On Finding Better Friends in Social Networks

Philipp Brandes[1] and Roger Wattenhofer[2]

[1] ETH Zurich
Computer Engineering and Networks Lab (TIK), Switzerland,
pbrandes@ethz.ch
[2] ETH Zurich
Computer Engineering and Networks Lab (TIK), Switzerland,
wattenhofer@ethz.ch

Abstract. We study the dynamics of a social network. Each node has to decide locally which other node it wants to befriend, i.e., to which other node it wants to create a connection in order to maximize its welfare, which is defined as the sum of the weights of incident edges. This allows us to model the cooperation between nodes where every node tries to do as well as possible. With the limitation that each node can only have a constant number of friends, we show that every local algorithm is arbitrarily worse than a globally optimal solution. Furthermore, we show that there cannot be a best local algorithm, i.e., for every local algorithm exists a social network in which the algorithm performs arbitrarily worse than some other local algorithm. However, one can combine a number of local algorithms in order to be competitive with the best of them. We also investigate a slightly different valuation variant. Nodes include another node's friends for their valuation. There are scenarios in which this does not converge to a stable state, i.e., the nodes switch friends indefinitely. We also analyze the consequences if ending a friendship permanently damages it.

Keywords: distributed algorithms, social networks, dynamic networks, local algorithms, stable states

1 Introduction

Psychologists claim that you have a limit of how many friends you can handle [8]. Consequently, you should assess your current friends, and drop those that are unsatisfactory, to make room for new ones! In this paper we study the computational side of finding friends in social networks. We assume that people can only choose new friends among their current social environment, i.e., one can only become friends with friends of friends, or more generally with acquaintances in the ℓ-hop neighborhood of the current friendship graph. If people constantly improve on their friendships with this *local* strategy, will this eventually lead to a social optimum, or at least an approximate solution? What is the best strategy to find new friends? Should one just greedily pick the best available friends? Or should one rather try to be friends with a diverse set of people, in order to profit from a larger set of possible new friends?

Not so surprisingly, we show that any local friend-finding strategy will only converge to a solution that is arbitrarily worse than a global optimum. More surprisingly however, there is no best local strategy. No matter what the strategy is, there is always a possible input scenario where other local strategies are much better. In addition we study mixing strategies, i.e., we allow everyone to use several strategies to find their friends. Additionally, we investigate slightly changed valuation models. We show that judging a friend not on his own, but also by his friends, can lead to unstable states, i.e., nodes switch friends indefinitely. We also analyze a valuation model in which breaking up a friendship reduces the valuation of the friendship permanently.

1.1 Related Work

An early ancestor of our work is the stable marriage problem, introduced by Gale and Shapley [6] in 1962: We are given n nodes, partitioned into two sets commonly denoted as men and women. Each woman has a strictly ordered preference list over all men and vice versa. They now want to create a stable matching. A matching is called stable if there is no pair of man and woman such that, instead of being matched to their current partner, they would prefer to be matched to each other. The roommate problem [6] is another related research area, where the nodes are not partitioned into two disjoint sets. Each node again has a complete, strictly ordered preference list. In this basic setting there might not be a stable matching. The problem is further investigated in [1, 7], stating restrictions to allow and find stable matchings. An overview on stable marriage can be found in e.g., [11] and a more detailed analysis of matchings in bipartite graphs in [20]. Stable marriage has also been studied as an online problem where the preferences of the men are revealed one at a time [17]. In this setting there are $\Omega(n^2)$ initially unstable marriages in the worst case.

Much research has been done in the stable marriage area on preference lists with ties [9, 16], i.e., when the constraint of strictly ordered preference lists is lifted. In our model we assume locality of information, i.e., nodes do not know their complete preference list. Furthermore, we do not require the nodes to have a strict ordering. Finding a maximum matching for stable marriage with these extensions, ties and incomplete preference lists, is known to be $\mathcal{NP}$-hard [14, 19]. It can be approximated within a factor of 2 [19]. It can also be approximated within a factor which depends on the number of ties in the preference lists [12].

There have been several approaches to solve stable marriage in a distributed way. In [10] the nodes can only try to be matched to a fixed set of adjacent nodes.

Generally related to our work are network formation games from the field of economics. The nodes create links, as a one shot game or dynamically, to generate welfare which depends on the created links. This welfare is allocated to the players according to some specific rules. These games include models of so called market sharing agreements, in which companies can agree not to sell goods on each others markets to increase their profits [4], and labor markets, where workers get jobs offered and pass the offer to one of their friends if they are already employed [5]. Another example is the model of a general buyer and seller market in which a link represents a potential transaction [18]. A survey on this area is in [15].

So far, the possible matching edges were a fixed set of edges. In [3], the nodes are partitioned into two sets, workers and firms. There are static connections between some workers which indicate friendship. The workers are matched using a local variant of the Gale-Shapley algorithm, but only to firms which are known to their friends. This introduces a dynamic set of matching partners. If a worker changes his company, this can change the set of possible matching partners for his friends. The model used by Martin Hoefer generalizes this [13]. The set of nodes is not partitioned and nodes can possibly have more than one matching partner. In his paper, Hoefer studies the convergence time of matching edges in a social network, with a limited lookahead ℓ. For $\ell = 2$ this means that the nodes only know the neighbors of their neighbors. In general, nodes can only create a connection to nodes which are in a distance of at most ℓ hops. Hoefer's model distinguishes between social links and matching links. Social links are static edges which already exist in the initial graph, and keep existing throughout the execution of the algorithm. Matching links on the other hand are created and possibly removed by the algorithm. In this paper, we drop this difference, and only use one kind of (dynamic) edges. If needed, we can easily emulate social (static) links by adding edges with maximal quality, which will not be removed at any point of the algorithm. Whereas Hoefer's focus was primarily on runtime, we primarily investigate the achieved welfare. Since our model is used to describe cooperation between different players or actual friendship, we also assume that both partners value a potential relationship identically.

1.2 Our Contributions and Paper Structure

We explain our model and our assumptions in Section 2. We describe local algorithms in the context of social networks which try to maximize the welfare of the participants. This means that they try to find good partners for every node, i.e., edges of high quality. The distributed algorithms executed on every node try selfishly to maximize the sum of the qualities of incident edges. We prove in Section 3 that there cannot be an optimal, local algorithm. We do this in two steps. First, in Subsection 3.1, we show that no local algorithm can compete with a global, optimal algorithm, i.e., any local algorithm will be arbitrarily worse in certain scenarios. Afterwards, in Subsection 3.2, in the spirit of [2], we compare local algorithms with other local algorithms. We prove that there is no best local algorithm, i.e., one which is always at least as good as every other local algorithm. For every local algorithm there exists a scenario where it is arbitrarily worse than another local algorithm. This includes randomized algorithms. In Subsection 3.3

we let the nodes execute several algorithms in parallel. Although every local algorithm can be arbitrarily worse than a global optimum, we show that the nodes can achieve a factor 2 approximation in comparison to the best applied strategy.

Furthermore, in Subsection 3.4 we study a slightly modified model, where friends of potential friends also matter for the valuation. We show that there exist scenarios in which a simple algorithm no longer achieves a stable state. In Subsection 3.5 we assume that ending a friendship permanently damages the quality of a friendship. Assuming a breakup reduces the quality of the friendship by a constant, the runtime of any algorithm is limited. Another model is that the quality of the friendship is reduced by a constant factor if the friendship is ended. Under this assumption there are scenarios in which edges are recreated very often and thus the quality gets reduced significantly.

2 Model

We model a social network with a set V of nodes (human beings), $n = |V|$. Between any two nodes $u, v \in V$ there is a quality function $q(u, v) \in [0, 1]$, representing the quality of the friendship, a larger value means better quality. We do not consider negative edge qualities since no node has an incentive to create an edge which reduces its welfare. The quality is symmetric, i.e., $q(u, v) = q(v, u)$. Without symmetry we can create the same cycling as in the roommate problem [6] and thus not reach a stable state. Initially, the nodes are connected by an arbitrary graph $G = (V, E)$, representing the initial knowledge graph. In other words, if two nodes are neighbors in G, they are initially friends. Nodes might decide to create new friendships (edges), and friendships can also be ended. We refrain from changing the friendship graph externally by changing the quality of an edge during the execution of the algorithm, since we want to analyze local algorithms. The set of edges E is as such highly dynamic.

A node's welfare (happiness) depends on the quality of its friendships. Formally, the welfare of a node v is defined as $\sum_{u \in N(v)} q(u, v)$, where $N(v)$ denotes the set of neighbors (current friends) of v. Nodes try to maximize their personal welfare by finding new friends with high quality values.

In reality, one cannot be friends with everybody. However, since our edge qualities are non-negative, nodes could just accumulate more and more friends, until G is a clique. We do not want this effect in our study; as such each node v has a maximum number of possible friends k_v, an individual constant parameter. If a node v already has k_v friends, and fancies a new friend, it must drop an old friend instead.

Nodes cannot choose friends arbitrarily. Instead, they can only choose friends that are already within their visibility. More formally, we define the constant parameter ℓ which we call lookahead. A node can only get a friend within ℓ hops of graph G. For example, if $\ell = 2$, apart from its friends (neighbors in G) a node can only see its 2-hop neighbors (friends of friends); new friends can only be found among these 2-hop neighbors. As such we deal with so-called ℓ-local algorithms. A node has all the information in its ℓ-hop neighborhood. In particular it knows about all the friendships and all the qualities in the ℓ-hop neighborhood.

Nodes run ℓ-local algorithm in order to optimize their friendship graph. Since friendships are a serious business with lots of potential for conflicts of interest, one needs to be careful about the issue which node can propose friendships to which other node at

what time. There are various meaningful models here. Indeed, our proofs are relatively robust and work in different kinds of algorithmic models.

For the sake of concreteness, we suggest the *round-robin* model. In this model, all nodes take turns in a round-robin fashion. Whenever it is the turn of a node v, v can propose friendship to different nodes in its ℓ-hop neighborhood. The order in which it asks these nodes is completely up to v; this order is basically the friend-finding algorithm. If v has already k_v friends, it only proposes to nodes whose friendship is more valuable (edge quality is higher) than that of v's worst friend, i.e., to better friends. A node u that is asked by v evaluates the proposed friendship. If u still has room for a new friend, or if v's proposed friendship is better than the worst of u's current friendships, u will accept the edge (and drop its worst friendship if necessary). In other words, a new edge is only added to the graph if both nodes u, v adjacent to edge (u, v) want the edge. If a node gets rejected from a potential new friend, it continues to ask other candidates according to the ordering.

If a node does not find any better friend, the round-robin model asks the next node to find a friend. The procedure ends if no new friendships can be discovered, i.e., if a whole round-robin loop does not change the friendship graph anymore. We call this a stable state.

Note that the initial friendship graph G constrains the final outcome. If two nodes are in different components of the initial graph G, then they can never become friends as they cannot learn about each other. Also, components may partition into smaller components during the execution of the algorithm. For the sake of simplicity, we assume that the initial friendship graph G is connected; however, alternatively, just think of our analysis to take place in one of the original components.

We can imagine various ways to increase the power of the nodes. In particular, nodes might have additional, constant size memory. Memory allows nodes to remember special former partners, e.g., the best ones they dropped, nodes for which the creation of the corresponding edge had some specific properties, or any other mechanism imaginable. Nodes stored in the memory can be added to the list of candidates which will be asked by the algorithm. An algorithm might try to combine several algorithms into one by executing them in parallel. This can be done by performing one round of each algorithm alternately and using the memory to remember the states of the other algorithms. Note that due to the constant memory, only a constant number of algorithms can be combined in this way.

3 On Welfare

In this section we compare different algorithms. As a measurement we use the welfare in the stable states. We compare the globally achieved welfares, i.e., the sum of welfare achieved by all nodes. Algorithm A is said to be arbitrarily worse than algorithm B if the welfare in the stable state of A is $\mathcal{O}(n \cdot \varepsilon)$ whereas it is $\Omega(n)$ in the stable state of algorithm B. Note that ε can be arbitrarily small, e.g., as small as any function in n such as $\varepsilon = 2^{-n}$.

In the model section we have described algorithms which only choose beneficial partners. Let us justify why we focus on this class of algorithms. We show that temporarily accepting worse friends results only in a constant increase in the lookahead.

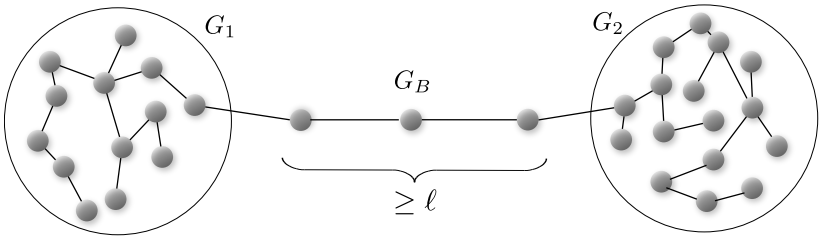

Fig. 1. Two subgraphs G_1, G_2 which are never in contact with each other because they are separated by a bridge G_B with a diameter of at least ℓ.

Lemma 1. *If all nodes are allowed to temporarily choose c worse neighbors, the length of shortest path between two nodes u, v can only decrease by at most a constant factor of ℓ^c.*

Proof. Let u, v be two nodes which are not neighbors. If the graph is connected, there exists a path from u to v via nodes $u_1, \ldots, u_k$. If each node is only allowed to select one worse partner, the distance is minimized if every ℓ-th node connects to a node in distance ℓ bypassing the $\ell - 1$ now unnecessary nodes in between. The path now consists of the nodes $u, u_\ell, u_{2\ell}, \ldots, v$. Hence, the distance can be reduced by at most a factor of ℓ in total, i.e., to at least $\frac{k}{\ell}$ which is a constant factor reduction since ℓ is independent of n. Thus, if we only allow a constant number c of consecutive bad choices, the distance decreases by a factor of at most ℓ^c, which is only a constant. □

Hence, if nodes are allowed to temporarily choose worse partners, all the proofs still hold by increasing the distances from ℓ to ℓ^c. Thus, we will not treat this separately but mention it briefly in the proofs.

3.1 Local vs Global Algorithms

After having introduced the basic idea of local algorithms, let us now analyze how they perform against a global optimum, i.e., a graph which maximizes the sum of welfare achieved by all nodes.

Theorem 2. *For nodes with a constant lookahead ℓ and a constant size memory there exist scenarios for every local algorithm such that its reached welfare is arbitrarily worse than a global optimum.*

Proof. Consider a scenario as depicted in Figure 1. The initial graph consists of three subgraphs G_1, G_2 and G_B. The two larger subgraphs G_1, G_2 are connected through a bridge G_B which has a diameter of at least ℓ and each node v in the bridge has already k_v friends. The valuations are such that for every pair $(u, v) \in G_1 \times G_2$ the quality is 1, for every pair $(u, v) \in G_i \times G_i$ with $i \in \{1, 2\}$, we have $q(u, v) = \varepsilon$. Furthermore, for every $(u, v) \in G_i \times G_B$ with $i \in \{1, 2\}$, $q(u, v) = \varepsilon/2$. Thus, we only need to specify the valuations within G_B. Every node $v \in G_B$ values other nodes $u \in G_B$ with 2ε if the edge exists in the initial friendship graph and 0 otherwise.

Hence, the nodes in the bridge already have their best possible friends and therefore will not change their friends. These valuations represent an initial friendship graph in which no node really likes his friends but does not know any better candidates.

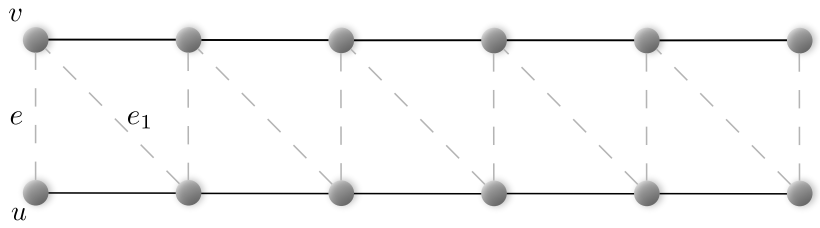

Fig. 2. A track going from left to right. The dashed, gray edges are created by the execution of a local algorithm, the black edges are given in the initial friendship graph.

But this setting is a stable state, hence no local algorithm will create an edge between any node from G_1 with any node from G_2 because of the sheer distance between those subgraphs. This holds due to Lemma 1 even if the nodes are allowed to choose non-beneficial partners. Furthermore, the connecting nodes are not appealing for any node and thus remain isolated. Therefore, the best achievable stable state is $\mathcal{O}(n \cdot \varepsilon)$. In the optimal solution the nodes from the sets G_1, G_2 are connected to each other to achieve a stable state with value $\Theta(n - |G_B|) = \Theta(n)$. □

Note that this result relies on the fact that any reasonable, local algorithm is only willing to create connections to beneficial partners or is only willing to accept a worse partner a constant number of times. Let us further remark that the initial friendship graph and any stable state reached by a local algorithm is not necessarily Pareto efficient. This means that there are scenarios where we can easily improve the welfare of some nodes without lowering the welfare of other nodes, e.g., by moving one node v from G_1 to G_2. Now v can connect to nodes which it valuates higher. This increases v's welfare (and the welfare of the nodes which are connected to v) but does not lower the welfare of any node.

3.2 Local vs Local Algorithms

We now show that there cannot even be a best local algorithm, i.e., an algorithm which can achieve a stable state whose welfare is at least as good as the welfare of any other local algorithm. We prove that for every local algorithm there exist scenarios in which it is arbitrarily worse than another, local algorithm. But first, we need a few more terms.

Proposition 3. *A* track *consists of two disjoint sets, each with j nodes. The nodes of each set are initially arranged in a line as shown in Figure 2 (connected to each other with the edges colored in black). The length of the track is j. The dashed, gray edges have a strictly monotonic increasing quality from left to right. The black edges from the initial friendship graph have a quality of $\mathcal{O}(\varepsilon)$. The remaining edges have a quality of 0 and are therefore never used. Every node v of a track has $k_v \geq 4$. After the* initial edge *e is created, any algorithm in our model will create the dashed, gray edges, starting with e_1, one by one from left to right, i.e., in increasing order regarding their quality. The creation of one dashed, gray edge allows the creation of the next dashed, gray edge since those nodes are now within ℓ hops of each other. We call the creation of the edges* exploring a track.

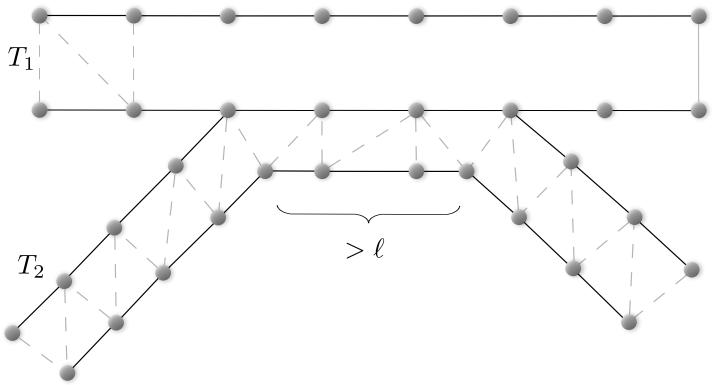

Fig. 3. Two tracks T_1 and T_2 interacting. T_1 is blocked by T_2 since the shared nodes of both tracks have no incentive to create the edges of T_1 since they are content with the edges of T_2.

Proposition 4. *A* successful track *generates a welfare of $\Omega(n)$ if the initial edge e is created. Without this initial edge the track has only a welfare of $\mathcal{O}(n \cdot \varepsilon)$. A track of length $\Omega(n)$ can have these properties. In such a track we can set the quality of the edges in the initial friendship graph to $\mathcal{O}(\varepsilon)$ and the edge quality of the selected edges connecting those two sets, i.e., the dashed, gray edges in Figure 2 to a constant.*

Upon creating the initial edge e on the left, the track can be explored. After every dashed, gray edge is created, the welfare is $\Omega(n)$; without the initial edge the welfare is generated only by the edges of the initial friendship graph and thus $\mathcal{O}(n \cdot \varepsilon)$.

Proposition 5. *A track T is said to be* blocked *if, during the exploration, no further edge is created because at least ℓ consecutive nodes v have already k_v edges which are better than those of the track T. This stops the exploration since the nodes have no incentive to continue to explore the track.*

Similarly, a track T_2 blocks *another track T_1 if the edges of T_2 are part of the reason why T_1 is blocked. An example of this can be seen in Figure 3.*

Theorem 6. *For nodes with a constant lookahead ℓ and a constant size memory there exist scenarios for every deterministic, local algorithm such that its reached welfare is arbitrarily worse than that of another deterministic, local algorithm.*

Proof. Consider two concatenated tracks T_1, T_2 each of length at least ℓ. On top of each of the ℓ nodes of the first track T_1 are four nodes in a line, i.e., each node u_i is connected to a different intermediate node v_i' which is connected to node v_i. This subgraph of the initial friendship graph is depicted in Figure 4. We define $k_{v_i} = 2$ and $k_{u_i} = 4$ for all nodes u_i, v_i with $i \in \{1, \ldots, \ell\}$, i.e., every node u_i can create a connection to exactly one more node whereas v_i must sever an edge to create a new edge. Let v_i' be v_i's worst friend. The qualities are such that $q(u_i, v_i) > q(u_i, x_i)$ holds. Let the edges from the initial friendship graph have a quality larger than $q(u_i, v_i)$ for all $i \in \{1, \ldots, \ell\}$.

We assume that node v_i has two options. It can either create a connection to node u_i or to node w_i; all other nodes are valued with 0. If all nodes v_i decide to create a connection with u_i, T_1 is blocked. Let us explain this in more detail. Since $q(u_i, v_i) > q(u_i, x_i)$ and the quality of every edge of the initial friendship graph is also larger than

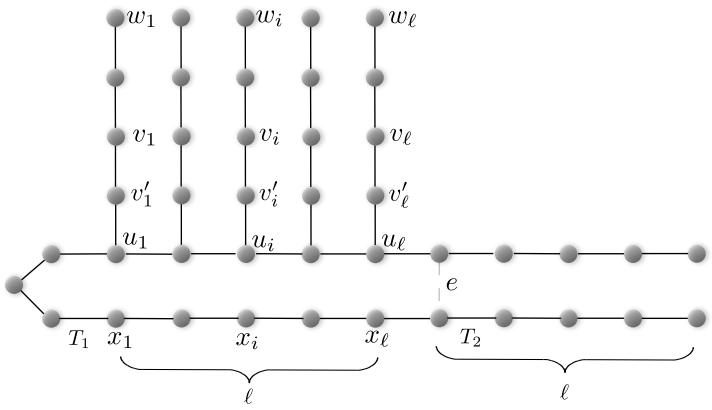

Fig. 4. A subgraph of size $\mathcal{O}(1)$ with outgoing track. Node v_i must now decide if it wants to create a connection to node w_i or to node u_i. If all the connections to u_i are made, the track cannot be explored. Note that edge e is not part of the initial friendship graph.

$q(u_i, x_i)$, node u_i has no incentive to create the edge (u_i, x_i) or any other edge. Thus, the track T_1 is not explored and the initial edge e of T_2 is not created. But if all nodes v_i choose to create a connection with w_i, track T_1 and subsequently track T_2 are explored. Note that it is unimportant if the nodes v_i may have the option to temporarily revise their decision by using their constant memory. It only matters whether the track T_2 is explored at some point execution of the algorithm.

Since the nodes can only see the graph and a part of the track, they have to make a decision with only a subset of the information available. Hence, they have to base their decision on insufficient information because they cannot know whether the exploration of the track T_2 is necessary in order to create partnerships between most of the nodes. This can occur if the track T_2 turns out to be a successful track. But it might also happen that this track blocks a successful track and should therefore not be explored. The latter scenario is shown in Figure 3. In this scenario the chosen track prevents the exploring of the other track. Since these scenarios are indistinguishable for any algorithm, the remainder of the graph can be such that its choice is wrong. It is easy to see that there is another local algorithm which decides correctly for this particular scenario. Limiting the quality of the edges in the subgraph to $\mathcal{O}(\varepsilon)$ yields the theorem. □

We can prove a similar result for algorithms which try to execute a constant number of different algorithms in parallel to avoid the aforementioned problem. This allows them to emulate algorithms where one might explore a track whereas another might not.

Corollary 7. *For nodes with a constant lookahead ℓ and a constant size memory there exist scenarios for every local algorithm, which executes several algorithms in parallel, such that its reached welfare is arbitrarily worse than that of another local algorithm.*

Proof. To deal with the fact that one algorithm might explore the track whereas another might not, we use a slightly more sophisticated idea. Once again, we use the idea of a successful track. Let T_3 be such a track. We concatenate different subgraphs to prove the corollary. We can use the subgraphs to ensure that each of the executed algorithms makes a severe mistake, i.e., has no longer a chance of being optimal. We now construct the whole graph step by step.

Imagine a subgraph with two for the nodes indistinguishable outgoing tracks T_1, T_2 of length $\Omega(\ell)$. Any algorithm has only four choices available. Either it explores no track, T_1, T_2 or both. W.lo.g. half of the nodes explore either T_1 (and possibly T_2 as well) or explore no track at all. Apparently, we want that decision to be incorrect. Hence, the graph is such that T_1 blocks the successful track T_3. Furthermore, exploring T_2 is necessary in order to explore T_3 later on. Thus, the optimal algorithm explores only track T_2. This construction also deals with algorithms which explore both tracks. Hence, half of the executed algorithms have chosen in a way such that they cannot be optimal anymore.

Track T_2 leads to another subgraph where the process is repeated. This subgraph has also two, for the algorithm indistinguishable, outgoing tracks. One of the tracks blocks the successful track T_3 whereas the other one leads to the next subgraph. This is repeated a sufficient number of times. In the last subgraph one of the tracks is the successful track T_3 and the other track once again blocks that track.

By setting the edge qualities in the subgraphs to $\mathcal{O}(\varepsilon)$, we ensure that the local algorithm which emulates other algorithms, can only choose edges with quality $\mathcal{O}(\varepsilon)$ and thereby limiting its welfare to $\mathcal{O}(n \cdot \varepsilon)$. Thus, even running several algorithms in parallel is not sufficient to obtain the best local algorithm. With these edge qualities it can be arbitrarily worse than another local algorithm. □

Theorem 8. *For nodes with a constant lookahead ℓ and a constant size memory there exist scenarios for every randomized, local algorithm such that its reached welfare is arbitrarily worse than that of another deterministic, local algorithm.*

Proof. Imagine a scenario similar as in Theorem 6. Any randomized local algorithm either chooses to explore the track with probability at least $\frac{1}{2}$ or chooses not to explore the track with probability at least $\frac{1}{2}$. Hence, we can first concatenate b of these structures with $b = \Theta(\log n)$ such that the probability that the randomized algorithm chooses correctly is 2^{-b}, i.e., chooses the correctly with low probability, i.e., with probability $n^{-\alpha}$ for some constant $\alpha, \alpha > 0$. Once again, using the same argument as before, there exists a local algorithm which always chooses deterministically and correctly which achieves an optimal solution. By choosing the edge values accordingly, we can ensure that the achieved stable states differ sufficiently. □

3.3 Executing Local Algorithms in Parallel

We have seen that the welfare in the stable states that different algorithms reach differs significantly. Although, as seen in Corollary 7, none of them may produce an optimal solution, we try to salvage as much as possible by selecting a constant number of algorithms and trying to be as close as possible to the best achieved solution. If we execute several algorithms in parallel, we obtain more than one solution. Due to the fact that the nodes only have a local view, they cannot know which of their connections is part of the best achieved solution. With their limited knowledge, the obvious strategy for the nodes is to simply try to choose the best available connection if still available. This does not yield the best solution as shown in the easy example depicted in Figure 5. But we get a factor 2 approximation compared to the welfare achieved by any of the executed algorithms. To be able to pick edges greedily at the end, we need to show an upper bound on the runtime which allows the nodes to know when any algorithm has terminated.

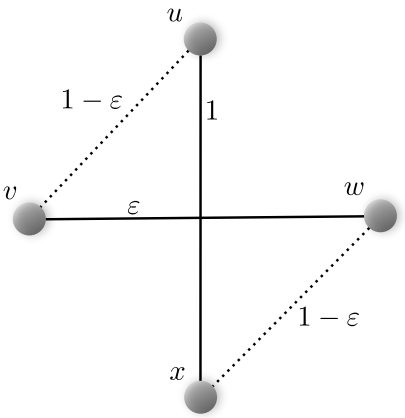

Fig. 5. The best solution consists of the edges $\{u, v\}$ and $\{w, x\}$ whereas our greedily picked solution consists of $\{u, x\}$ and $\{v, w\}$ and is thus a factor of 2 worse than the best solution.

Lemma 9. *The runtime of any local algorithm that only chooses higher quality edges is* $\mathcal{O}\left(2^{n^2}\right)$.

Proof. We use a potential function to prove this. Consider a bitstring of length n^2. The i-th bit represents the edge with the i-th largest quality. There is a 1 at position i if the corresponding edge with the i largest quality exists in the graph. The bitstring can be regarded as a counter. Since we only allow beneficial changes, this potential function increases with every change. This limits the total runtime of any algorithm of this type to 2^{n^2}. □

Note that the nodes cannot know when exactly the execution has terminated because of their limited view, but only know the rather weak upper bound of 2^{n^2}. Hence, the greedily selecting of the edges will be started after 2^{n^2} rounds. Since at least one edge is picked every round, this allows the nodes to output a valid solution after $\mathcal{O}\left(2^{n^2} + n^2\right)$ rounds.

Theorem 10. *By running several algorithms in parallel and greedily selecting the best edges at the end, we obtain a factor 2 approximation compared to the best of the executed algorithms.*

Proof. This proof is similar to the proof that any maximal matching is a factor 2 approximation of a maximum matching. Consider the union of edges of all solutions. Whenever two nodes mutually agree to pick an edge, this edge can either be part of the best solution in which case the choice is good. Otherwise, our choice might make it impossible to pick at most two edges from the best solution since both are connected to one of the vertices. But both must have a lower quality than our choice. Hence, our solution is at least half as good as the union of all solutions and therefore at least half as good as the best solution. Continuing this inductively yields the claim. □

3.4 Valuing Friends of my Friends

In a real social network it might not be sufficient to simply evaluate a friend by himself. In order to evaluate a friendship, it is sometimes necessary to also consider the friends

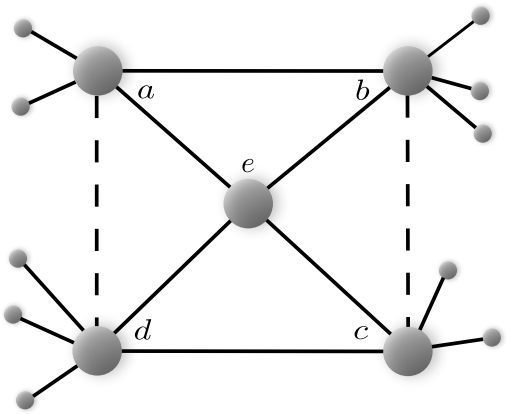

Fig. 6. The edges $\{a, b\}, \{c, d\}$ exist. In this setting b prefers to be paired up with c and c would be happier with that matching. Afterwards a would match with c and thus b with d. In the next round, the cycle would start anew.

of a friend. Thus, we want to introduce another friendship valuation variant. An edge continues to represent an existing friendship, but the new edge quality is a weighted, combined value of the node and its neighbors. More formally this can be expressed as $Q(u, v) := q(u, v) + c \sum_{x \in N(v) \setminus \{u\}} q(u, x)$ where q denotes the quality function as defined before and c is some constant. In this model every edge quality $Q(\cdot, \cdot)$ is directed, i.e., $Q(u, v) \neq Q(v, u)$ is possible.

In this slightly advanced model, there may not be a stable state. This is due to the asymmetric valuations of the nodes which can be used to create valuations similar as in the roommate problem in [6].

Theorem 11. *If we include the neighbors of a node in the valuation function, there are scenarios in which a local algorithm does not reach a stable state.*

Proof. It is sufficient to consider the nodes a, b, c, d, e and their friends with the following valuation for each other: $q(a, b) = q(a, c) = q(b, c) > q(a, d) = q(b, d) = q(c, d)$. This means the three nodes a, b, c prefer each other over node d. The valuations of node e are such that it has no incentive to choose another friend. Now we can set the edge qualities of the friends of each node such that the final edge qualities are $Q((a, b)) > Q((a, c)) > Q((a, d))$. Node a prefers the friends of b over those of node c and so on. Furthermore, we require $Q(b, c) > Q(b, a) > Q(b, d)$ and $Q(c, a) > Q(c, b) > Q(c, d)$. The evaluations of node d do not matter. A brief technical remark has to be made. In order to achieve this, the friends of each node have to be content with being paired up with their respective partners. Furthermore, neither node must be willing to initiate a connection with any other node than a, b, c or d. For the nodes to be able to know each other all the time, the fifth node e can be connected to a, b, c, d but without any incentive to change its connections. This scenario is depicted in Figure 6.

This enables us to create a cycle. No matter which node is matched with node d, it wants to change to another node which is willing to do so. Hence, no stable state can be reached. □

Clearly, neither a statement about the convergence time nor about the globally achieved welfare is possible in this setting.

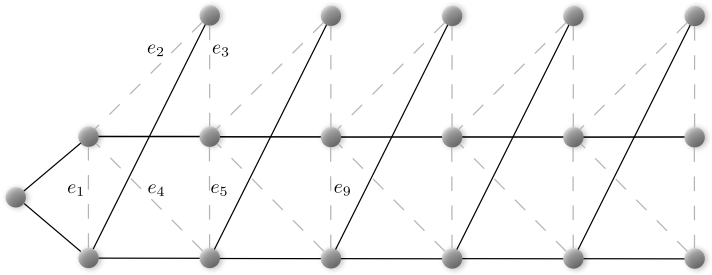

Fig. 7. A counter as presented by Hoefer [13]. Edges which are created from the algorithm in dashed, gray, the others in black.

3.5 Breaking Up a Friendship is Expensive

In a real social network, breaking up friendships is hardly without consequences to that friendship. To model this we assume that the edge quality decreases every time the corresponding edge is removed. There are two natural choices to reduce the quality of an edge: Either reduce it by a constant term, or by a constant factor.

Let us first analyze some immediate consequences of this to the runtime of any local algorithm. In [13] it has been proven that a simple greedy algorithm which does memory has a runtime of $\Omega\left(2^{\Theta(n)}\right)$. We obtain a dramatically smaller runtime if the quality of the edge gets reduced by a constant.

Theorem 12. *If the edge quality $q(e)$ gets reduced by a constant term every time the edge e is removed, the runtime of any local algorithm is $\mathcal{O}\left(n^2\right)$.*

Proof. Every edge can only be created and severed $\mathcal{O}(1)$ times before the nodes have no incentive to create that edge because its quality is 0. Hence, after $\mathcal{O}\left(n^2\right)$ any algorithm must terminate. □

Before we describe the effects of reducing the edge qualities by a constant factor, let us describe an initial friendship graph. It is the same used in [13] to prove the lower bound for the simple algorithm. We will show that the result still holds if the edge quality is decreased and consider the consequences for the welfare.

Imagine the nodes arranged as show in Figure 7. The edge qualities are such that $q(e_1) < q(e_2) < q(e_3) < q(e_4) < q(e_5)$ holds. The whole graph is constructed by concatenating these blocks. We assume that the edge qualities are constant. For every node v its value k_v is such that it can create one additional friendship. For simplicity's sake, we now assume that the lookahead ℓ is limited to 2. We analyze a greedy local algorithm which always chooses the best available option.

Let us describe what happens in this scenario. At the beginning edge e_1 can be created which enables the partnership e_2. While creating e_3, this edge will also be severed. Thus, the creation of edge e_1 is again possible. Since the edge e_2 cannot be created, the simple algorithm creates e_4, thereby severing e_1 the second time. Finally, the partnership e_5 is formed. Note that edge e_1 was created twice in order to create e_5.

Lemma 13. *The greedy algorithm without memory has a runtime of* $\Omega\left(2^{\Theta(n)}\right)$.

Proof. The inequalities stated above still hold even if we decrease the quality by a constant factor every time we remove the corresponding edge. We can concatenate b of those structures with $b = \Theta(n)$. Since e_1 must be created twice in order to create e_5 once and e_5 in turn must be created twice in order to enable the partnership e_9, the concatenation of these blocks requires e_1 to be created $\Omega(2^b)$ times and thus the lemma follows. □

Theorem 14. *If the edge quality* $q(e)$ *gets reduced by a constant factor every time the edge* e *is removed, there is scenario such that the combined quality of the edges created by the greedy algorithm can be upper bounded by a constant. But in the same scenario, using memory enables an algorithm to create edges with a combined quality of* $\Theta(n)$.

Proof. As explained above, every edge gets recreated often, i.e., in the i-th building block $b - i$ times. Let $1/x$ be the factor by which each edge quality gets reduced upon removing. The total welfare of the edges created in the i-th block is at most $c \cdot 2^{b-i}$. Summing up over all blocks yields a total welfare of

$$c \cdot \sum_{i=1}^{b} 1/x^{b-i} = c \cdot \left(\frac{x}{x-1} - \frac{x^{1-b}}{x-1} \right) = \mathcal{O}(1).$$

Imagine an algorithm where the nodes simply remember every friend they ever had (which is a constant in this scenario). This means that after the first creation of e_5, it is no longer necessary to sever and recreate e_1. Thus, every edge gets only removed and recreated a constant number of times. This allows the total welfare to be $\Theta(n)$. □

Note that we can adapt this proof for $\ell > 2$ by using a slightly more advanced way to construct the graph but keeping the basic idea. We concatenate blocks in which one edge needs to be created twice in order to create another edge.

4 Conclusion & Outlook

We showed that any local algorithm for finding better friends that has constant memory and constant lookahead is arbitrarily worse than the global optimum. We also compared local algorithms to each other: For every local algorithm there exists a scenario in which it performs arbitrarily worse than another local algorithm.

This was shown for some specific initial friendship graphs. An interesting open problem is how to characterize graph classes where a best local algorithm exists. Are there also some general graph classes where the welfare of a local algorithm is only a constant factor worse than the global optimum? Another open question is whether our results can be generalized. Which problems cannot be solved well by a local algorithm; neither in comparison to other local algorithms nor compared to the optimal solution?

References

1. D.J. Abraham, A. Levavi, D.M. Manlove, and G. O'Malley. The stable roommates problem with globally-ranked pairs. In *Proceedings of WINE 2007: 3rd International Workshop On Internet and Network Economics*, volume 4858 of *Lecture Notes in Computer Science*, pages 431–444. Springer, December 2007.
2. Miklos Ajtai, James Aspnes, Cynthia Dwork, and Orli Waarts. A theory of competitive analysis for distributed algorithms. In *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on*, pages 401–411. IEEE, 1994.
3. Esteban Arcaute and Sergei Vassilvitskii. Social Networks and Stable Matchings in the Job Market. In *WINE*, pages 220–231, 2009.
4. Paul Belleflamme and Francis Bloch. Market Sharing Agreements and Collusive Networks. *International Economic Review*, 45(2):387–411, 2004.
5. Antoni Calvó-Armengol and Matthew O. Jackson. Networks in labor markets: Wage and employment dynamics and inequality. *Journal of Economic Theory*, 132(1):27–46, 2007.
6. D. Gale and L.S. Shapley. College Admission and the Stability of Marriage. *American Mathematical Monthly*, 69(1):9–15, 1962.
7. Effrosyni Diamantoudi, Eiichi Miyagawa, and Licun Xue. Random paths to stability in the roommate problem. *Games and Economic Behavior*, 48(1):18–28, 2004.
8. Robin Dunbar. *How Many Friends Does One Person Need?: Dunbar's Number and Other Evolutionary Quirks*. Harvard University Press, 2010.
9. Federico Echenique and Jorge Oviedo. A theory of stability in many-to-many matching markets. *Theoretical Economics*, 1(2):233–273, 2006.
10. Patrik Floréen, Petteri Kaski, Valentin Polishchuk, and Jukka Suomela. Almost Stable Matchings by Truncating the Gale-Shapley Algorithm. *Algorithmica*, 58(1):102–118, 2010.
11. Dan Gusfield and Robert W. Irving. *The Stable marriage problem - structure and algorithms*. Foundations of computing series. MIT Press, 1989.
12. Magnús M. Halldórsson, Robert W. Irving, Kazuo Iwama, David F. Manlove, Shuichi Miyazaki, Yasufumi Morita, and Sandy Scott. Approximability results for stable marriage problems with ties. *Theoretical Computer Science*, 306(1–3):431–447, 2003.
13. Martin Hoefer. Local Matching Dynamics in Social Networks. *Automata Languages and Programming*, pages 113–124, 2011.
14. Kazuo Iwama, Shuichi Miyazaki, David Manlove, and Yasufumi Morita. Stable Marriage with Incomplete Lists and Ties. In *Proceedings of the 26th International Colloquium on Automata, Languages and Programming*, ICALP '99, pages 443–452, London, UK, 1999. Springer-Verlag.
15. Matthew O. Jackson. *A Survey of Models of Network Formation: Stability and Efficiency*, pages 1–62. Cambridge University Press, 2005.
16. Alexander S. Kelso and Vincent P. Crawford. Job Matching, Coalition Formation, and Gross Substitutes. *Econometrica*, 50(6):1483–1504, 1982.
17. Samir Khuller, Stephen G. Mitchell, and Vijay V. Vazirani. On-line algorithms for weighted bipartite matching and stable marriages. *Theoretical Computer Science*, 127:255–267, May 1994.
18. Rachel E. Kranton and Deborah F. Minehart. A theory of buyer-seller networks. *American Economic Review*, 91(3):485–508, 2001.
19. David F. Manlove, Robert W. Irving, Kazuo Iwama, Shuichi Miyazaki, and Yasufumi Morita. Hard variants of stable marriage. *Theoretical Computer Science*, 276(1-2):261–279, 2002.
20. Alvin E. Roth and Marilda Sotomayor. Two-sided matching. *Handbook of Game Theory vol 1*, 78(2):75–95, 1990.