Speed Dating
An Algorithmic Case Study Involving Matching and Scheduling

Bastian Katz, Ignaz Rutter, Ben Strasser, and Dorothea Wagner

Faculty of Informatics, Karlsruhe Institute of Technology (KIT), Germany
{bastian.katz,ignaz.rutter,dorothea.wagner}@kit.edu,
mail@ben-strasser.net

Abstract. In this paper we formalize and solve the *speed dating problem*. This problem arises in the context of speed dating events, where several people come together to date each other for a short time. For larger events of this type it is not possible to have each possible pair of persons meet. Instead, based on forms filled out by the participants, the organizer of such an event decides in advance, which pairs of people should meet and also schedules the times of their dates. Moreover, since people pay for participating in such events, aside from the overall quality of the dates, it is important to find a *fair* schedule, where people from the same group (e.g., all women) have a comparable number of dates.

We model the organizer's problem for speed dating, study its complexity and design efficient algorithms for solving it. Finally, we present an experimental evaluation and show that our algorithms are indeed able to solve realistic problem instances within a reasonable time frame.

1 Introduction

A speed dating event is an event, where people that wish to find a partner, come and date each other for a short period of time. When such events came up they were usually small enough that during one evening every potential pair could have a date. Due to the increasing size of such events this is no longer feasible and the organizer has to schedule in advance who meets whom, and when. The schedule is usually set up in rounds such that in each round each person dates at most one other person. Sometimes it is not possible that everybody has a date in every round, for example the ratio of men to women usually is around 3:2, and hence some men need to skip a round now and then.

To come up with potential matches, the organizer asks the participants to fill out forms about their ideal partner before the event. Based on this information he estimates a certain quality for each date. Certain dates have such a poor quality that they would only upset the participants and therefore should in no case take place. One approach to solving the speed dating problem would now be to select a set of dates, such that the total quality of all dates is maximized and no person has more dates than the number of rounds. This allows to limit the number of rounds and thus guarantees a fixed time frame for the whole event.

There are a few problems to consider when choosing the set of pairs that meet each other and an according schedule. First of all, in each round every person can be involved in at most one date, i.e., for each round, the set of dates forms a matching of the people. Some persons may be more attractive than others. When maximizing the total quality of all dates, they would probably get a lot of dates, possibly at the cost of other people. Since all people participating in a speed dating event pay for the registration, it is crucial to find a fair distribution of dates. However, a completely fair distribution is not always feasible. For example there generally are more men than women participating in such events. The only way to deal with this problem is to make some men skip some of the rounds. Again these should be fairly distributed among all men. If there are n men and m women with $n > m$, σ rounds yield a total of $m\sigma$ dates. Hence every man should get roughly $m/n \cdot \sigma$ dates. As $\sigma \cdot m$ may not be divisible by n it would be reasonable to require for every man at least $\lfloor m/n \cdot \sigma \rfloor$ and at most $\lceil m/n \cdot \sigma \rceil$ dates. We will however introduce more flexible bounds that allow an upper and lower bound for each vertex individually. In this way more elaborate constraints can be modeled, such as VIP persons who pay more and are less likely to skip rounds. Finally, it is important that a solution can be computed quickly as the organizer of such events may wish to perform the computation of the schedule as close to the start of the actual event as possible, in order to accomodate for late registrations or people that register but do not show up.

So far we have only described *bipartite speed dating*, where the people are divided into two groups (men and women) and all dates take place between people from different groups. We also consider the *general speed dating* problem, where we do not make this distinction between men and women and allow arbitrary dates to take place. This more general setting could also be applied in other settings, for example at scientific meetings in Dagstuhl people are usually randomly assigned to tables for dinner. To facilitate an efficient communication among researchers, it may be desirable to determine this assignment based on a network representing common research topics, instead. Note that the bipartite speed dating problem is a special case of the general speed dating problem, where the dates between persons of the same gender are rated very badly. We will see later that these two problems behave quite differently in terms of computational complexity.

Outline and Contribution. We introduce the problem of scheduling meetings of pairs in a group of people, as they arise for example in speed dating events. Based on the already identified criteria total quality and fairness, we derive a precise problem formulation for the speed dating problem. We show that the general problem is NP-hard and give a polynomial-time algorithm for the bipartite case. We further show that the general case admits a polynomial-time algorithm that simultaneously approximates the total quality and the fairness violation. Finally, we demonstrate the effectiveness and efficiency of our algorithms with an experimental evaluation that compares the performances of our algorithms with a greedy solution on a variety of problem instances, among them randomly generated instances and real-world instances of social networks.

The paper is organized as follows. First, we formalize the speed dating problem and study its complexity in Section 2. In Section 3 we give a polynomial-time algorithm for the bipartite case and modify it into an approximation algorithm for the general case in Section 4. We present our experimental evaluation and our conclusions in Section 5.

2 Preliminaries

In this section we formalize the speed dating problem, arrive at a formal problem statement, and consider its complexity status. Along the way we introduce notations that we use throughout this paper.

The input to the general speed dating problem consists of a tuple (G, q, ℓ, h, σ), where $G = (V, E)$ is an undirected graph with weights $q\colon E \longrightarrow \mathbb{N} \setminus \{0\}$ and two functions $\ell, h\colon V \longrightarrow \mathbb{N}_0$ specifying lower and upper bounds for the number of dates for each vertex such that $\ell(v) \leq h(v) \leq \deg(v)$ holds for all $v \in V$, as well as a number of rounds σ.

A *feasible solution* to the speed dating problem can be encoded as a subgraph $G' = (V, E')$ of G with the property that $\ell(v) \leq \deg_{G'}(v) \leq h(v)$ for all $v \in V$ along with a *proper coloring* of the edges that uses at most σ colors. A proper coloring of the edges is such that any two edges sharing a vertex have distinct colors. The *quality* of a feasible solution can be measured by the total weight of its edges, i.e., $q(G') = q(E') = \sum_{e \in E'} q(e)$. For a solution G' of the speed dating problem we also write $\text{dates}(v)$ instead of $\deg_{G'}(v)$ to denote the number of dates a vertex v participates in.

The main problem with this approach is that, depending on the structure of the input graph G, a feasible solution may not even exist as it may not be possible to find a solution with $\text{dates}(v) \geq \ell(v)$ for all $v \in V$. We therefore relax this lower bound on the number of dates and consider any solution G' that is properly σ-edge colored and satisfies $\text{dates}(v) \leq h(v)$ for all $v \in V$. We measure the quality of such a solution in terms of the quality as introduced before and by its *fairness violation* $\delta(G')$. The fairness violation $\delta(v)$ of a single vertex v is the amount to which its lower bound is violated or 0 if it is satisfied, i.e., $\delta(v) = \max\{\ell(v) - \text{dates}(v), 0\}$. The overall fairness violation of a solution is the maximum fairness violation among all vertices, i.e., $\delta(G') = \max_{v \in V} \delta(v)$.

In a nutshell, the fairness violation δ describes the degree of fairness of a given solution and the quality q describes the overall quality of the selected dates. Since customer satisfaction is a priority, we focus on minimizing the fairness violation first and on optimizing the quality of the dates as a second priority. We are now ready to formally state the speed dating problem.

Given an instance (G, q, ℓ, h, σ), the problem SPEEDDATING asks to find a solution G' that minimizes the fairness violation $\delta(G')$ and among all such solutions has the maximum quality $q(G')$. The bipartite speed dating problem is defined analogously, except that the graph G is bipartite. In the following we will assume that $\max_{v \in V} h(v) \leq \sigma$ as it never makes sense to allow more dates than rounds for any person. Unfortunately, the general problem is NP-complete.

Theorem 1. SPEEDDATING *is NP-complete, even if* $\sigma = 3$.

The proof is by reduction from EDGECOLORING, we omit it due to space constraints. Although this problem is NP-hard in its general form, solving the speed dating problem is not completely hopeless since EDGECOLORING admits solutions for quite some interesting cases. First of all, for bipartite graphs Δ colors always suffice and a solution can be computed efficiently [2]. Second, every graph can be colored with at most $\Delta + 1$ colors [7] and such a coloring can be computed efficiently [6].

3 Bipartite Speed Dating

In this section we design an algorithm for the bipartite case of SPEEDDATING. The key observation here is that any edge set E' satisfying the upper bound on the number of dates for each vertex forms a bipartite graph with maximum degree at most σ, and therefore it can always be colored with σ colors. Hence, the coloring subproblem is trivially solvable and does not impose any additional constraints on the subgraph that needs to be selected. This simplifies the problem to finding a subgraph of G that maximizes the total weight among all solutions that minimize the fairness violation. Our algorithm therefore works in three phases.

1. Determine the minimum value δ such that an edge set E' with $\ell(v) - \delta \leq \text{dates}(v) \leq h(v)$ for all $v \in V$ exists, using a binary search.
2. Determine the maximum weight edge set E' of G with $\ell(v) - \delta \leq \text{dates}(v) \leq h(v)$ for all $v \in V$ for the fixed value of δ determined in the previous phase.
3. Color the edge set determined in the second phase with at most $\Delta_{G'}$ colors.

We will now describe the phases. Note that due to the properties of the bipartite edge coloring problem, Phase 3 is completely independent from the previous two phases. This independence is however not given between Phases 1 and 2. In fact, Phase 1 will make use of the algorithm developed in Phase 2 to check whether a solution exists for a given fixed δ.

Phase 1: Determining the minimum fairness violation. Given a fixed fairness violation δ, the speed dating problem can be transformed into an equivalent instance, where the lower bounds on the number of dates are strict, i.e., $\ell(v) \leq \text{dates}(v)$ must hold for each node v by setting $\ell(v) \leftarrow \max\{0, \ell(v) - \delta\}$. After this transformation, finding an uncolored edge set boils down to the *weighted degree-constrained edge-subset problem* (WDCES).

Problem 1 (WDCES). Given a graph $G = (V, E)$ with edge weights $w : E \longrightarrow \mathbb{N} \setminus \{0\}$, lower and upper node capacities $\ell(v), h(v) : V \longrightarrow \mathbb{N}$ with $\ell(v) \leq h(v) \leq \deg(v)$, determine an edge set $E' \subseteq E$ of maximum weight such that $\ell(v) \leq \text{dates}(v) \leq h(v)$ holds for all nodes.

This problem is also known as weighted b-matching with unit edge capacities and has been previously studied by Gabow [3]. In the description of Phase 2 we will describe a simple MINCOSTFLOW based algorithm for solving this problem in the bipartite case. In Phase 1 we essentially discard the weights and simply wish to determine whether any feasible solution exists. We use this to determine the minimum value of the fairness violation δ by a binary search. The optimum value of δ is in $\{0, \ldots, \max h(v)\}$. If, for a fixed δ, the resulting WDCES problem does not admit a feasible solution, then smaller values of δ are also infeasible. Similarly, feasible solutions remain feasible for higher values of δ. Hence, binary search can be used to determine the smallest value δ for which the instance becomes feasible. This takes $O(\log n)$ feasibility tests.

Phase 2: Determining a Maximum Weight Subgraph. This phase consists of solving the WDCES problem that is obtained from an instance of bipartite SPEEDDATING by fixing the fairness violation to a certain value δ. To this end, we model the problem as a min cost flow problem (MINCOSTFLOW) as follows.

Let $G = (V,E)$ be an instance of WDCES with weight function w and lower and upper bounds for the nodes ℓ and h, stemming from fixing the fairness violation of a bipartite speed dating instance to a certain value δ. We construct an instance of min cost flow as follows. Recall that G is bipartite and hence $V = V_1 \cup V_2$ with $V_1 \cap V_2 = \emptyset$ and all edges in E have endpoints in both sets. The flow network N has vertices $V_1 \cup V_2 \cup \{s,t\}$, where s will act as a super source and t as a super sink. For each edge $uv \in E$ with $u \in V_1$ and $v \in V_2$ we add the arc (u,v) to N and set its lower capacity $\ell_N(u,v) = 0$ and its upper capacity $h_N(u,v) = 1$, and its cost $c(u,v) = -q(uv)$. We call these arcs *main arcs*, all other arcs will be *helper arcs*. For each vertex $u \in V_1$ we add an arc (s,u) with lower capacity $\ell_N(s,u) = \ell(u)$ and upper capacity $h_N(s,u) = h(u)$. Similarly, for all vertices $v \in V_2$ we add arcs (v,t) again with lower capacity $\ell_N(v,t) = \ell(v)$ and upper capacity $h_N(v,t) = h(v)$. All helper arcs have cost 0.

We set the desired flow value p that should flow from s to t to $|E|$. Further, we introduce an arc from s to t with $\ell_N(s,t) = 0$, $h_N(s,t) = p$ and $c(s,t) = 0$. This ensures that enough flow can be routed through the network to possibly select all edges in E, and on the other hand that superfluous flow can be piped along the (s,t)-arc with no cost. The transformed graph contains $n+2$ nodes and $n+m+1$ edges and thus its size is linear in the size of G. For a flow ϕ we denote its cost by $c(\phi) = \sum_{(u,v)\in N} \phi(u,v) \cdot c(u,v)$.

We show that solving the min cost flow problem yields an optimal solution to the WDCES problem, we omit the proof due to space constraints.

Lemma 1. *Let $G = (V_1 \cup V_2, E)$ be a bipartite instance of the WDCES problem and let N be the flow network constructed as above and let ϕ be an optimal solution of N that is integral. Then the edge set $E' = \{uv \in E \mid u \in V_1, v \in V_2, \phi(u,v) = 1\}$ is an optimal solution of the WDCES problem.*

Using an algorithm by Goldberg and Tarjan [5], this MINCOSTFLOW problem can be solved in time $O(nm\log^2 n)$ (note that the maximum absolute cost of our instance is 1), thus the running time for Phase 2 is in $O(nm\log^2 n)$. Further, in Phase 1 we do not require an optimal solution, instead we simply check whether a feasible solution exists using a MaxFlow algorithm with a running time of $O(nm\log n)$ [4]. The final step consists of edge-coloring the bipartite graph resulting from the previous phase, which requires $O(m\log n)$ time, using the algorithm by Cole and Hopcroft [2]. The following theorem summarizes our findings.

Theorem 2. *Let (G,q,ℓ,h,σ) be an instance of bipartite* SPEEDDATING *such that G has n vertices and m edges. An optimal solution can be computed in $O(nm\log^2 n)$ time.*

4 General Speed Dating

As we have seen in the previous section, the bipartite speed dating problem admits a polynomial-time algorithm. The crucial observation was that for bipartite graphs the third phase, consisting of coloring the graph with at most σ colors, can be carried out independently from the result of the previous two phases. In the general case an optimal solution to the first two phases is an edge set E' that in general no longer induces a bipartite graph and therefore may not admit a coloring using only σ colors. Hence, even if we find a solution to the WDCES problem of Phase 2 (the above reduction to a flow problem relied on bipartiteness), the problem arising in the third phase is not necessarily solvable.

Suppose that G' is any subgraph determined in the first two phases. Although G' cannot necessarily be colored with σ colors and it is NP-complete to find out whether this is the case, by Vizing's theorem [7], we know that G' can be colored with at most $\sigma+1$ colors; we obtain a schedule that has one round too much. To remedy this, we introduce a new phase that simply removes the color class with the smallest total weight. In summary our algorithm works as follows.

1. Determine the minimum value δ such that an edge set E' with $\ell(v)-\delta \leq \text{dates}(v) \leq h(v)$ for all $v \in V$ exists, using a binary search.
2. Determine the maximum weight edge set E' of G with $\ell(v)-\delta \leq \text{dates}(v) \leq h(v)$ for all $v \in V$ for the fixed value of δ determined in the previous phase.
3. Color the edge set determined in the second phase with at most $\sigma+1$ colors.
4. If the previous phase uses $\sigma+1$ colors remove the edges of the lightest color.

Phase 1 works in the same way as in the bipartite case. Phase 2 can be reduced as in the bipartite case to WDCES. The reduction to MINCOSTFLOW however breaks, because the graph is not necessarily bipartite. To solve general WDCES we make use of an exact polynomial-time algorithm developed by Gabow [3], which runs in time $O(n^2\sigma^2\Delta^3 \log n)$. We avoid an additional factor of $\log n$ for Phase 1 by neglecting the weights during the binary search, which, using the same reduction, allows for a faster algorithm with running time $O(nm\Delta + n^2\Delta^2)$. Phase 3 can be solved using Vizing's algorithm with running time $O(nm)$ [6]. Phase 4 sums up the weights of the edges of each color and removes the edges of the lightest color if necessary, which requires $O(n+m)$ time. The total running time for the algorithm therefore is $O(n^2\sigma^2\Delta^3 \log n)$. We now show that the algorithm gives provable performance guarantees on the quality of the solutions.

If Phase 3 succeeds with coloring the resulting graph using at most σ colors, Phase 4 does not remove any edges and since the first two phases are solved optimally, the algorithm calculates an optimal solution in this case. For the sake of deriving worst case performance guarantees we therefore assume that this is not the case and that in Phase 4 all edges of the lightest color are removed. We have the following lemma.

Lemma 2. *Let* (G,q,ℓ,h,σ) *be an instance of* SPEEDDATING. *Let* $G_{\text{OPT}} = (V, E_{\text{OPT}})$ *be an optimal solution and let* $G' = (V,E')$ *be the solution computed by the above algorithm. Then* $\delta(G') \leq \delta(G_{\text{OPT}})+1$ *and* $q(G') \leq \frac{\sigma}{\sigma+1} q(G_{\text{OPT}})$.

Proof. Let E_0 be the edge set that was computed in Phase 2 of the algorithm during the execution that resulted in G'. We denote by G_0 the graph (V,E_0).

We first bound the fairness violation. Since E_0 is an optimal solution to Phases 1 and 2, we have $\delta(G_0) \leq \delta(G_{\mathrm{OPT}})$. Moreover, since E' results from E_0 by removing a matching we have $\delta(G') \leq \delta(G_0) + 1$. Together with the previous inequality this yields $\delta(G') \leq \delta(G_{\mathrm{OPT}}) + 1$.

For the overall quality consider again that E_0 is an optimal solution to the first two phases and hence $q(G_0) \geq q(G_{\mathrm{OPT}})$. Recall that G' results from G_0 by removing the edges of the lightest color, denote them by E_{light}. By the pigeon-hole principle we have $q(E_{\mathrm{light}}) \leq q(E_0)/(\sigma+1)$. For the overall quality of G' we thus get $q(G') = q(G_0) - q(E_{\mathrm{light}}) \geq q(G_0) - \frac{q(G_0)}{\sigma+1} = \frac{\sigma}{\sigma+1} q(G_0) \geq \frac{\sigma}{\sigma+1} q(G_{\mathrm{OPT}})$. This concludes the proof of the lemma. □

The following theorem summarizes the results of this section.

Theorem 3. *Let (G, q, ℓ, h, σ) be an instance of* SPEEDDATING. *Let G_{OPT} be an optimal solution. A solution G' with $\delta(G') \leq \delta(G_{\mathrm{OPT}}) + 1$ and $q(G') \geq \sigma/(\sigma+1) \cdot q(G_{\mathrm{OPT}})$ can be computed in $O(n^2 \sigma^2 \Delta^3 \log n)$ time.*

Note that for realistic values of σ, e.g., $\sigma = 12$, we have $\sigma/(\sigma+1) \geq 0.92$, i.e., the solution quality is at least 92% of the optimum. While the algorithm is favorable in terms of fairness and quality, its worst-case time complexity is quite high. We therefore also propose a greedy algorithms for the general speed dating problem, which may be advantageous in terms of running time.

We note that the running times of the algorithm for the bipartite case as well as of the approximation algorithm can be improved to $O(n\sigma \min\{m \log n, n^2\})$ using a more sophisticated reduction by Gabow [3]. However, this algorithm is very complicated, as it requires to modify a weighted maximum matching algorithm to dynamically manipulate the graph it is working on during the execution. We therefore chose to present the running times that more closely match the complexity of the implementations we are going to evaluate.

A Greedy Strategy for General Speed Dating. Let (G, q, ℓ, h, σ) with $G = (V, E)$ be an instance of SPEEDDATING. The algorithm GREEDY maintains a set S of edges that are colored with σ colors. In the course of the algorithm edges are only added to S and never removed. Moreover, every edge is colored upon insertion with one of the σ colors such that the graph (V, S) is properly σ-edge-colored and no edge in S ever changes its color. To pick the next edge, the algorithm picks the edge uv with the highest valuation $k(uv)$, where k is some formula computing the value of an edge. We use $k(uv) = w_{\max} \max\{\delta_S(u), \delta_S(v)\} + q(uv)$, where $w_{\max} = \max_{e \in E} q(e)$, i.e., it priorizes edges that are incident to a vertex whose fairness constraints are strongly violated in the current solution, the quality is a second criterion. The algorithm iteratively finds the edge with the highest value $k(uv)$ and either adds it to S, if this is possible, i.e., if $\mathrm{dates}_S(u) \leq h(u) - 1$ and $\mathrm{dates}_S(v) \leq h(v) - 1$ and the endpoints u and v have a common unused color, which then becomes the color of uv. If uv cannot be added, it is discarded and removed from E. The algorithm stops when $E = \emptyset$. The algorithm can be implemented to run in $O(m\Delta \log m)$ time.

5 Evaluation

We implemented the algorithms described above in C++. Our implementations use the LEMON 1.2 library[1], which provides efficient implementations of MinCostFlow and weighted maximum matching algorithms. All our experiments were run on one core of a computer with an Intel Q6600 processor with 4MB cache and 2GB RAM. As compiler we used g++ version 4.4.1 with compiler flags `-DNDEBUG` and `-O3`.

Problem Instances. We evaluate our algorithms on a variety of problem instances. We use four types of instances. The first two are produced by random generators that produce randomly filled-out forms and build graphs based on the similarity of these forms. Additionally, we evaluate our algorithms on social networks. The reason for this is that instances to SpeedDating are graphs that model the probability that people like each other. It is to be expected that such graphs do not strongly deviate from graphs modeling that people actually know or like each other, i.e., social networks. We therefore use the small world generator by Brandes et al. [1] to produce instances. Additionally, we use twelve instances of a real-world social network, stemming from the email network of the Faculty of Informatics at the Karlsruhe Institute of Technology.

In all cases, we set the number of rounds σ to 12. Assuming that each date lasts ten minutes, this implies that the whole event lasts at least two hours, which appears to be a realistic total duration. In the general case we always set $\ell(p) = h(p) = \min\{12, \deg(p)\}$ for all persons p in the instance. For the bipartite case we choose the genders randomly with a change of 40% for being a woman. Let N be the number of men and M the number of women. We set $\ell(w) = h(w) = \min\{12, \deg(w)\}$ for all women w. For all men m we set $\ell(m) = \lfloor \sigma M/N \rfloor$ and $h(m) = \lceil \sigma M/N \rceil$ to evenly distribute the dates.

For the generation of the edges and their weights, recall that each participant fills out a form about himself and his ideal partner before the speed dating event. Our random instance generators are based on randomly generating the contents of such forms and then computing for each possible pairing of two persons x and y a *score* for the edge xy, based on their forms. If the score of an edge xy is non-positive, we discard the pairing so that the resulting graph is not necessarily complete. For the bipartite case, we restrict the considered pairings to pairs of a man and a woman.

The first generator models the forms using random vectors of size 30, each entry drawn uniformly at random from $\{0/5, \ldots, 5/5\}$. The score of a pair xy is based on the Euclidean distance of their corresponding vectors v_x and v_y. We compute the weight of the edge xy as $\lfloor(\sqrt{30}/2 - |v_x - v_y|_2)\rfloor$ and discard it, if it is not positive. Note that the weight is positive about 50% of the time.

The second generator tries to model more complex forms. For each person we generated a random form composed of 30 questions about what traits the ideal partner should have. For each question q every person p answers on a scale of 0 to 5 how well she fulfills these traits ($\mathrm{has}_q(p)$) and how the partner should fulfill it ($\mathrm{wants}_q(p)$). Further each person must indicate for each question the level of importance on a scale of 0 to 5 ($\mathrm{imp}_q(p)$). Again, we assume that all choices are uniformly distributed.

[1] Available at `http://lemon.cs.elte.hu/trac/lemon`

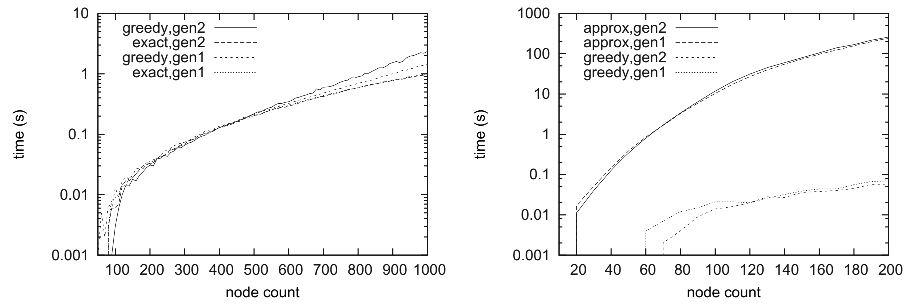

Fig. 1. Comparison of the running times of the exact algorithm and the greedy algorithm for generators 1 and 2 on bipartite instance (left) and comparison of the running times for the approximation algorithm and the greedy heuristic on instances of the general speed dating problem (right).

We define a function that estimates whether x likes y as $\text{likes}(x,y) = \alpha \cdot \sum_{q=1}^{30} \text{imp}_q(x) - \sum_{q=1}^{30} \left|\text{has}_q(y) - \text{wants}_q(x)\right| \cdot \text{imp}_q(x)$. The first term acts as a threshold telling how demanding the person x is. The second term evaluates how close y is to the profile x would like to have for his partner, weighted by the importance that x gives to each of the traits. Since the score function we use should be symmetric, we set $\text{score}(xy) = \text{likes}(x,y) + \text{likes}(y,x)$.

The constant α is a tuning parameter that affects the density of the resulting graph. For $\alpha = 2$ the generator produces almost complete graphs, for $\alpha = 3$ the graphs are extremely sparse. We choose $\alpha = 2.3$ for our experiments, as this generates plausible, relatively dense graphs.

As instances from social networks, we use graphs generated according to the small world model, generated by the generator of Brandes et al. [1]. The parameters we use for the generator are the following. We generate instances with 200 nodes and set the rewiring probability $p = 0.2$. The minimum degree k ranges from 10 to 100 in steps of 2. For each value of k we use five random samples to even out random influences. The edge weights are integers, chosen uniformly at random in the range from 1 to 20.

Finally, we use real-world social networks stemming from the email network of the Faculty of Informatics at the Karlsruhe Institute of Technology. For each month from September 2006 to August 2007 the corresponding graph contains all people of the network and every pair is connected by an edge whose weight is the number of emails they exchanged.

Experiments. We first present the results on instances produced by our form based generators. In our plots we indicate which generator was used by either gen_1 or gen_2. For the bipartite speed dating problem, we generate instances with n persons, where n ranges from 100 to 1000 in steps of size 10. The approximation algorithm is much slower, due to the different reduction that is necessary in Phase 2. We therefore take only instances of sizes 10 up to 200, again in steps of size 10. To remedy outliers, we average the results over ten samples of each size in both cases. We also run the greedy algorithm on all the instances.

Figure 1 shows the running times of our algorithms both in the bipartite and in the general case. Surprisingly, in the bipartite case, the greedy algorithm is constantly

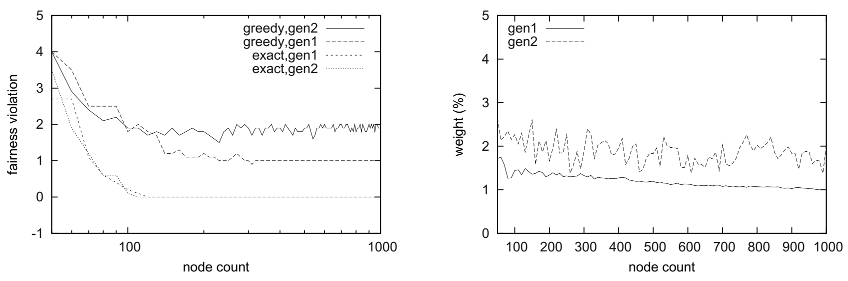

Fig. 2. Solution quality of the solutions produced by the exact solver and the greedy algorithm for bipartite instances generated by gen_1 and gen_2

slower than our exact solution algorithm although it has a far better worst case complexity. This is probably due to the fact that we did not take a lot of care of micro-optimizing the inner loops of the greedy algorithm whereas the inner working of the MINCOSTFLOW solver provided in the LEMON library are carefully tuned. All four curves suggest a quadratic growth in the number of people. As there are quadratically many potential dates this is linear in the size of the input.

For the general case, the situation is quite different. While for moderate sizes of around 100 people, an approximate solution can still be computed in less than two minutes, this quickly grows to roughly 10 minutes for events with 200 people. Expectedly, the greedy algorithm grossly outperforms the approximation algorithm in terms of running time and solves all instances within less than 0.1 seconds. As we will see later, the running time of the approximation algorithm is much better for sparser instances. Nevertheless, even for dense graphs the approximation algorithm needs less than ten minutes in the worse case tested, which is still within the acceptable bounds given by the problem motivation.

Next, we compare the solution quality of our algorithms. We plot the δ values of all solutions. Since the absolute quality value does not really have a means of interpretation, we only show the relative difference of the greedy solution and the solution computed by the exact and the approximate algorithm, respectively. A value of 3% indicates that the weight of the solution computed by the greedy algorithm is 3% higher than the solution computed by the exact algorithm (for bipartite instances) or the approximation algorithm (for general instances). A negative percentage indicates that the weight of the solution computed by the greedy algorithm is smaller.

Fig. 2 shows the performance of the greedy algorithm on bipartite instances with respect to quality. For the fairness violation δ, note that except for the very small instances, the greedy algorithm misses the optimal δ value usually by 1 or 2. Considering that there are only 12 rounds and that minimizing the fairness violation is our main optimization criterion, being off by 2 units is rather bad. In terms of overall quality, the greedy algorithm performs quite well and is never off the true value by more than 5%, which is acceptable as it is unlikely that participants would notice this.

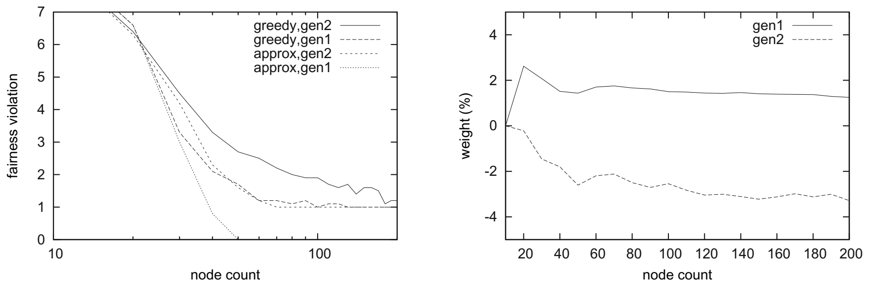

Fig. 3. Solution quality of the solutions produced by the approximation algorithm and the greedy algorithm for general instances of both generators

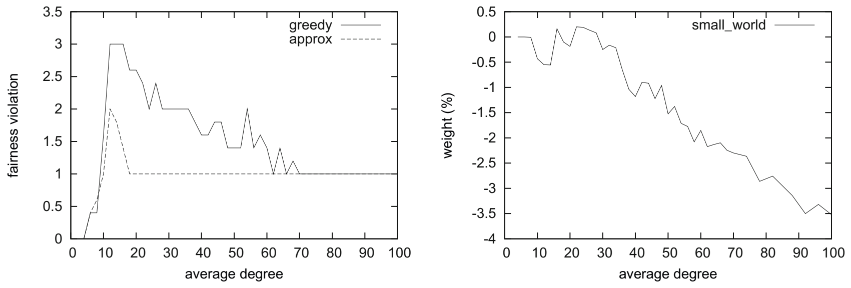

Fig. 4. Comparison of the greedy and approximation algorithm on small world graphs of varying density

Fig. 3 shows the corresponding evaluation for the general case. The greedy algorithm performs quite well, for instances of the first generator it misses the optimal value by only 1 for smaller instances and finds an optimal value for larger instances. For the second generator, the approximation algorithm does not find optimal solutions and starting from a certain size, always finds solutions with fairness violation 1. Since this fairness violation stems from discarding some edges and the graphs are rather dense, it is not unlikely that the optimal fairness violation may be 0. The greedy algorithm performs comparably, is slightly worse for small instances and achieves the same fairness violation as the approximation for most larger instance sizes. Interestingly, the greedy algorithm outperforms the approximation algorithm for instances from gen_2 in terms of quality by up to 5%.

Figure 4 shows the results of the experiments on small world graphs. Note that for this experiment the number of vertices is fixed to 200 and the average degree varies from 4 to 100. The approximation algorithm performs consistently better than the greedy algorithm. Only for very dense graph does the greedy algorithm find solutions with the same δ but with less weight. Moreover, the running time of the approximation algorithm is consistently below one second. Finally, the results on the email networks are shown in Table 1. Again the approximation algorithm is favorable in terms of δ but slightly worse than greedy in terms of total quality. Moreover, this shows that the approximation algorithm is extremely fast on sparse instances.

Table 1. Performance of our algorithms on instances stemming from the email network at the Karlsruhe Institute of Technology. For time and δ, the first value is for the approximation algorithm, the second for the greedy algorithm. The quality column shows the difference between the greedy and the approximate solution, relative to the approximate solution quality.

month	n	m	time [s]		δ		q [%]
1	59	2994	1.86	0.12	3	4	+2.1
2	431	1898	0.37	0.04	2	5	+2.8
3	444	1729	0.31	0.04	2	3	+3.7
4	432	1660	0.32	0.04	2	4	+5.1
5	450	1846	0.43	0.06	3	3	+0.2
6	430	1815	0.44	0.03	3	4	+0.9
7	440	1906	0.47	0.06	3	4	+2.1
8	405	1563	0.24	0.05	2	3	+2.8
9	432	1633	0.28	0.04	2	3	+0.7
10	558	2180	0.67	0.07	3	4	+0.2
11	504	1994	0.42	0.06	2	4	+5.5
12	881	3427	4.71	0.16	3	5	+7.1

Conclusion. Our experimental evaluation shows that for the bipartite speed dating problem, our exact polynomial-time algorithm is the preferred solution. It produces exact results, is very fast and solves even problem instances with 1000 people within a few seconds and therefore is the method of choice for these instances. For the general speed dating problem the situation is not that clear. The approximation algorithm takes a few minutes to find solutions for large, dense instances and the quality of solutions found by the greedy algorithm is in many cases comparable in this case. To maintain the theoretical guarantees the best solution for this case seems to be to run both algorithms and to pick the better solution. If execution time is crucial, the greedy algorithm is the algorithm of choice for these instances as it is very fast and likely to produce solutions of high quality. For sparser instances the approximation algorithm is both fast and gives better results than greedy.

Acknowledgments. We thank Robert Görke for helpful discussions and providing us with the email networks we used for evaluation.

References

1. Batagelj, V., Brandes, U.: Efficient generation of large random networks. Physical Review E 036113 (2005)
2. Cole, R., Hopcroft, J.: On edge coloring bipartite graphs. SIAM J. Comput. 11(3), 540–546 (1982)
3. Gabow, H.N.: An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In: Proc. 15th Annu. ACM Sympos. Theor. Comput(STOC 1983), pp. 448–456. ACM, New York (1983)
4. Goldberg, A., Tarjan, R.E.: A new approach to the maximum flow problem. J. Assoc. Comput. Mach. 35, 921–940 (1988)
5. Goldberg, A., Tarjan, R.E.: Finding minimum-cost circulations by canceling negative cycles. J. Assoc. Comput. Mach. 36, 873–886 (1989)
6. Misra, J., Gries, D.: A constructive proof of Vizing's Theorem. In: Inf. Proc. Let., pp. 131–133 (1992)
7. Vizing, V.G.: On an estimate of the chromatic class of a p-graph. Diskret. Analiz, pp. 25–30 (1964) (in Russian)