mathbf{w}$ are volatile, and can take values that are smaller than $w(BG)$. If, for example, the value of $\mathbf{w}_B$ becomes low enough, node F will try to form clique BF, blocking at the same time the creation of clique CF, until BF stops being proper. An analogous situation may occur between nodes G and C. Thus, the clique CF may be permanently formed only once BG is permanently formed which in turn depends on clique AH. As a result, the convergence time of the k-clique matching protocol will be proportional to the length of the stability dependency chains of cliques in the graph.

APPENDIX B

Arithmetic Mean Of Clique Edges Weights

Lemma B.1. *Let $G = (V,E)$ be a weighted graph with $w(e) \to (0,1)$ as a function that assigns weight to each edge $e \in E$. Then for any clique with at least three vertices T in G there exists a clique S such that $S \subsetneq T$, $|V(S)|+1 = |V(T)|$, and the average weight of edges in S, $\omega(S)$, is larger than the average weight of edges in T, $\omega(T)$, i.e.:*

$$\omega(T) = \frac{1}{\binom{|V(T)|}{2}} \cdot \sum_{e \in E(T)} w(e) \leq \frac{1}{\binom{|V(S)|}{2}} \cdot \sum_{e \in E(S)} w(e) = \omega(S)$$

Proof. First, notice that:

$$\sum_{e \in E(T)} w(e) = \frac{1}{2} \sum_{v \in V(T)} \sum_{u \in V(T)-\{v\}} w(\langle v,u \rangle),$$

because in the right-hand side of the equation the weight of each edge $e = \langle v,u \rangle \in E(T)$ is added twice — first, when the inner sum is over all vertices belonging to $V(T)-\{v\}$, second, when the inner sum is over all vertices belonging to $V(T)-\{u\}$. Using that observation, we can express $\omega(T)$ as follows:

$$\begin{aligned}
\omega(T) &= \frac{1}{\binom{|V(T)|}{2}} \sum_{e \in E(T)} w(e) \\
&= \frac{1}{\binom{|V(T)|}{2}} \frac{1}{2} \sum_{v \in V(T)} \sum_{u \in V(T)-\{v\}} w(\langle v,u \rangle) \\
&= \frac{1}{|V(T)| \cdot (|V(T)|-1)} \sum_{v \in V(T)} \sum_{u \in V(T)-\{v\}} w(\langle v,u \rangle) \\
&= \frac{1}{|V(T)|} \sum_{v \in V(T)} \frac{1}{|V(T)|-1} \sum_{u \in V(T)-\{v\}} w(\langle v,u \rangle)
\end{aligned}$$

where the last expression can be read as an average of averages of edges incident to vertices from T.

Among $|V(T)|$ components $\frac{1}{|V(T)|-1}\sum_{u\in V(T)-\{v\}} w(\langle v,u\rangle)$ there must exist at least one such that its value is smaller or equal to their average $\omega(T)$. Let t denote a vertex in $V(T)$ such that $\frac{1}{|V(T)|-1}\sum_{u\in V(T)-\{t\}} w(\langle t,u\rangle) \leq \omega(T)$ and let S be a clique induced by vertices in $V(T)-\{t\}$. Then:

$\omega(T) = \frac{1}{|V(T)|\cdot(|V(T)|-1)}\sum_{v\in V(T)}\sum_{u\in V(T)-\{v\}} w(\langle v,u\rangle)$ $\Leftrightarrow$

(reordering the components

$\omega(T) = \frac{1}{|V(T)|\cdot(|V(T)|-1)}\left(\sum_{v\in V(S)}\sum_{u\in V(S)-\{v\}} w(\langle v,u\rangle) + 2\sum_{u\in V(S)} w(\langle t,u\rangle)\right)$ $\Leftrightarrow$

first sum expressed with $\omega(S)$

$\omega(T) = \frac{1}{|V(T)|\cdot(|V(T)|-1)}\left(|V(S)|(|V(S)|-1)\omega(S) + 2\sum_{u\in V(S)} w(\langle t,u\rangle)\right)$ $\Rightarrow$

from the assumption about t

$\omega(T) \leq \frac{1}{|V(T)|\cdot(|V(T)|-1)}\left(|V(S)|\cdot(|V(S)|-1)\omega(S) + (|V(T)|-1)\omega(T)\right)$ $\Leftrightarrow$

$\omega(T) \leq \frac{|V(S)|\cdot(|V(S)|-1)}{|V(T)|\cdot(|V(T)|-1)}\omega(S) + \frac{1}{|V(T)|}\omega(T)$ $\Leftrightarrow$

$\frac{|V(T)|-1}{|V(T)|}\omega(T) \leq \frac{|V(S)|\cdot(|V(S)|-1)}{|V(T)|\cdot(|V(T)|-1)}\omega(S)$ $\Leftrightarrow$

from the assumption $V(S) = V(T)-\{t\}$

$\frac{|V(T)|-1}{|V(T)|}\omega(T) \leq \frac{|V(T)|-2}{|V(T)|}\omega(S)$ $\Leftrightarrow$

$\omega(T) \leq \frac{|V(T)|-2}{|V(T)|-1}\omega(S)$ $\Rightarrow$

$\omega(T) < \omega(S)$

□

SUMMARY

Decentralized k-Clique Matching

When two or more brands collaborate together to create a new product, to offer a bundle of their products, or to put forward a joint marketing campaign, we call such collaboration a brand alliance. An example of such alliance is the partnership between Nike and Apple whose result was creation of *Nike+iPod Sports kit*, a device for tracking workout performance for runners.

One of the decisive factors to the success of brand alliances is the choice of suitable partners. Yet finding the most suitable partners can be a complex and time consuming task, especially if we take into account that with a large number of brands, the number of possible combinations of two or more of them is vast. At the same time each of these brands is guided by their own self-interest which makes reaching an agreement more difficult.

The above problem of forming most promising partnerships amidst a large pool of brands can be modeled as a *weighted k-clique matching* problem, or one of its generalizations. To this end, assume that each brand is represented by a vertex in a graph and the weight of an edge corresponds to a fit estimation of two brands when paired up. The k-clique matching is then defined as a set of disjoint cliques, each with k vertices, and the goal is to find such a set with the highest total weight of cliques. Possible generalizations of this problem include relaxing the constraint on the clique sizes or the number of cliques per node in the clique matching.

In this thesis, we propose to solve the above problems in a fully decentralized fashion where each brand is a node in a computer network. We put special attention to the fairness, scalability and robustness of the devised algorithms as well as the quality of the final k-clique matching constructed by these algorithms. We support the ideas laid down in this thesis with both theoretical analysis and experimental validation.

We start off by introducing distributed self-stabilizing approximation algo-

rithms for solving the weighted k-clique matching problem and its generalizations. In these algorithms, the formation of distributed cliques emerges from the local decisions of each node based only on the information limited to its direct neighborhood. The fairness of these algorithms stems directly from the fact that all nodes execute the same code, with no node playing a specific role that could allow it to gain a preferential position in the system. Moreover, although the decisions made by the nodes are selfish in nature, with each node trying to maximize the weight of its clique, the algorithms impose that nodes also respect their neighbors' choices. This leads nodes to the formation of a *stable* clique matching, in which there is no group of nodes that would prefer to form a clique together outside of this matching instead of staying in their current situation. Apart from the stability of the final clique matching, we also prove that its total weight is at most k times worse than the optimal k-clique matching. Although this approximation factor is not high, we do not try to improve it, as for applications in which nodes are inherently selfish, and thus preoccupied with maximizing its own choices rather than the global state, the quality of the total clique matching seems less important than ensuring fairness of the algorithms and the stability of the final solution. Finally, the robustness of our initial algorithms is ensured by their self-stabilizing property. By definition, this guarantees that these protocols can gracefully recover from any transient errors such as nodes joining or leaving the system, or messages getting lost or becoming corrupted.

Although these initial algorithms have very quick convergence times (linear to the number of cliques in the final clique matching), their weak points are computational and communication costs that grow quickly with the increasing number of neighbors per node. To address scalability issues related to computational costs of our algorithms we propose various heuristics, each coming with different tradeoffs ranging form decreased quality of solution to larger communication costs and potentially smaller convergence speed. Further, we show how some of these heuristics can be combined with gossiping protocols. These protocols can provide nodes with partial views of the network having desired properties, e.g. a subset of random or most suitable neighbors. Such use of gossiping protocols additionally alleviates another scalability issue related to the necessity of nodes to possess full knowledge about their closest neighborhood. We also utilize gossiping protocols in one other way: as a replacement for broadcasting of messages, which decreases the communication costs of our algorithms.

We combined all of the above findings into one coherent framework creating an efficient, fully distributed k-clique matching service well suited for brand alliances formation as well as many other applications that demand creating non-overlapping (or overlapping a limited number of times) groups of nodes.

SAMENVATTING

Gedecentraliseerde k-Clique Matching

Als twee of meer merken samenwerken om gezamenlijk een nieuw product te creëeren, een bundel van producten aan te bieden, of om een gezamenlijke marketingcampagne op te zetten, noemen we deze samenwerking een brand alliance. Een voorbeeld van een brand alliance is de partnerschap tussen Nike en Apple resulterende in de *Nike+iPod Sports kit*, een apparaat voor het bijhouden van trainingsprogramma's van renners.

Een van de doorslaggevende factoren voor het succes van brand alliances is de keuze van passende partners. Echter, het vinden van passende partners kan een complexe en tijdrovende taak zijn. Met name geldt dit als we het grote aantal merken in ogenschouw nemen, wat betekent dat het aantal mogelijke combinaties van twee of meer merken enorm is. Tegelijkertijd laat elk van deze merken zich leiden door hun eigenbelang, wat het bereiken van een overeenkomst nog lastiger maakt.

Het bovengenoemde probleem, betreffende het vormen van de meest veelbelovende partnerschappen uit een verzameling van merken, kan gemodeleerd worden als een *weighted k-clique matching* probleem of als een mogelijke generalisatie daarvan. Neem hiervoor aan dat elk merk kan worden uitgedrukt als een knoop van een graaf, en dat de fitness van een partnerschap tussen twee merken kan worden uitgedrukt in het gewicht van een lijn van de graaf. De k-clique matching bestaat dan uit een verzameling losse cliques, elk met k knopen. Het doel is dan het vinden van een verzameling met het hoogste totaalgewicht van cliques. Een mogelijke generalisatie van dit probleem is het minder strikt zijn betreffende de grootte van de cliques, of betreffende van het aantal cliques per knoop in de clique matching.

In dit proefschrift stellen wij voor om de genoemde problemen op een volledig

gedecentraliseerde manier op te lossen, waarbij elk merk een node ('knoop') in een computernetwerk is. In het bijzonder letten we op fairness, schaalbaarheid en robuustheid van onze algoritmes, alsook op de kwaliteit van de k-clique matching voorgesteld door deze algoritmes. We ondersteunen de ideëen in dit proefschrift met zowel theoretische analyse als experimentele validatie.

We beginnen met de bespreking van gedistributeerde, zelf-stabiliserende, approximatie (benaderings) algoritmes voor het oplossen van een weighted k-clique matching probleem en generalisaties daarvan. In deze algoritmes worden cliques gevormd door lokale beslissingen van elk node, enkel gebaseerd op informatie uit de directe, nabijgelegen, buurt. De fairness van deze algoritmes komt voort uit het feit dat alle nodes dezelfde code uitvoeren, zonder dat een node een speciale rol heeft die tot een voorkeurspositie in het systeem zou kunnen leiden. Daarnaast: hoewel de beslissingen van de nodes van nature uit eigenbelang worden genomen, waarbij elk node het gewicht van zijn eigen clique probeert te maximaliseren, zorgen de algoritmes er voor dat nodes de keuzes uit hun buurt respecteren. Dit leidt nodes tot de vorming van een *stabiele* clique matching, waarbij er geen groep van nodes bestaat die liever een clique buiten de matching vormt.

Naast stabiliteit van de resulterende k-clique matching, bewijzen we ook dat het totale gewicht van de k-clique matching op zijn hoogst k maal erger is dan de optimale k-clique matching. Hoewel deze benaderingsfactor niet hoog is, proberen wij hem niet te optimaliseren. Dit doen wij omdat voor toepassingen waarbij nodes inherent in eigenbelang dienen, en dus voornamelijk bezig zijn met het maximaliseren van de eigen keuzes in plaats van de globale status, de kwaliteit van de totale k-clique matching ons minder belangrijk lijkt dan het zorgen voor fairness van de algoritmes, en de stabiliteit van de uiteindelijke oplossing.

Tot slot wordt de robuustheid van onze algoritmes gewaarborgd door hun eigenschap van zelf-stabilisering. Per definitie garandeert deze eigenschap dat onze algoritmes goed kunnen herstellen van niet-systematische fouten, zoals nodes die bij het systeem komen of die het systeem verlaten, of het verloren danwel anderszins mis gaan van berichten.

Hoewel onze algoritmes een korte convergentietijd hebben (lineair met het aantal cliques in de resulterende clique matching), is hun zwakke punt dat de kosten voor communicatie en berekening snel groeien als het aantal buren per node groeit. Als maatregel voor de schaalbaarheidsproblemen, gerelateerd aan computationele kosten, stellen we verschillende heuristieken voor. Deze heuristieken impliceren elk verschillende afwegingen, van een lagere kwaliteit van de oplossing tot hogere communicatie-kosten, tot een mogelijk langere convergentietijd.

Daarnaast laten we zien hoe deze heuristieken gecombineerd kunnen worden met gossiping protocollen. Deze protocollen kunnen voor nodes, voor een deel

van het netwerk, de gewenste eigenschappen identificeren. Bijvoorbeeld: een deelverzameling van willekeurige of gewenste buren. Het gebruik van gossiping protocollen heeft nog een ander positief effect, namelijk op het schaalbaarheidsprobleem betreffende de noodzaak van nodes om volledige informatie van de dichtstbijzijnde buren te bezitten. We gebruiken gossiping ook nog op een andere manier: als vervanger van het broadcasten van berichten. Dit vermindert de communicatiekosten van onze algoritmes.

We hebben de bovenstaande bevindingen tot een coherent raamwerk gecombineerd. Zodoende hebben we een efficiente, volledig gedistribueerde k-clique matching dienst gecreëerd die gebruikt kan worden voor de vorming van brand alliances. Daarnaast is onze dienst toepasbaar voor elk probleem dat gekarakteriseerd wordt door niet-overlappende groepen van nodes (of althans groepen van nodes met een beperkte overlap).

BIBLIOGRAPHY

[Aak97] Jennifer L. Aaker. Dimensions of brand personality. *Journal of Marketing Research*, 34(3):347–356, August 1997.

[AH98] E.M. Arkin and R. Hassin. On local search for weighted k-set packing. *Mathematics of Operations Research*, 23(3):640–648, August 1998.

[Avi83] David Avis. A survey of heuristics for the weighted matching problem. *Networks*, 13:475–493, 1983.

[BB99] Yannis Bakos and Erik Brynjolfsson. Bundling information goods: Pricing, profits, and efficiency. *Management Science*, 45(12):1613–1630, 1999.

[BB00] Yannis Bakos and Erik Brynjolfsson. Bundling and competition on the internet. *Marketing Science*, 19(1):63–82, 2000.

[BGFvS09] Rena Bakhshi, Daniela Gavidia, Wan Fokkink, and Maarten van Steen. An analytical model of information dissemination for a gossip-based protocol. *Comput. Netw.*, 53(13):2288–2303, August 2009.

[Blu90] Norbert Blum. A new approach to maximum matching in general graphs. *Automata, Languages and Programming*, 443:586–597, 1990.

[BMUN09] Jack Brimberg, Nenad Mladenovic, Dragan Urosevic, and Eric Ngai. Variable neighborhood search for the heaviest k-subgraph. *Computers & Operations Research*, 36(11):2885 – 2891, 2009.

[CH90] Kenneth Ward Church and Patrick Hanks. Word association norms, mutual information, and lexicography. *Computational Linguistics*, 16(1):22–29, 1990.

[CH99] Barun Chandra and Magnús Halldórsson. Greedy local improvement and weighted set packing approximation. In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '99, pages 169–176, Philadelphia, PA, USA, 1999. Society for Industrial and Applied Mathematics.

[CH07] A. Czygrinow and M. Hańćkowiak. Distributed approximations for packing in unit-disk graphs. *Distributed Computing*, 4731:152–164, 2007.

[Cha02] Patrali Chatterjee. Interfirm alliances in online retailing. *Journal of Business Research*, 57(7):714–723, 2002.

[CHP82] G. Cornujols, D. Hartvigsen, and W. Pulleyblank. Packing subgraphs in a graph. *Operations Research Letters*, 1(4):139 – 143, September 1982.

[CHW08] Andrzej Czygrinow, Michal Hańćkowiak, and Wojciech Wawrzyniak. Distributed packing in planar graphs. In *Proceedings of the twentieth annual symposium on Parallelism in algorithms and architectures*, SPAA '08, pages 55–61, New York, NY, USA, 2008. ACM.

[CK98] Pierluigi Crescenzi and Viggo Kann. A compedium of np optimization problems. http://www.nada.kth.se/theory/compedium, 1998.

[CvS10] Anna Chmielowiec and Maarten van Steen. Optimal decentralized formation of k-member partnerships. In *Fourth IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pages 154–163, Sept/Oct 2010.

[dABR92] Saul de Amorim, Jean-Pierre Barthlemy, and Celso Ribeiro. Clustering and clique partitioning: Simulated annealing and tabu search approaches. *Journal of Classification*, 9:17–41, 1992.

[DBHE08] Elena Delgado-Ballester and Miguel Hernández-Espallardo. Building online brands through brand alliances in internet. *European Journal of Marketing*, 42(9/10):954–976, 2008.

[DBOB00] Robert Davis, Margo Buchanan-Oliver, and Roderick J Brodie. Retail service branding in electronic-commerce environments. *Journal of Service Research*, 3(2):178–186, 2000.

[Dij74] Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Communications of the ACM*, 17(11):643–644, November 1974.

[Dij76] Edsger W. Dijkstra. *A Discipline of Programming*. Prentice Hall, 1976.

[DJP08] U. Dorndorf, F. Jaehn, and E. Pesch. Modelling robust flight-gate scheduling as a clique partitioning problem. *Transportation Science*, 42(3):292–301, 2008.

[DJP12] Ulrich Dorndorf, Florian Jaehn, and Erwin Pesch. Flight gate scheduling with respect to a reference schedule. *Annals of Operations Research*, 194:177–187, 2012. 10.1007/s10479-010-0809-8.

[Dol00] Shlomi Dolev. *Self-Stabilization*. The MIT Press, 2000.

[DP94] U. Dorndorf and E. Pesch. Fast clustering algorithms. *ORSA Journal on Computing*, 6:141–141, 1994.

[Edm65] Jack Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

[Edm70] Jack Edmonds. Matching: A well-solved class of integer linear programs. *Combinatorial Structures and Their Applications*, pages 89–92, 1970.

[For10a] Fortune. Fortune 500. http://money.cnn.com/magazines/fortune/fortune500/2010/full_list/, May 2010. Issue date: May 3, 2010.

[For10b] Fortune. Global 500. http://money.cnn.com/magazines/fortune/global500/2010/full_list/, July 2010. Issue date: July 26, 2010.

[Gab90] Harold N. Gabow. 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*, pages 434–443, Philadelphia, PA, USA, 1990. Society for Industrial and Applied Mathematics.

[GP10a] Michel Gendreau and Jean-Yves Potvin, editors. *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*. Springer, 2010.

[GP10b] G. Georgiadis and M. Papatriantafilou. Overlays with preferences: Approximation algorithms for matching with preference lists. In *Parallel Distributed Processing (IPDPS), 2010 IEEE International Symposium on*, pages 1 –10, april 2010.

[GP12] Giorgos Georgiadis and Marina Papatriantafilou. Adaptive distributed b-matching in overlays with preferences. *Experimental Algorithms*, 7276:208–223, 2012.

[GT91] H.N. Gabow and R.E. Tarjan. Faster scaling algorithms for general graph matching problems. *Journal of the ACM (JACM)*, 38(4):815–853, 1991.

[GW89] M. Grtschel and Y. Wakabayashi. A cutting plane algorithm for a clustering problem. *Mathematical Programming*, 45:59–96, 1989.

[HH06] Arik Hesseldahl and Stanley Holmes. Apple and nike, running mates. online, May 2006. BloombergBusinessweek Technology.

[HK84] P. Hell and D. G. Kirkpatrick. Packings by cliques and by finite families of graphs. *Discrete Mathematics*, 49(1):45–59, March 1984.

[HKNP05] P. Hell, S. Klein, L.T. Nogueira, and F. Protti. Packing r-cliques in weighted chordal graphs. *Annals of Operations Research*, 138:179–187, 2005.

[HMMP10] P. Hansen, N. Mladenović, and J.A. Moreno Pérez. Variable neighbourhood search: methods and applications. *Annals of Operations Research*, 175(1):367–407, March 2010.

[Hoe04] Jaap-Henk Hoepman. Simple distributed weighted matchings, 2004. `http://arxiv.org/abs/cs/0410047`.

[HR06a] Refael Hassin and Shlomi Rubinstein. An approximation algorithm for maximum triangle packing. *Discrete Applied Mathematics*, 154(6):971–979, April 2006.

[HR06b] Refael Hassin and Shlomi Rubinstein. Erratum to "an approximation algorithm for maximum triangle packing": [discrete applied mathematics 154 (2006) 971–979]. *Discrete Applied Mathematics*, 154(18):2620–2620, 2006.

[HS89] C.A.J. Hurkens and A. Schrijver. On the size of systems of sets every t of which have an sdr, with an application to the worst-case ratio of heuristics for packing problems. *SIAM Journal on Discrete Mathematics*, 2(1):68–72, 1989.

[HV07] Eric van Heck and Peter Vervest. Smart business networks: how the network wins. *Commun. ACM*, 50(6):28–37, June 2007.

[IS07] R.W. Irving and S. Scott. The stable fixtures problem–a many-to-many extension of stable roommates. *Discrete Applied Mathematics*, 155(16):2118–2129, 2007.

[JMB09] Márk Jelasity, Alberto Montresor, and Ozalp Babaoglu. T-man: Gossip-based fast overlay topology construction. *Comput. Netw.*, 53(13):2321–2339, 2009.

[Kan94] Viggo Kann. Maximum bounded h-matching is max snp-complete. *Information Processing Letters*, 49(6):309–318, March 1994.

[Kel93] Kevin Lane Keller. Conceptualizing, measuring, and managing consumer-based brand equity. *Journal of Marketing*, 57(1):1–22, January 1993.

[Kel99] Kevin Kelly. *New Rules for the New Economy*. Penguin Books, 1999.

[KGAW05] Gary Kochenberger, Fred Glover, Bahram Alidaee, and Haibo Wang. Clustering of microarray data via clique partitioning. *Journal of Combinatorial Optimization*, 10:77–92, 2005. 10.1007/s10878-005-1861-1.

[KH78] David G. Kirkpatrick and Pavol Hell. On the completeness of a generalized matching problem. In *STOC '78 Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 240–245, 1978.

[KMW06] Fabian Kuhn, Thomas Moscibroda, and Roger Wattenhofer. The price of being near-sighted. In *Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm*, SODA '06, pages 980–989, New York, NY, USA, 2006. ACM.

[KvS07] Anne-Marie Kermarrec and Maarten van Steen. Gossiping in distributed systems. *SIGOPS Oper. Syst. Rev.*, 41(5):2–7, October 2007.

[KW05] Fabian Kuhn and Roger Wattenhofer. Constant-time distributed dominating set approximation. *Distributed Computing*, 17:303–310, 2005. 10.1007/s00446-004-0112-5.

[KY09] Christos Koufogiannakis and Neal E. Young. Distributed fractional packing and maximum weighted b-matching via tail-recursive duality. In *Proceedings of the 23rd international conference on Distributed computing*, DISC'09, pages 221–238, Berlin, Heidelberg, 2009. Springer-Verlag.

[KY11] C. Koufogiannakis and N.E. Young. Distributed algorithms for covering, packing and maximum weighted matching. *Distributed Computing*, 24(1):45–63, 2011.

[LCDL96] Aron M. Levin, J. Charlene Davis, and Irwin Levin. Theoretical and empirical linkages between consumers' responses to different branding strategies. *Advances in Consumer Research*, 23(1):296 – 300, 1996.

[LGH04] Barbara A Lafferty, Ronald E Goldsmith, and G Tomas M Hult. The impact of the alliance on the partners: A look at cause–brand alliances. *Psychology and Marketing*, 21(7):509–531, 2004.

[LKS03] Lance Leuthesser, Chiranjeev Kohli, and Rajneesh Suri. 2+2=5? a framework for using co-branding to leverage a brand. *Journal of Brand Management*, 11(1):35–47, 2003.

[LPSP08] Zvi Lotker, Boaz Patt-Shamir, and Seth Pettie. Improved distributed approximate matching. In *SPAA '08: Proceedings of the twentieth annual symposium on Parallelism in algorithms and architectures*, pages 129–136, New York, NY, USA, 2008. ACM.

[Mat08] F. Mathieu. Self-stabilization in preference-based systems. *Peer-to-Peer Networking and Applications*, 1(2):104–121, 2008.

[MHS99] M. Müller-Hannemann and A. Schwartz. Implementing weighted b-matching algorithms: towards a flexible software design. *J. Exp. Algorithmics*, 4:1–28, December 1999.

[MJ09] Alberto Montresor and Márk Jelasity. PeerSim: A scalable P2P simulator. In *Proc. of the 9th Int. Conference on Peer-to-Peer (P2P'09)*, pages 99–100, Seattle, WA, September 2009.

[MM07] Fredrik Manne and Morten Mjelde. A self-stabilizing weighted matching algorithm. In *Stabilization, Safety, and Security of Distributed Systems*, volume 4838/2007 of *LNCS*, pages 383–393, 2007.

[Moo93] James F. Moore. Predators and prey: A new ecology of competition. *Harvard Business Review*, 71(3):75–86, 1993.

[MV80] Silvio Micali and Vijay V. Vazirani. An $o(\sqrt{(}|v|)|e|)$ algorithm for finding maximum matching in general graphs. In *SFCS '80: Proceedings of the 21st Annual Symposium on Foundations of Computer Science*, pages 17–27, Washington, DC, USA, 1980. IEEE Computer Society.

[NB97] B.J. Nalebuff and A.M. Brandenburger. Co-opetition: Competitive and cooperative business strategies for the digital economy. *Strategy & leadership*, 25(6):28–35, 1997.

[Nie08] Tim Nieberg. Local, distributed weighted matching on general and wireless topologies. In *DIAL M-POMC '08: Proceedings of the fifth international workshop on Foundations of mobile computing*, pages 87–92, New York, NY, USA, 2008. ACM.

[Orl12] Kyle Orland. Humble thq bundle threatens to ruin the brand's reputation (updated), November 2012.

[PJS96] C Whan Park, Sung Youl Jun, and Allan D Shocker. Composite branding alliances: an investigation of extension and feedback effects. *Journal of Marketing Research*, 33:453–466, 1996.

[Pre99] Robert Preis. Linear time 1/2-approximation algorithm for maximum weighted matching in general graphs. In *in General Graphs, Symposium on Theoretical Aspects of Computer Science, STACS 99*, volume 1563/1999 of *LNCS*, pages 259–269. Springer, 1999.

[PS10] Alessandro Panconesi and Mauro Sozio. Fast primal-dual distributed algorithms for scheduling and matching problems. *Distributed Computing*, 22:269–283, 2010. 10.1007/s00446-010-0100-x.

[RQR99] Akshay R Rao, Lu Qu, and Robert W Ruekert. Signaling unobservable product quality through a brand ally. *Journal of Marketing Research*, 36:258–268, 1999.

[RR94] A.R. Rao and R.W. Ruekert. Brand alliances as signals of product quality. *Sloan Management Review*, 36:87–97, 1994.

[SA11] Travis Service and Julie Adams. Coalition formation for task allocation: theory and algorithms. *Autonomous Agents and Multi-Agent Systems*, 22:225–248, 2011.

[Sch93] Marco Schneider. Self-stabilization. *ACM Comput. Surv.*, 25:45–67, March 1993.

[SK98] Onn Shehory and Sarit Kraus. Methods for task allocation via agent coalition formation. *Artificial Intelligence*, 101:165–200, 1998.

[SPNW06] K. Sun, P. Peng, P. Ning, and C. Wang. Secure distributed cluster formation in wireless sensor networks. In *Computer Security Applications Conference, 2006. ACSAC'06. 22nd Annual*, pages 131–140. IEEE, 2006.

[SR98] Bernard L. Simonin and Julie A. Ruth. Is a company known by the company it keeps? assessing the spillover effects of brand alliances on consumer brand attitudes. *JMR, Journal of Marketing Research*, 35(1):30–42, 1998.

[TA05] Predrag Tošić and Gul Agha. Maximal clique based distributed coalition formation for task allocation in large-scale multi-agent systems. In Toru Ishida, Les Gasser, and Hideyuki Nakashima, editors, *Massively Multi-Agent Systems I*, volume 3446 of *Lecture Notes in Computer Science*, pages 573–573. Springer Berlin / Heidelberg, 2005.

[Tel00] Gerard Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, 2nd edition, 2000.

[TH11] Volker Turau and Bernd Hauck. A new analysis of a self-stabilizing maximum weight matching algorithm with approximation ratio 2. *Theor. Comput. Sci.*, 412:5527–5540, September 2011.

[TTL00] Don Tapscott, David Ticoll, and Alex Lowy. *Digital capital : harnessing the power of business webs*. Harvard Business School Press, Boston, Mass., USA, 2000.

[Uya09] Ahmet Uyar. Investigation of the accuracy of search engine hit counts. *Journal of Information Science*, 35(4):469–480, 2009.

[VA06] L. Vig and J.A. Adams. Multi-robot coalition formation. *Robotics, IEEE Transactions on*, 22(4):637 –649, aug. 2006.

[VA07] Lovekesh Vig and Julie Adams. Coalition formation: From software agents to robots. *Journal of Intelligent & Robotic Systems*, 50:85–118, 2007. 10.1007/s10846-007-9150-0.

[VB06] Ivar Vermeulen and Jeroen Bruggeman. Automated longitudinal data collecton on the web: Competition between search engines, 1993-2000. In *ASSR Working Paper*, volume 6. 2006.

[Ver07] Ivar E. Vermeulen. Matchmaking in cyberspace: An application of web-based brand image measurement. In F. Costa Pereira and J. Verissimo, editors, *Proceedings of the International Conference of Research in Advertising*, Lisbon, 2007. Universidade de Lisboa.

[VGvS05] Spyros Voulgaris, Daniela Gavidia, and Maarten van Steen. Cyclon: Inexpensive membership management for unstructured p2p overlays. *Springer Journal of Network and Systems Management*, 13(2):197–217, Jun 2005.

[VM09] R Venkatesh and Vijay Mahajan. *Handbook of Pricing Research in Marketing*, chapter 11 The design and pricing of bundles: a review of normative guidelines and practical approaches, pages 232–257. Edward Elgar Publishing, 2009.

[Vou06] Spyros Voulgaris. *Epidemic-Based Self-Organization in Peer-to-Peer Systems*. PhD thesis, Vrije Universiteit, Amsterdam, The Netherlands, 2006.

[VvSI07] Spyros Voulgaris, Maarten van Steen, and Konrad Iwanicki. Proactive gossip-based management of semantic overlay networks. *Concurrency and Computation: Practice and Experience*, 19(17):2299–2311, 2007. Special Issue: Parallel and Distributed Computing (EuroPar 2005).

[WAGK06] Haibo Wang, Bahram Alidaee, Fred Glover, and Gary Kochenberger. Solving group technology problems via clique partitioning. *International Journal of Flexible Manufacturing Systems*, 18:77–97, 2006. 10.1007/s10696-006-9011-3.

[WTP04] J.H. Washburn, B.D. Till, and R. Priluck. Brand alliance and customer-based brand-equity effects. *Psychology & Marketing*, 21(7):487–508, 2004.

[Yus07] Raphael Yuster. Combinatorial and computational aspects of graph packing and graph decomposition. *Computer Science Review*, 1(1):12 – 26, 2007.

SIKS Dissertation Series

2009-01 Rasa Jurgelenaite (RUN). Symmetric Causal Independence Models

2009-02 Willem Robert van Hage (VU). Evaluating Ontology-Alignment Techniques

2009-03 Hans Stol (UvT). A Framework for Evidence-based Policy Making Using IT

2009-04 Josephine Nabukenya (RUN). Improving the Quality of Organisational Policy Making using Collaboration Engineering

2009-05 Sietse Overbeek (RUN). Bridging Supply and Demand for Knowledge Intensive Tasks - Based on Knowledge, Cognition, and Quality

2009-06 Muhammad Subianto (UU). Understanding Classification

2009-07 Ronald Poppe (UT). Discriminative Vision-Based Recovery and Recognition of Human Motion

2009-08 Volker Nannen (VU). Evolutionary Agent-Based Policy Analysis in Dynamic Environments

2009-09 Benjamin Kanagwa (RUN). Design, Discovery and Construction of Service-oriented Systems

2009-10 Jan Wielemaker (UvA). Logic programming for knowledge-intensive interactive applications

2009-11 Alexander Boer (UvA). Legal Theory, Sources of Law & the Semantic Web

2009-12 Peter Massuthe (TUE, Humboldt-Universitaet zu Berlin). Operating Guidelines for Services

2009-13 Steven de Jong (UM). Fairness in Multi-Agent Systems

2009-14 Maksym Korotkiy (VU). From ontology-enabled services to service-enabled ontologies (making ontologies work in e-science with ONTO-SOA).

2009-15 Rinke Hoekstra (UvA). Ontology Representation - Design Patterns and Ontologies that Make Sense

2009-16 Fritz Reul (UvT). New Architectures in Computer Chess

2009-17 Laurens van der Maaten (UvT). Feature Extraction from Visual Data

2009-18 Fabian Groffen (CWI). Armada, An Evolving Database System

2009-19 Valentin Robu (CWI). Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets

2009-20 Bob van der Vecht (UU). Adjustable Autonomy: Controling Influences on Decision Making

2009-21 Stijn Vanderlooy (UM). Ranking and Reliable Classification

2009-22 Pavel Serdyukov (UT). Search For Expertise: Going beyond direct evidence

2009-23 Peter Hofgesang (VU). Modelling Web Usage in a Changing Environment

2009-24 Annerieke Heuvelink (VUA). Cognitive Models for Training Simulations

2009-25 Alex van Ballegooij (CWI). ”RAM: Array Database Management through Relational Mapping”

2009-26 Fernando Koch (UU). An Agent-Based Model for the Development of Intelligent Mobile Services

2009-27 Christian Glahn (OU). Contextual Support of social Engagement and Reflection on the Web

2009-28 Sander Evers (UT). Sensor Data Management with Probabilistic Models

2009-29 Stanislav Pokraev (UT). Model-Driven Semantic Integration of Service-Oriented Applications

2009-30 Marcin Zukowski (CWI). Balancing vectorized query execution with bandwidth-optimized storage

2009-31 Sofiya Katrenko (UvA). A Closer Look at Learning Relations from Text

2009-32 Rik Farenhorst (VU) and Remco de Boer (VU). Architectural Knowledge Management: Supporting Architects and Auditors

2009-33 Khiet Truong (UT). How Does Real Affect Affect Affect Recognition In Speech?

2009-34 Inge van de Weerd (UU). Advancing in Software Product Management: An Incremental Method Engineering Approach

2009-35 Wouter Koelewijn (UL). Privacy en Politiegegevens; Over geautomatiseerde normatieve informatie-uitwisseling

2009-36 Marco Kalz (OUN). Placement Support for Learners in Learning Networks

2009-37 Hendrik Drachsler (OUN). Navigation Support for Learners in Informal Learning Networks

2009-38 Riina Vuorikari (OU). Tags and self-organisation: a metadata ecology for learning resources in a multilingual context

2009-39 Christian Stahl (TUE, Humboldt-Universitaet zu Berlin). Service Substitution – A Behavioral Approach Based on Petri Nets

2009-40 Stephan Raaijmakers (UvT). Multinomial Language Learning: Investigations into the Geometry of Language

2009-41 Igor Berezhnyy (UvT). Digital Analysis of Paintings

2009-42 Toine Bogers (UvT). Recommender Systems for Social Bookmarking

2009-43 Virginia Nunes Leal Franqueira (UT). Finding Multi-step Attacks in Computer Networks using Heuristic Search and Mobile Ambients

2009-44 Roberto Santana Tapia (UT). Assessing Business-IT Alignment in Networked Organizations

2009-45 Jilles Vreeken (UU). Making Pattern Mining Useful

2009-46 Loredana Afanasiev (UvA). Querying XML: Benchmarks and Recursion

2010-01 Matthijs van Leeuwen (UU). Patterns that Matter

2010-02 Ingo Wassink (UT). Work flows in Life Science

2010-03 Joost Geurts (CWI). A Document Engineering Model and Processing Framework for Multimedia documents

2010-04 Olga Kulyk (UT). Do You Know What I Know? Situational Awareness of Co-located Teams in Multidisplay Environments

2010-05 Claudia Hauff (UT). Predicting the Effectiveness of Queries and Retrieval Systems

2010-06 Sander Bakkes (UvT). Rapid Adaptation of Video Game AI

2010-07 Wim Fikkert (UT). Gesture interaction at a Distance

2010-08 Krzysztof Siewicz (UL). Towards an Improved Regulatory Framework of Free Software. Protecting user freedoms in a world of software communities and eGovernments

2010-09 Hugo Kielman (UL). A Politiele gegevensverwerking en Privacy, Naar een effectieve waarborging

2010-10 Rebecca Ong (UL). Mobile Communication and Protection of Children

2010-11 Adriaan Ter Mors (TUD). The world according to MARP: Multi-Agent Route Planning

2010-12 Susan van den Braak (UU). Sensemaking software for crime analysis

2010-13 Gianluigi Folino (RUN). High Performance Data Mining using Bio-inspired techniques

2010-14 Sander van Splunter (VU). Automated Web Service Reconfiguration

2010-15 Lianne Bodenstaff (UT). Managing Dependency Relations in Inter-Organizational Models

2010-16 Sicco Verwer (TUD). Efficient Identification of Timed Automata, theory and practice

2010-17 Spyros Kotoulas (VU). Scalable Discovery of Networked Resources: Algorithms, Infrastructure, Applications

2010-18 Charlotte Gerritsen (VU). Caught in the Act: Investigating Crime by Agent-Based Simulation

2010-19 Henriette Cramer (UvA). People's Responses to Autonomous and Adaptive Systems

2010-20 Ivo Swartjes (UT). Whose Story Is It Anyway? How Improv Informs Agency and Authorship of Emergent Narrative

2010-21 Harold van Heerde (UT). Privacy-aware data management by means of data degradation

2010-22 Michiel Hildebrand (CWI). End-user Support for Access to Heterogeneous Linked Data

2010-23 Bas Steunebrink (UU). The Logical Structure of Emotions

2010-24 Dmytro Tykhonov Designing Generic and Efficient Negotiation Strategies

2010-25 Zulfiqar Ali Memon (VU). Modelling Human-Awareness for Ambient Agents: A Human Mindreading Perspective

2010-26 Ying Zhang (CWI). XRPC: Efficient Distributed Query Processing on Heterogeneous XQuery Engines

2010-27 Marten Voulon (UL). Automatisch contracteren

2010-28 Arne Koopman (UU). Characteristic Relational Patterns

2010-29 Stratos Idreos (CWI). Database Cracking: Towards Auto-tuning Database Kernels

2010-30 Marieke van Erp (UvT). Accessing Natural History - Discoveries in data cleaning, structuring, and retrieval

2010-31 Victor de Boer (UvA). Ontology Enrichment from Heterogeneous Sources on the Web

2010-32 Marcel Hiel (UvT). An Adaptive Service Oriented Architecture: Automatically solving Interoperability Problems

2010-33 Robin Aly (UT). Modeling Representation Uncertainty in Concept-Based Multimedia Retrieval

2010-34 Teduh Dirgahayu (UT). Interaction Design in Service Compositions

2010-35 Dolf Trieschnigg (UT). Proof of Concept: Concept-based Biomedical Information Retrieval

2010-36 Jose Janssen (OU). Paving the Way for Lifelong Learning; Facilitating competence development through a learning path specification

2010-37 Niels Lohmann (TUE). Correctness of services and their composition

2010-38 Dirk Fahland (TUE). From Scenarios to components

2010-39 Ghazanfar Farooq Siddiqui (VU). Integrative modeling of emotions in virtual agents

2010-40 Mark van Assem (VU). Converting and Integrating Vocabularies for the Semantic Web

2010-41 Guillaume Chaslot (UM). Monte-Carlo Tree Search

2010-42 Sybren de Kinderen (VU). Needs-driven service bundling in a multi-supplier setting - the computational e3-service approach

2010-43 Peter van Kranenburg (UU). A Computational Approach to Content-Based Retrieval of Folk Song Melodies

2010-44 Pieter Bellekens (TUE). An Approach towards Context-sensitive and User-adapted Access to Heterogeneous Data Sources, Illustrated in the Television Domain

2010-45 Vasilios Andrikopoulos (UvT). A theory and model for the evolution of software services

2010-46 Vincent Pijpers (VU). e3alignment: Exploring Inter-Organizational Business-ICT Alignment

2010-47 Chen Li (UT). Mining Process Model Variants: Challenges, Techniques, Examples

2010-48 Withdrawn

2010-49 Jahn-Takeshi Saito (UM). Solving difficult game positions

2010-50 Bouke Huurnink (UvA). Search in Audiovisual Broadcast Archives

2010-51 Alia Khairia Amin (CWI). Understanding and supporting information seeking tasks in multiple sources

2010-52 Peter-Paul van Maanen (VU). Adaptive Support for Human-Computer Teams: Exploring the Use of Cognitive Models of Trust and Attention

2010-53 Edgar Meij (UvA). Combining Concepts and Language Models for Information Access

2011-01 Botond Cseke (RUN). Variational Algorithms for Bayesian Inference in Latent Gaussian Models

2011-02 Nick Tinnemeier (UU). Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language

2011-03 Jan Martijn van der Werf (TUE). Compositional Design and Verification of Component-Based Information Systems

2011-04 Hado van Hasselt (UU). Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference learning algorithms

2011-05 Base van der Raadt (VU). Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline.

2011-06 Yiwen Wang (TUE). Semantically-Enhanced Recommendations in Cultural Heritage

2011-07 Yujia Cao (UT). Multimodal Information Presentation for High Load Human Computer Interaction

2011-08 Nieske Vergunst (UU). BDI-based Generation of Robust Task-Oriented Dialogues

2011-09 Tim de Jong (OU). Contextualised Mobile Media for Learning

2011-10 Bart Bogaert (UvT). Cloud Content Contention

2011-11 Dhaval Vyas (UT). Designing for Awareness: An Experience-focused HCI Perspective

2011-12 Carmen Bratosin (TUE). Grid Architecture for Distributed Process Mining

2011-13 Xiaoyu Mao (UvT). Airport under Control. Multiagent Scheduling for Airport Ground Handling

2011-14 Milan Lovric (EUR). Behavioral Finance and Agent-Based Artificial Markets

2011-15 Marijn Koolen (UvA). The Meaning of Structure: the Value of Link Evidence for Information Retrieval

2011-16 Maarten Schadd (UM). Selective Search in Games of Different Complexity

2011-17 Jiyin He (UvA). Exploring Topic Structure: Coherence, Diversity and Relatedness

2011-18 Mark Ponsen (UM). Strategic Decision-Making in complex games

2011-19 Ellen Rusman (OU). The Mind ' s Eye on Personal Profiles

2011-20 Qing Gu (VU). Guiding service-oriented software engineering - A view-based approach

2011-21 Linda Terlouw (TUD). Modularization and Specification of Service-Oriented Systems

2011-22 Junte Zhang (UvA). System Evaluation of Archival Description and Access

2011-23 Wouter Weerkamp (UvA). Finding People and their Utterances in Social Media

2011-24 Herwin van Welbergen (UT). Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior

2011-25 Syed Waqar ul Qounain Jaffry (VU). Analysis and Validation of Models for Trust Dynamics

2011-26 Matthijs Aart Pontier (VU). Virtual Agents for Human Communication - Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots

2011-27 Aniel Bhulai (VU). Dynamic website optimization through autonomous management of design patterns

2011-28 Rianne Kaptein (UvA). Effective Focused Retrieval by Exploiting Query Context and Document Structure

2011-29 Faisal Kamiran (TUE). Discrimination-aware Classification

2011-30 Egon van den Broek (UT). Affective Signal Processing (ASP): Unraveling the mystery of emotions

2011-31 Ludo Waltman (EUR). Computational and Game-Theoretic Approaches for Modeling Bounded Rationality

2011-32 Nees-Jan van Eck (EUR). Methodological Advances in Bibliometric Mapping of Science

2011-33 Tom van der Weide (UU). Arguing to Motivate Decisions

2011-34 Paolo Turrini (UU). Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations

2011-35 Maaike Harbers (UU). Explaining Agent Behavior in Virtual Training

2011-36 Erik van der Spek (UU). Experiments in serious game design: a cognitive approach

2011-37 Adriana Burlutiu (RUN). Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference

2011-38 Nyree Lemmens (UM). Bee-inspired Distributed Optimization

2011-39 Joost Westra (UU). Organizing Adaptation using Agents in Serious Games

2011-40 Viktor Clerc (VU). Architectural Knowledge Management in Global Software Development

2011-41 Luan Ibraimi (UT). Cryptographically Enforced Distributed Data Access Control

2011-42 Michal Sindlar (UU). Explaining Behavior through Mental State Attribution

2011-43 Henk van der Schuur (UU). Process Improvement through Software Operation Knowledge

2011-44 Boris Reuderink (UT). Robust Brain-Computer Interfaces

2011-45 Herman Stehouwer (UvT). Statistical Language Models for Alternative Sequence Selection

2011-46 Beibei Hu (TUD). Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work

2011-47 Azizi Bin Ab Aziz (VU). Exploring Computational Models for Intelligent Support of Persons with Depression

2011-48 Mark Ter Maat (UT). Response Selection and Turn-taking for a Sensitive Artificial Listening Agent

2011-49 Andreea Niculescu (UT). Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality

2012-01 Terry Kakeeto (UvT). Relationship Marketing for SMEs in Uganda

2012-02 Muhammad Umair (VU). Adaptivity, emotion, and Rationality in Human and Ambient Agent Models

2012-03 Adam Vanya (VU). Supporting Architecture Evolution by Mining Software Repositories

2012-04 Jurriaan Souer (UU). Development of Content Management System-based Web Applications

2012-05 Marijn Plomp (UU). Maturing Interorganisational Information Systems

2012-06 Wolfgang Reinhardt (OU). Awareness Support for Knowledge Workers in Research Networks

2012-07 Rianne van Lambalgen (VU). When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions

2012-08 Gerben de Vries (UvA). Kernel Methods for Vessel Trajectories

2012-09 Ricardo Neisse (UT). Trust and Privacy Management Support for Context-Aware Service Platforms

2012-10 David Smits (TUE). Towards a Generic Distributed Adaptive Hypermedia Environment

2012-11 J.C.B. Rantham Prabhakara (TUE). Process Mining in the Large: Preprocessing, Discovery, and Diagnostics

2012-12 Kees van der Sluijs (TUE). Model Driven Design and Data Integration in Semantic Web Information Systems

2012-13 Suleman Shahid (UvT). Fun and Face: Exploring non-verbal expressions of emotion during playful interactions

2012-14 Evgeny Knutov (TUE). Generic Adaptation Framework for Unifying Adaptive Web-based Systems

2012-15 Natalie van der Wal (VU). Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes.

2012-16 Fiemke Both (VU). Helping people by understanding them - Ambient Agents supporting task execution and depression treatment

2012-17 Amal Elgammal (UvT). Towards a Comprehensive Framework for Business Process Compliance

2012-18 Eltjo Poort (VU). Improving Solution Architecting Practices

2012-19 Helen Schonenberg (TUE). What's Next? Operational Support for Business Process Execution

2012-20 Ali Bahramisharif (RUN). Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing

2012-21 Roberto Cornacchia (TUD). Querying Sparse Matrices for Information Retrieval

2012-22 Thijs Vis (UvT). Intelligence, politie en veiligheidsdienst: verenigbare grootheden?

2012-23 Christian Muehl (UT). Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction

2012-24 Laurens van der Werff (UT). Evaluation of Noisy Transcripts for Spoken Document Retrieval

2012-25 Silja Eckartz (UT). Managing the Business Case Development in Inter-Organizational IT Projects: A Methodology and its Application

2012-26 Emile de Maat (UvA). Making Sense of Legal Text

2012-27 Hayrettin Gurkok (UT). Mind the Sheep! User Experience Evaluation & Brain-Computer Interface Games

2012-28 Nancy Pascall (UvT). Engendering Technology Empowering Women

2012-29 Almer Tigelaar (UT). Peer-to-Peer Information Retrieval

2012-30 Alina Pommeranz (TUD). Designing Human-Centered Systems for Reflective Decision Making

2012-31 Emily Bagarukayo (RUN). A Learning by Construction Approach for Higher Order Cognitive Skills Improvement, Building Capacity and Infrastructure

2012-32 Wietske Visser (TUD). Qualitative multi-criteria preference representation and reasoning

2012-33 Rory Sie (OUN). Coalitions in Cooperation Networks (COCOON).

2012-34 Pavol Jancura (RUN). Evolutionary analysis in PPI networks and applications

2012-35 Evert Haasdijk (VU). Never Too Old To Learn – On-line Evolution of Controllers in Swarm- and Modular Robotics

2012-36 Denis Ssebugwawo (RUN). Analysis and Evaluation of Collaborative Modeling Processes

2012-37 Agnes Nakakawa (RUN). A Collaboration Process for Enterprise Architecture Creation

2012-38 Selmar Smit (VU). Parameter Tuning and Scientific Testing in Evolutionary Algorithms

2012-39 Hassan Fatemi (UT). Risk-aware design of value and coordination networks

2012-40 Agus Gunawan (UvT). Information Access for SMEs in Indonesia

2012-41 Sebastian Kelle (OU). Game Design Patterns for Learning

2012-42 Dominique Verpoorten (OU). Reflection Amplifiers in self-regulated Learning

2012-43 Withdrawn

2012-44 Anna Tordai (VU). On Combining Alignment Techniques

2012-45 Benedikt Kratz (UvT). A Model and Language for Business-aware Transactions

2012-46 Simon Carter (UvA). Exploration and Exploitation of Multilingual Data for Statistical Machine Translation

2012-47 Manos Tsagkias (UvA). Mining Social Media: Tracking Content and Predicting Behavior

2012-48 Jorn Bakker (TUE). Handling Abrupt Changes in Evolving Time-series Data

2012-49 Michael Kaisers (UM). Learning against Learning - Evolutionary dynamics of reinforcement learning algorithms in strategic interactions

2012-50 Steven van Kervel (TUD). Ontologogy driven Enterprise Information Systems Engineering

2012-51 Jeroen de Jong (TUD). Heuristics in Dynamic Sceduling; a practical framework with a case study in elevator dispatching

2013-01 Viorel Milea (EUR). News Analytics for Financial Decision Support

2013-02 Erietta Liarou (CWI). MonetDB/DataCell: Leveraging the Column-store Database Technology for Efficient and Scalable Stream Processing

2013-03 Szymon Klarman (VU). Reasoning with Contexts in Description Logics

2013-04 Chetan Yadati (TUD). Coordinating autonomous planning and scheduling

2013-05 Dulce Pumareja (UT). Groupware Requirements Evolutions Patterns

2013-06 Romulo Goncalves (CWI). The Data Cyclotron: Juggling Data and Queries for a Data Warehouse Audience

2013-07 Giel van Lankveld (UvT). Quantifying Individual Player Differences

2013-08 Robbert-Jan Merk (VU). Making enemies: cognitive modeling for opponent agents in fighter pilot simulators

2013-09 Fabio Gori (RUN). Metagenomic Data Analysis: Computational Methods and Applications

2013-10 Jeewanie Jayasinghe Arachchige (UvT). A Unified Modeling Framework for Service Design.

2013-11 Evangelos Pournaras (TUD). Multi-level Reconfigurable Self-organization in Overlay Services

2013-12 Marian Razavian (VU). Knowledge-driven Migration to Services

2013-13 Mohammad Safiri (UT). Service Tailoring: User-centric creation of integrated IT-based homecare services to support independent living of elderly

2013-14 Jafar Tanha (UvA). Ensemble Approaches to Semi-Supervised Learning Learning

2013-15 Daniel Hennes (UM). Multiagent Learning - Dynamic Games and Applications

2013-16 Eric Kok (UU). Exploring the practical benefits of argumentation in multi-agent deliberation

2013-17 Koen Kok (VU). The PowerMatcher: Smart Coordination for the Smart Electricity Grid

2013-18 Jeroen Janssens (UvT). Outlier Selection and One-Class Classification

2013-19 Renze Steenhuizen (TUD). Coordinated Multi-Agent Planning and Scheduling

2013-20 Katja Hofmann (UvA). Fast and Reliable Online Learning to Rank for Information Retrieval

2013-21 Sander Wubben (UvT). Text-to-text generation by monolingual machine translation

2013-22 Tom Claassen (RUN). Causal Discovery and Logic

2013-23 Patricio de Alencar Silva (UvT). Value Activity Monitoring

2013-24 Haitham Bou Ammar (UM). Automated Transfer in Reinforcement Learning

2013-25 Agnieszka Anna Latoszek-Berendsen (UM). Intention-based Decision Support. A new way of representing and implementing clinical guidelines in a Decision Support System

2013-26 Alireza Zarghami (UT). Architectural Support for Dynamic Homecare Service Provisioning

2013-27 Mohammad Huq (UT). Inference-based Framework Managing Data Provenance

2013-28 Frans van der Sluis (UT). When Complexity becomes Interesting: An Inquiry into the Information eXperience

2013-29 Iwan de Kok (UT). Listening Heads

2013-30 Joyce Nakatumba (TUE). Resource-Aware Business Process Management: Analysis and Support

2013-31 Dinh Khoa Nguyen (UvT). Blueprint Model and Language for Engineering Cloud Applications

2013-32 Kamakshi Rajagopal (OUN). Networking For Learning; The role of Networking in a Lifelong Learner's Professional Development

2013-33 Qi Gao (TUD). User Modeling and Personalization in the Microblogging Sphere

2013-34 Kien Tjin-Kam-Jet (UT). Distributed Deep Web Search

2013-35 Abdallah El Ali (UvA). Minimal Mobile Human Computer Interaction Promotor: Prof. dr. L. Hardman (CWI/UVA).

2013-36 Than Lam Hoang (TUe). Pattern Mining in Data Streams

2013-37 Dirk Börner (OUN). Ambient Learning Displays

2013-38 Eelco den Heijer (VU). Autonomous Evolutionary Art

2013-39 Joop de Jong (TUD). A Method for Enterprise Ontology based Design of Enterprise Information Systems

2013-40 Pim Nijssen (UM). Monte-Carlo Tree Search for Multi-Player Games

2013-41 Jochem Liem (UvA). Supporting the Conceptual Modelling of Dynamic Systems: A Knowledge Engineering Perspective on Qualitative Reasoning

2013-42 Léon Planken (TUD). Algorithms for Simple Temporal Reasoning

2013-43 Marc Bron (UvA). Exploration and Contextualization through Interaction and Concepts

2014-01 Nicola Barile (UU). Studies in Learning Monotone Models from Data

2014-02 Fiona Tuliyano (RUN). Combining System Dynamics with a Domain Modeling Method

2014-03 Sergio Raul Duarte Torres (UT). Information Retrieval for Children: Search Behavior and Solutions

2014-04 Hanna Jochmann-Mannak (UT). Websites for children: search strategies and interface design - Three studies on children's search performance and evaluation

2014-05 Jurriaan van Reijsen (UU). Knowledge Perspectives on Advancing Dynamic Capability

2014-06 Damian Tamburri (VU). Supporting Networked Software Development

2014-07 Arya Adriansyah (TUE). Aligning Observed and Modeled Behavior

2014-08 Samur Araujo (TUD). Data Integration over Distributed and Heterogeneous Data Endpoints

2014-09 Philip Jackson (UvT). Toward Human-Level Artificial Intelligence: Representation and Computation of Meaning in Natural Language

2014-10 Ivan Salvador Razo Zapata (VU). Service Value Networks

2014-11 Janneke van der Zwaan (TUD). An Empathic Virtual Buddy for Social Support

2014-12 Willem van Willigen (VU). Look Ma, No Hands: Aspects of Autonomous Vehicle Control

2014-13 Arlette van Wissen (VU). Agent-Based Support for Behavior Change: Models and Applications in Health and Safety Domains

2014-14 Yangyang Shi (TUD). Language Models With Meta-information

2014-15 Natalya Mogles (VU). Agent-Based Analysis and Support of Human Functioning in Complex Socio-Technical Systems: Applications in Safety and Healthcare

2014-16 Krystyna Milian (VU). Supporting trial recruitment and design by automatically interpreting eligibility criteria

2014-17 Kathrin Dentler (VU). Computing healthcare quality indicators automatically: Secondary Use of Patient Data and Semantic Interoperability

2014-18 Mattijs Ghijsen (VU). Methods and Models for the Design and Study of Dynamic Agent Organizations

2014-19 Vincius Ramos (TUE). Adaptive Hypermedia Courses: Qualitative and Quantitative Evaluation and Tool Support

2014-20 Mena Habib (UT). Named Entity Extraction and Disambiguation for Informal Text: The Missing Link

2014-21 Kassidy Clark (TUD). Negotiation and Monitoring in Open Environments

2014-22 Marieke Peeters (UT). Personalized Educational Games - Developing agent-supported scenario-based training

2014-23 Eleftherios Sidirourgos (UvA/CWI). Space Efficient Indexes for the Big Data Era

2014-24 Davide Ceolin (VU). Trusting Semi-structured Web Data

2014-25 Martijn Lappenschaar (RUN). New network models for the analysis of disease interaction

2014-26 Tim Baarslag (TUD). What to Bid and When to Stop

2014-27 Rui Jorge Almeida (EUR). Conditional Density Models Integrating Fuzzy and Probabilistic Representations of Uncertainty