Karlsruhe Institute of Technology

SpeedDating: An Algorithmic Case Study involving Matching and Scheduling

Study Thesis of

Ben Strasser

At the faculty of Computer Science
Institute of Theoretical Informatics
Algorithmics I

Advisors: Dr. rer.-nat. Bastian Katz
Dipl.-Inform. Ignaz Rutter
Prof. Dr. Dorothea Wagner

Processing Time: 10. May 2010 – 19. August 2010

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und die vorgestellten Ergebnisse ohne die Hilfe Dritter erarbeitet habe. Ich habe auf keine anderen als die angegebenen Quellen und Hilfsmittel zurückgegriffen.

Ben Strasser
Karlsruhe, den 19. August 2010

Contents

1 Introduction

A speed dating event is an event where a number of people come together to date each other for a short period of time. As dates can only involve two persons it is not possible that everybody dates everyone at the same time. It is usually done by scheduling dating rounds such that in each round every person dates at most one other person.

Consider a group of six men and six women that attend such an event. To make sure that every man dates every woman and vice versa we need six rounds. Suppose that a round lasts ten minutes then the whole event will last at least one hour. Most likely it will last longer as people need to change seats to get to the next date. With such a small group of people we can simply pair everyone with everyone. However, in larger groups such as 200 men and 200 women, the event would last about one and a half day. Hence, the organizers of such events need to make a selection of dates in advance to limit the number of rounds.

One approach to tackle this problem is to make the participants fill out forms about their ideal partner before the event. Based on this information it is possible to estimate in advance how well a certain date would work. We call this the *quality* of the date. We now select the pairings so that the overall quality is maximized and no person has more dates than the maximum number of rounds. This way we can cap the number of rounds to say twelve and the whole event will then only last between two and three hours.

There are however a couple of problems with this approach.

- Certain persons are more attractive than others and would get all of the dates leaving none for the less attractive ones. Distributing the dates in a fair way therefore is important.
- Generally there are more men than women on such events. The only way to manage this is to make every man miss a couple of turns.
- Some persons, which we call *VIP-persons*, may be more important than others because of certain circumstances. They might for example have paid extra money. Even though we generally try to be fair in distributing dates these persons clearly must be preferred. We especially want to make sure that these persons do not have to miss turns.
- People may not show up at the event even though they registered. The problem with this is that all the people that were scheduled for this person will get one date less than planned if the planning can not be adjusted in time. The simplest way of solving this problem is simply not to do any calculations in advance. The plan should be generated just before the event starts. This way the event organizers know who showed up. As one can not force the people to show up several hours before the event takes place these calculations must be finished within a couple of minutes.

So far we have only described heterosexual speed dating but one may also consider homosexual dating. In this setting we do not make a distinction between men and women. Every person may date every other person. We can regard this as a generalization of the heterosexual dating problem, where the dates between persons of the same sex have a very bad quality. We show that this generalization is a lot harder to solve.

In this thesis we develop algorithms to generate the dating rounds based on the data collected using the filled forms. Each of these rounds consists of a list of pairings. We do not order these rounds in time nor do we try to optimize the time the people need to change their seats.

Selecting the dates is a matching problem and distributing them among the different rounds is a scheduling problem. There are a few additional non-standard constraints on the matching to make sure that the dates are selected in a way that is fair for every person. We investigate whether these two subproblems may be solved separately. It turns out that this is not possible when considering homosexual dating.

After we recalled a few basic definitions in Section 2 we formalize the generalized homosexual problem setting and classify it in Section 3. It turns out that the problem is NP-hard. We formulate the problem in graph-theoretic terms and as an *integer linear program* (ILP). Afterwards we develop a polynomial-time algorithm for the heterosexual special case in Section 4. We then try to generalize this algorithm and obtain a polynomial-time approximation algorithm with good performance guarantees in Section 5. This done, we investigate a couple of greedy heuristics to solve the problems in Section 6. Finally we evaluate the performances of the algorithms described by implementing them and applying them to random test data in Section 7.

2 Definitions

We will use mostly standard notation for graph theory as defined by e.g. Diestel [5]. The most important difference is that the induced subgraph of an edge set contains all nodes and not only those that are an end of some edge. We recall the most important terms.

A *graph* G is a tuple (V, E) where V is a finite set called the set of *nodes* and E is a subset of $\binom{V}{2}$ called the set of *edges*. We denote the number of nodes with n and the number of edges with m. The *degree* $d(v)$ of a node v is the number of incident edges, i.e. , $d(v) = |\{e \in E \mid v \in e\}|$. The *maximum degree* over all nodes is denoted by $\triangle_G$ i.e. $\triangle_G = \max_{v \in V} d(v)$. A *subgraph* of a graph $G = (V, E)$ is itself a graph $G' = (V', E')$ such that $V' \subseteq V$ and $E' \subseteq E$. The degree of a node in a subgraph is denoted by $d'(v)$. Given an edge set E' the graph (V, E') is called the *induced subgraph*. A *directed graph* is a tuple (V, A) where V is a finite node set and $A \subseteq V \times V \backslash \{(v, v) \mid v \in V\}$ is the set of *arcs*.

A *matching* is an edge subset such that each node has at most degree one in the induced graph. A *perfect matching* is a matching where each node has a degree of exactly one. A matching is *maximum* if no matching exists that contains more edges.

An *edge coloring* is a function that maps E onto a set of colors C such that all edges incident to the same node are colored differently. We identify the colors with the numbers $\{1, 2, \ldots, |C|\}$. A node *misses* a color if it has no incident edge of that color. The set of natural numbers $\mathbb{N}$ starts with zero, i.e., $\mathbb{N} = \{0, 1, 2, \ldots\}$.

3 Problem description

This section formalizes the homosexual dating problem and its heterosexual special case described in the introduction. We first provide a problem statement in graph-theoretic terms. Then we model this problem as an ILP and we show that the general Speed Dating problem is NP-complete.

3.1 Formal Specification

The people are represented using nodes and the potential dates as edges. We discard potential dates with a very bad quality leading to a graph that is not necessarily complete. We refer to dates with such a bad quality as *bad dates*. The graph $G = (V, E)$ obtained in this way is simple and undirected. It does not have to be connected. The heterosexual special case provides a bipartite graph structure since no woman dates any woman and no man dates any man.

We formalize the quality of a potential date using a non-zero natural number. A high value describes a high quality whereas a value close to 1 means a second-rate date. This makes the graph weighted. VIP-persons can be given a boost by augmenting the weight on their outgoing edges. One goal is to maximize the sum of the weights of the selected dates. Another goal is to distribute the dates in a fair way.

To make sure that no person gets no dates just because he or she is unattractive, we set a lower and an upper bound called ℓ and h for the number of dates that each person gets. Each person v has at most $h(v)$ dates and if possible at least $\ell(v)$ dates. In most situations one would set $h(v) = \ell(v)$ but differentiating between both allows for more flexibility. Consider for example a dating event with w women, m men, σ rounds and heterosexual dating. Let us recall that there are less women than men. Then one could set $\ell(v) = h(v) = \sigma$ for each woman resulting in a total of wt dates. As this number can not always evenly be divided between all the men it therefore makes sense to set $\ell(v) = \lfloor wt/m \rfloor$ and $h(v) = \lceil wt/m \rceil$. Figure 1 shows an example. VIP-persons may be given a higher value for $\ell(v)$ than other people. As one can see it makes sense to choose different bounds for different people and in some cases even $h(v) \neq \ell(v)$ makes sense.

Our primary objective is to select the dates in such a way that the ℓ and h bounds are met. This is however not always possible. If this is not the case then we lower ℓ for each person by 1e until a solution exists. We call the value by which we have to lower ℓ the *fairness violation* δ. In the following ℓ always refers to the fixed optimal lower bound given as input and $\ell - \delta$ is the actual lower bound used. The smaller the violation δ is the better.

Each person v has $d(v)$ potential dates and $d'(v)$ dates that actually take place. As a consequence the following inequality holds:

$$0 \leq \ell(v) - \delta \leq d'(v) \leq h(v) \leq d(v)$$

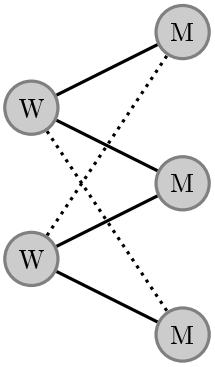


Figure 1: A possible selection of dates for two women, three men and two rounds. The solid lines are the selected dates. No matter what dates we select there will always be one man with two dates and two men with one date.

We can interpret $d(v)$ as the degree of v in the given graph and $d'(v)$ as the degree in the graph induced by the selected dates. We can regard the search for these dates as a search for a subgraph $G' = (V, E')$ of G that fulfills the given requirements on the node degrees.

A further restriction is that it must be possible to distribute the selected dates onto the different rounds. In graph-theoretic terms this means that we must be able to color the selected dates using at most σ colors, where σ is the number of turns. It turns out that this is the requirement that causes the homosexual version to be NP-hard.

Among the solutions that satisfy the conditions formulated and that have an optimal fairness violation we choose the solution with the maximum total weight. Now that we have formalized the different parts of the problem we are ready to state the *speed dating problem*.

Problem 1. Given a graph $G = (V, E)$ with edge weights $w : E \to \mathbb{N}\backslash\{0\}$ and lower and upper node capacities $\ell, h : V \to \mathbb{N}$ such that for each vertex v the inequality $\ell(v) \leq h(v) \leq d(v)$ holds, determine an edge subset colored with at most $\sigma = \max_{v \in V} h(v)$ colors. The degree $d'(v)$ of each vertex v in the induced subgraph should be smaller than $h(v)$. The subgraph should be optimal with regard to the following two criteria:

- Maximize the sum of the weights of the edges in the subgraph. We refer to this as the *weight criterion*.
- Minimize the maximum difference δ between $\ell(v)$ and $d'(v)$. We call this difference fairness violation. More formally
$$\delta = \max(0, \max_{v \in V} \ell(v) - d'(v))$$
We refer to this criterion as the *fairness criterion*.

It is possible to look at this problem as a bicriterial one. However we first minimize the fairness violation δ. Among the smallest δ we choose the solution that maximizes the weight. It is therefore always possible to determine whether a solution is better, worse or just as good as another one.

3.2 Integer Linear Program

The speed dating problem can be formulated as an integer linear program (ILP). We introduce binary decision variables $x_{e,c} \in \{0, 1\}$ that model the question if an edge e is colored using the color c. If $x_{e,c} = 1$ then the edge e is colored using c. Further, we introduce two variables δ and s representing how well the fairness and weight criteria are met. Let us recall that C denotes the set of colors. The constraints of the program are:

$$\sum_{\{u,v\}\in E} x_{\{u,v\},c} \leq 1 \qquad \forall u \in V, c \in C \tag{1}$$

$$\sum_{c\in C} x_{e,c} \leq 1 \qquad \forall e \in E \tag{2}$$

$$\sum_{\substack{\{u,v\}\in E \\ c\in C}} x_{\{u,v\},c} \leq h(u) \qquad \forall u \in V \tag{3}$$

$$\sum_{\substack{\{u,v\}\in E \\ c\in C}} x_{\{u,v\},c} \geq \ell(u) - \delta \qquad \forall u \in V \tag{4}$$

$$\delta \geq 0 \tag{5}$$

$$s = \sum_{\substack{e\in E \\ c\in C}} w(e)x_{e,c} \tag{6}$$

The constraint (1) ensures that all edges incident to the same node are colored differently. The equation (2) makes sure that every edge is colored using at most one color, i.e., that no date takes place more than once. An edge may be uncolored meaning that it is not selected. Inequality (3) ensures that the upper degree bound requirement is met for every node.

The constraints (4) and (5) measure how well the fairness criterion is met. Equation (6) does the same for the weight criterion. These three last constraints are only needed to be able to formulate the objective function. They do not influence which edge subsets are considered to be valid.

We can formulate the objective function using a large constant $\hat{s}$.

$$s - \delta\hat{s} = \max!$$

Since $\hat{s}$ is needed to make sure that the fairness criterion dominates the weight criterion it must be at least as big as any value that s can possibly take. The total weight s is maximum if the subgraph is the whole graph G, which leads us to a sufficient lower bound for $\hat{s}$, namely

$$\hat{s} \geq 1 + \sum_{\{u,v\}\in E} w(\{u,v\}) \ .$$

In Section 7 we evaluate the speed of a speed dating solver that is composed of this ILP problem formulation and an off-the-shelf ILP solver. It is largerly outperformed by the solvers described in Sections 4 and 5 which make use of combinatorial approachs.

3.3 Classification

We have formalized the speed dating problem. A natural question to ask before trying to solve it is how hard it is. In this Section we prove that it is NP-complete.

Lemma 2. *The speed dating problem is NP-hard.*

Proof. Determining whether a graph $G = (V, E)$ can be colored with exactly $\triangle_G$ colors is NP-hard [7]. Given an instance of this graph coloring problem we set $w(e) = 1$, $\ell(v) = 0$ and $h(v) = d(v)$ for all $e \in E$ and $v \in V$ and obtain an instance of the speed dating problem with the property that $\sigma = \triangle_G$. This transformation can be computed in polynomial time. Solving it yields a colored edge subset E'. As $\ell(u) = 0$ the fairness criterion is always satisfied optimally and can be ignored. As all edges have the same weight the speed dating problem tries to color as many edges as possible. This means it determines a largest edge subset E' that can be colored using $\triangle_G$ colors. A valid coloring of G exists if and only if $E = E'$. This implies that the speed dating problem is NP-hard. □

Another natural question to ask is how hard the problem is.

Lemma 3. *The decision problem corresponding to the speed dating problem is in NP.*

Proof. We have to show that given two positive integers δ, s and an instance of the speed dating problem one can determine whether a solution exists that is at least as good as δ with regard to the fairness criterion and as s with regard to the weight criterion using only a deterministic Turing machine with an oracle. First, the oracle guesses the solution and then the machine verifies in polynomial time whether it meets the requirements. Given a solution we can determine how good the fairness criterion is met by calculating the solution's δ using the formula given in the definition of the problem and comparing it to the given δ. The weight criterion can be tested in a similar way by summing up the weights of the edges in the subgraph. □

Combining Lemmas 2 and 3 shows that the speed dating problem is NP-complete. A consequence of this is that unless $P = NP$ it is not possible to formulate the problem as a linear program.

4 Algorithm for Heterosexual Speed Dating

Before trying to solve the complete problem formulated in the previous section we first solve the heterosexual special case. Let us recall that in graph-theoretic terms this means that the input graph is bipartite. Our algorithm works in three phases.

1. Determine the minimum fairness violation δ using a binary search.
2. Determine the maximum weight uncolored edge subset E' for a fixed δ. Denote with G' the induced graph.
3. Color the edge set determined in the second phase using $\triangle_{G'}$ colors.

A key observation is that every bipartite graph can be edge colored efficiently using $\triangle_{G'}$ colors. We describe in Section 4.3 an algorithm that achieves this. As a consequence we can compute the Phases 1 and 2 completely independently from Phase 3. This degree of independence is however not given between the first two phases. Phase 1 makes use of the algorithm developed in Phase 2 to check whether a solution exists given a fixed δ. Note that the solution is valid because $\triangle_{G'} \leq \max_{v \in V} h(v) = \sigma$

4.1 Phase 1: Determine the Minimum fairness violation

Given a fixed fairness violation δ we can transform the problem instance into one where the lower node capacities are tight bounds meaning that $\ell(v) \leq d'(v)$ must hold for each node v. The transformation is

$$\ell(v) \leftarrow \max\{0, \ell(v) - \delta\}$$

Once transformed the problem of finding an uncolored edge set boils down to the *weighted degree constrained edge subset problem* (WDCES), which can be stated as follows.

Problem 4 (WDCES)**.** Given a graph $G = (V, E)$ with edge weights $w : E \to \mathbb{N}\backslash\{0\}$, lower and upper node capacities $\ell, h : V \to \mathbb{N}$ such that for each node v the inequality $\ell(v) \leq h(v) \leq d(v)$ holds, determine an edge set E' with maximum weight such that $\ell(v) \leq d'(v) \leq h(v)$ holds where $d'(v)$ is the degree of the node v in the induced subgraph.

This problem is also known as b-matching with unit edge capacities. In Phase 2 we describe an algorithm to solve this problem. Further a straightforward variation is described that tests whether any feasible uncolored edge subset exists. As the third phase can never fail it is sufficient to determine whether the speed dating problem with a fixed fairness violation δ has a solution. It is important to note that at this point we exchange the variable δ in the original problem setting with a fixed constant.

We make use of this variation to determine the minimum δ using a binary search. We know that δ is in the interval $[0, \max_{v \in V} \ell(v)]$ and that it is integral. Further, if no feasible solution exists for a certain δ then none exists for smaller values either. If a solution exists then it is also a solution for all bigger values of δ. These are all the ingredients needed to determine the optimal δ using a binary search. We guess a δ and apply the feasibility test from Phase 2 to determine if it is too small or big enough and adjust δ accordingly. See Algorithm 1 for some pseudo code showing the details of this algorithm.

Algorithm 1 Algorithm to determine the optimal δ using a binary search

$\delta_{\min} \leftarrow 0$;
$\delta_{\max} \leftarrow \max_{v \in V} \ell(v)$;
while $\delta_{\min} \neq \delta_{\max}$ **do**
  $\delta \leftarrow \left\lfloor \frac{\delta_{\min} + \delta_{\max}}{2} \right\rfloor$;
  for *every vertex* v **do**
    $\ell'(v) \leftarrow \max(\ell(v) - \delta, 0)$;
  end
  if *phase 2 is feasible using* ℓ' **then**
    $\delta_{\max} \leftarrow \delta$;
  else
    $\delta_{\min} \leftarrow \delta + 1$;
  end
end
The optimal δ is $\delta_{\min}$

4.2 Phase 2: Determine the Maximum Weight Subgraph

This phase basically consists of solving the WDCES problem that was formulated in Section 4.1. Since we only consider bipartite graphs we can reduce it to the *min cost flow problem* (MCF). Several versions of this problem exist. We will use the one described in the Lemon library [2] as we want to make use of its implementation.

Problem 5 (MCF)**.** Given a directed graph $D = (V_D, A)$ with arc costs $c : A \to \mathbb{R}$, lower and upper arc capacities $\ell_D, h_D : A \to \mathbb{R}$ such that $\ell_D(a) \leq h_D(a)$ for each arc a and a node supply $q : V_D \to \mathbb{R}$ find a flow $f : A \to \mathbb{R}$ such that

$$\sum_{a \in A} c(a) f(a) = \min! \tag{7}$$

$$\sum_{(u,v) \in A} f(u,v) - \sum_{(v,u) \in A} f(v,u) \geq q(u) \qquad \forall u \in V \tag{8}$$

$$\ell_D(a) \leq f(a) \qquad \forall a \in A \tag{9}$$

$$h_D(a) \geq f(a) \qquad \forall a \in A \tag{10}$$

It has been shown that if a solution exists and ℓ_D, h_D and q only take integral values then an optimal solution exists where f also only takes integral values [3]. Lemon provides an algorithm that solves this problem optimally. This algorithm finds an integral flow when c, ℓ_D and h_D are integral.

The supply model used here is more general than we will need. In our case we will have $q(v) \neq 0$ only for two special nodes in D called the *source node* s and the *target node* t. We set $q(s) = -q(t)$. The flow has an amount of $q(s)$, which we will denote by p. This self-imposed restriction allows us to rewrite (8) as the better known flow conservation equation.

$$\forall u \in V_D \backslash \{s,t\} : \sum_{(u,v) \in A} f(u,v) = \sum_{(v,u) \in A} f(v,u) \tag{11}$$

It basically means that flow can only be created or destroyed at the source and at the target, respectively. Equation (7) minimizes the cost the flow causes along its way. The capacity constraints (9) and (10) reflect the node capacities of the weighted degree constrained subgraph problem.

Min Cost Flow Reduction

Given a WDCES instance we can construct an equivalent MCF instance based on it. Let D be a directed graph whose nodes are the same as G and additionally two new special nodes called s and t. As G is bipartite we can partition V into two disjoint node sets L and R such that each edge is incident to one node in L and one in R. For each edge $\{u,v\}$ in G with $u \in L$ and $v \in R$ we introduce an arc from u to v with $c(u,v) = -w(\{u,v\})$, $\ell_D(u,v) = 0$ and $h_D(u,v) = 1$. We refer to these arcs as the *main arcs* all other arcs are *helper arcs*. We will identify the arc (u,v) with

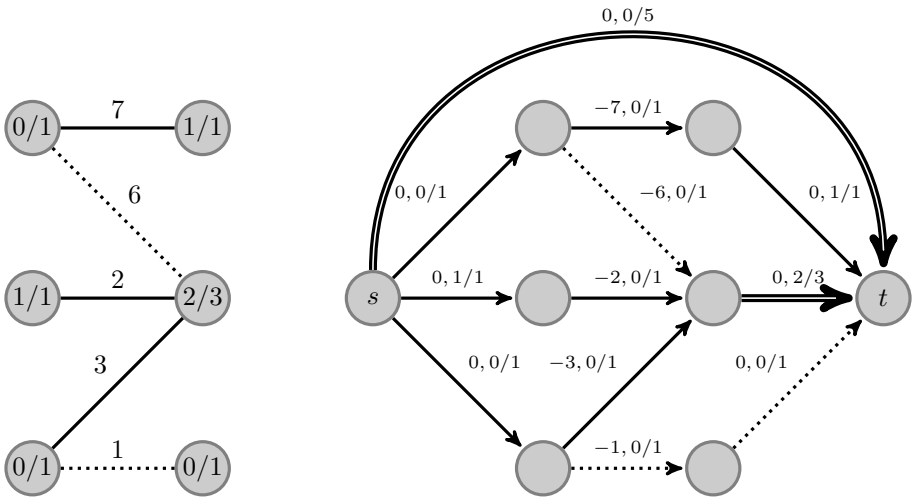


Figure 2: A bipartite weighted constrained subgraph instance G and its transformation to a min cost flow instance D. The labels of the nodes in G represent its capacity in the form $l(v)/h(v)$. The labels at the edges in G are their weight. The labels at the arcs of G' represent $c(e), l'(e)/h'(e)$. The solid edges in G represent the edges included in the subgraph. The dotted ones are not. In D the line style the amount of flow on that arc. Dotted means no flow. Solid means a flow of 1 and a double line means a flow of 2.

edge $\{u, v\}$. We add an arc from s to each node v in L, the helper arc of v, and set $\ell_D(s, v) = \ell(v)$ and $h_D(s, v) = h(v)$. For each v in R we add an arc, the helper arc of v, and set $\ell_D(v, t) = \ell(v)$ and $h_D(v, t) = h(v)$. All helper arcs have cost 0.

We set the desired flow value p to the number of edges in G. Further, we introduce an arc a from s to t with with $\ell_D(a) = 0$ and $h_D(a) = p$. The idea is to make sure that the source can produce enough flow and that superfluous flow can be piped along the (s, t)-arc with no costs. We could therefore also choose any value for p that is bigger as long as we make sure that $h_D(a) \geq p$ holds. The transformed graph contains $n + 2$ nodes and $m + n + 1$ arcs and thus has size linear in the size of G.

We apply an MCF algorithm to D. If a solution exists it consists of an optimal integral flow f. To obtain a solution of the WDCES problem we construct an edge subset E' that includes an edge e if $f(a)$ is 1, where a is the main arc that corresponds to e. Figure 2 shows an example of such a transformation. It remains to show that E' is valid and optimal and that when no feasible flow f exists then the WDCES instance is unsolvable.

Lemma 6. *If a feasible flow exists the corresponding edge set E' satisfies $\ell(v) \leq d'(v) \leq h(v)$ for each node v.*

Proof. For each $v \in L$ consider the corresponding helper arc $a \in A$. By construction $\ell(v) = \ell_D(a)$ and $h(v) = h_D(a)$ hold. As a is the only in-arc of v in D the flow conservation equality (11) boils down to $f(a) = \sum_{(v,u)\in A} f(v, u)$. The flow capacity constraint on a dictates that $\ell_D(a) \leq f(a) \leq h_D(a)$. Further, we constructed D in such a way that all out-arcs are main arcs. Let b be one of them then $0 \leq f(b) \leq 1$ must hold. As we know that f only takes integral values this forces $f(b)$ to be 0 or 1. This implies that $\sum_{(v,u)\in A} f(v, u)$ is the number of main arcs that carry a flow of 1 starting at v. Because of the way we constructed D this is also the degree of v in D. To sum up the following holds

$$\ell(v) = \ell_D(a) \leq f(a) = \sum_{(v,u)\in A} f(v, u) = d'(v) \leq h_D(a) = h(v) \ .$$

We can make a similar argument for the nodes in R. As $\ell(v) \leq d'(v) \leq h(v)$ is the only constraint of the WDCES problem E' is a valid solution. □

We have shown that E' is valid. It remains to show that it is optimal and that when no feasible flow f exists, then the WDCES instance is unsolvable.

Lemma 7. *If no feasible flow exists then G does not have an edge subset E' whose induced subgraph satisfies $\ell(v) \leq d'(v) \leq h(v)$ for each node v.*

Proof. Suppose that no flow but a valid solution E' to the WDCES problem exists. We can then construct a flow f as follows. For each edge $e \in E$ we set $f(e) = 1$ if $e \in E'$ and $f(e) = 0$ otherwise. It is clear that the capacity constraint for main arcs $0 \leq f(e) \leq 1$ holds. For each node $v \in V$ and the corresponding helper arc $a \in A$ we set $f(a) = d'(v)$. As $\ell(v) \leq d'(v) \leq h(v)$, $\ell_D(a) = \ell(v)$ and $h_D(a) = h(v)$ hold the capacity constraint $\ell_D(a) \leq f(a) \leq h_D(a)$ also holds. We set $f(s,t) = |E| - |E'|$. This is obviously within its capacity constraint $\ell_D(s,t) = 0 \leq f(s,t) = |E| - |E'| \leq |E| = h_D(s,t)$. It remains to show that the flow is conserved, meaning that (11) holds and that the total flow amount is equal to p.

The total flow amount t can be measured at the source.

$$\begin{aligned} t &= \sum_{(s,v)\in A} f(s,v) \\ &= f(s,t) + \sum_{v\in L} d'(v) \\ &= |E| - |E'| + |E'| \\ &= p \end{aligned}$$

Each node $v \in L$ has $d'(v)$ out-arcs with a flow of 1 and one in-arc with a flow of $d'(v)$. Therefore the flow is conserved. A similar argument works for the nodes in R. Hence the flow f is valid, which is a contradiction to the assumption that no feasible flow exists. □

Lemma 8. *An edge set E' constructed using the MCF reduction is optimal meaning that $\sum_{e\in E'} w(e)$ is maximum.*

Proof. The equality $\sum_{a\in A} c(a)f(a) = -\sum_{e\in E'} w(e)$ holds because $-w(e) = c(a)$ and $f(a) = 1$ if the corresponding edge is in E' and 0 otherwise. If $\sum_{a\in A} c(a)f(a)$ is minimum then $\sum_{e\in E'} w(e)$ must be maximum since otherwise a WDCES solution would exists with a bigger value. We could construct the corresponding MCF solution and obtain a better solution than the minimum value. This is of course not possible. □

The "Lemon" library also includes a faster algorithm that only finds a feasible solution to the min cost flow problem [2]. This means it basically does not care about the costs and ignores the objective function (7). We can therefore perform the feasibility test of the first phase using this faster algorithm.

4.3 Phase 3: Coloring the Edges

In the third phase we are given a bipartite graph $G = (V, E)$ with a maximum node degree of σ. Our goal is to color it using at most σ colors. We will describe an algorithm that uses a few ideas from Vizing's algorithm [9, 10] but manages to always color the graph. We iteratively color the edges and make sure that after each step the colored subgraph has a valid coloring. In order to Prove this we introduce the notion of coloralternating paths.

Definition 9. An *a-b*-coloralternating path starting at a node u is a path

$$u = v_0 \xrightarrow{a} v_1 \xrightarrow{b} v_2 \xrightarrow{a} \dots$$

such that the edge $\{v_i, v_{i+1}\}$ is colored using a if i is even and colored using b if i is odd.

Lemma 10. *Each a-b-coloralternating path is either a simple path or a cycle.*

Proof. The graph induced by the edges in the path is colored with only two colors namely a and b. As no node has more than one incident edge of a certain color each node in this graph has a maximum degree of 2. This means that this graph is a union of simple paths and simple cycles. As the graph is induced by a single path it is also connected. This means that it is either a simple path or a simple cycle.

Algorithm 2 Edge coloring a bipartite graph

```
for {u, v} ∈ E do
    c_0 ← color missed by u;
    c_1 ← color missed by v;
    i ← 0;
    while v does not miss c_i do
        Let {v, w} be the edge incident to v colored using c_i;
        Color {u, v} using c_i;
        i ← 1 − i;
        u ← v;
        v ← w;
    end
    Color {u, v} using c_i;
end
```

□

Lemma 11. *Given an edge subset E', an uncolored edge $e \in E \backslash E'$ and a valid coloring for (V, E') we can construct a valid coloring for $(V, E' \cup \{e\})$.*

Proof. Let u and v be the end nodes of e. As e is uncolored and incident to both u and v we can conclude that colors c_u and c_v must exist such that u misses c_u and v misses c_v. If $c_u = c_v$ then we simply color e using that color and we have found a valid coloring. If this is not the case then we consider the longest unique c_u-c_v-coloralternating path P starting at v. The previous lemma tells us that this is either a simple path or a simple cycle. If it was a simple cycle then v would be an inner node and therefore have an incident edge colored with c_v. It is therefore a simple path that ends in a node we denote by w. This node must miss either c_u or c_v as otherwise the path could be enlarged. Further we can show that $u \neq w$ using a contradiction with the bipartite graph structure. Suppose that $u = w$. In that case the first edge in the path must be colored using c_u and the last one using c_v. This is only possible if P contains an even number of edges. Adding e to P yields a cycle of odd length. In a bipartite graph this is however not possible. We may therefore swap all the colors along P without destroying the coloring or influencing u. After performing this operation v misses c_u and u still misses c_u. We can therefore color e using c_u. □

Using this algorithm we can iteratively color every edge of the bipartite graph. Algorithm 2 shows this. It remains to analyze how fast it is. The maximal length of a simple path is n. If it was necessary to construct such a path at each iteration we would touch nm nodes and therefore the running time of the algorithm is in $O(nm)$. Consider the graph that consists of only one long path. By coloring the edges from one end to the other we can at each iteration trigger a recoloring of every edge colored so far. The worst case running time is therefore in $\Theta(nm)$. A speed up can be achieved by first coloring the edges greedly and only once this is no longer possible to enlarge the colored edge subset using the method described above.

An algorithm with a worst case running time of $O(m \log n)$ exists [4]. We chose not to use it because that algorithm would have been more work to implement. Further, it will turn out that the edge coloring is not a bottleneck of the speed dating solver. The achieved speed up would therefore have been unnoticeable.

4.4 Analysis

In this section we derive an upper bound for the running time of the algorithm developed in the previous sections. As the heart of it is a MCF reduction and a multitude of algorithms exists for solving that problem, the running time of our algorithm depends on the underlying MCF algorithm.

Most descriptions of MCF solvers do not consider edge capacities. This is however not a restriction as one can get rid of the capacities using a linear graph transformation as Ahuja, Magnanti and Orlin describe in their book on network flows [3]. They further describe a cost scaling algorithm whose running time is in $O(n^2 m \log(nC))$ where C is the maximum absolute cost value. The feasibility test makes use of an algorithm that has a worst case running time of $O(n^2 m)$.

Let n and m be the node and edge counts of the input graph G and let n' and m' be the node and edge counts of the directed flow network D we reduce to. Let further W be the maximum

weight in the input graph and let C' be the maximum absolute cost value. In 4.2 we have shown that

$$\begin{aligned} n' &= n+2\ , \\ m' &= m+n+1\ \text{, and} \\ C' &= W\ . \end{aligned}$$

The reduction itself has a running time of $O(n+m)$. The second phase therefore has a running time of

$$O(n+m+n^2 m\log(nW)) = O(n^2 m\log(nW))\ .$$

The first phase determines δ using a binary search over $\{1,\ldots,\max_{v\in V}\ell(v)\}$. As $\max_{v\in V}\ell(v)\leq\sigma$ its running time is in

$$\begin{aligned} O((n+m+n^2m)\log\sigma) &= O(n^2 m\log(\sigma)) \\ &\subseteq O(n^2 m\log n)\ . \end{aligned}$$

The third phase has a running time in $O(mn)$. The overall running time therefore is

$$O((n^2 m\log n)+n^2 m\log(nW)+mn) = O(n^2 m\log(nW))\ .$$

In our experiments we used a simplex based algorithm to solve the MCF. No polynomial upper bound for its running time is known but the author of the implementation used claims that in most cases it is faster than his cost scaling algorithm [1]. The actual running time observed in our test series is far below this upper bound. See Section 7 for a detailed evaluation of this algorithm.

5 Approximation Algorithm for Homosexual Speed Dating

In this section we modify the algorithm described in the previous section to approximate a solution for the homosexual speed dating problem. We develop an algorithm that approximates the general speed dating problem described in Section 3.1 without making any assumptions on the problem instances. This especially means that the input graph no longer necessarily has a bipartite structure.

We showed that the speed dating problem is NP-hard in Section 3.3 so, unless $P=NP$, it is not possible to adapt our algorithm and retain both polynomial running time and optimality. We sacrifice the latter and obtain a polynomial-time approximation algorithm. In Section 5.1 we analyze its performance. The basic top level structure of our general algorithm is very similar to the bipartite one.

1. Determine the minimum fairness violation δ using a binary search. It is possible that the δ calculated here is too small by one unit.
2. Determine the maximum weight uncolored edge subset E' given a fixed δ. Denote with G' the induced graph.
3. Color the edge set determined in the second phase using at most $\triangle_{G'}+1$ colors.
4. If more than σ colors were used determine and discard the edges of the *lightest color* i.e. the color whose edges have the lowest total weight.

In the bipartite case we were able to color each graph using $\triangle_{G'}$ colors. This is however not always possible with general graphs as for example the complete graph with three nodes can not be edge-colored with only two colors. Figure 3 illustrates this. However he has also shown that each graph can be edge colored with at most $\triangle_{G'}+1$ colors in polynomial time. As a consequence we can no longer treat the first two phases independently from the third if we want an optimal solution. Suppose we treat them independently then it is possible that we select a too low fairness variation in the first phase resulting in an uncolored edge subset in Phase 2 that can no longer be colored using $\triangle_{G'}$ colors in the third phase.

As we only develop an approximation algorithm, we treat the first two phases independently from the third phase. This introduces a certain error in the solution with which we have to cope.

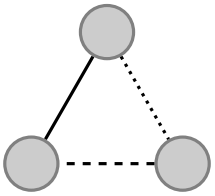


Figure 3: The complete graph with three nodes has a maximal degree of 2 but one needs 3 colors to color all edges. This subgraph can be produced by the first phase when $t = l(v) = u(v) = 2$ and the complete graph with three nodes is the input graph.

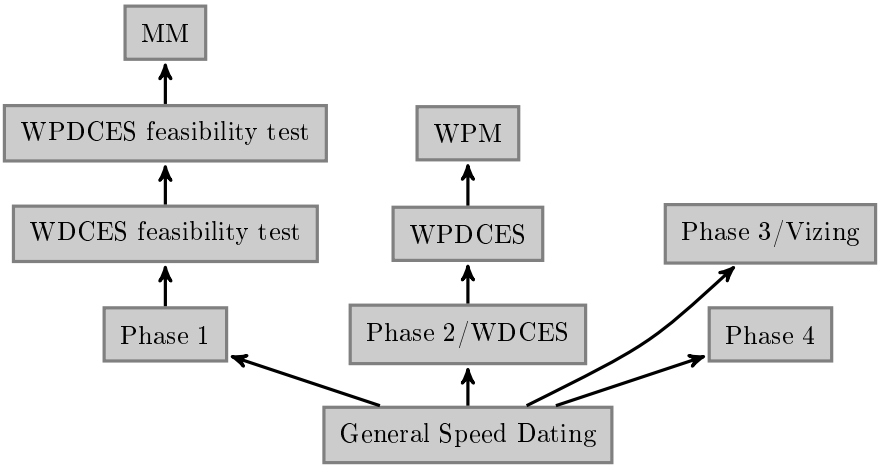


Figure 4: The reductions used in the approximation solver.

In Phase 3 we color the edges using $\triangle_{G'} + 1$ colors and make up for this with a fourth phase where we remove a color if necessary.

The first phase works in exactly the same way as in the bipartite case. See Section 4.1 for a description. The second phase uses the same top level idea but a completely different reduction. The third phase is Vizing's algorithm. The fourth phase is pretty straight forward. Figure 4 shows an overview of the different reductions used. Some of the problem names will only be defined later in this document.

5.1 Performance Guarantees

In this section we analyze the theoretical performance of the algorithm outlined in the previous section. Let I be an instance of the speed dating problem and δ_{OPT} and s_{OPT} the performance of an optimal solution with regard to the fairness and weight criteria, respectively. Our algorithm produces a solution with a performance of δ and s. We refer to the sum of the weights in the uncolored edge subset determined in the second phase by $\bar{s}$. It is important to note that at this point we have not yet removed the edges of the lightest color.

Removing those edges may decrease the degree of a node by at most 1 because each node has at most one incident edge of a certain color. If we remove an edge from a node that is at its lower bound then the fairness violation δ of the solution increases by 1. This leads to an absolute performance guarantee:

$$0 \leq \delta - \delta_{OPT} \leq 1$$

Determining a performance guarantee for the weight is a bit more complicated. In the worst case the algorithm needs $\triangle_{G'} + 1$ colors and $\triangle_{G'} = \sigma$. We therefore must dispose of a color so that only σ colors remain.

Given the optimal solution we can easily obtain an uncolored edge subset by just forgetting about the colors. The weight of the edge subset is not altered by this operation. This subset still satisfies all of the other requirements and therefore is a valid but maybe not optimal solution to

the second phase, i.e., to the WDCES problem. The optimal solution to this subproblem is $\bar{s}$. As this is a maximization problem this leads to $\bar{s} \geq s_{OPT}$.

We show using the pigeon hole principle that the lightest color has a weight of at most $\bar{s}/(\sigma+1)$. Suppose that the lightest color has a weight bigger than $\bar{s}/(\sigma+1)$ such as $\bar{s}/(\sigma+1)+\epsilon$ for some $\epsilon > 0$. The remaining σ colors must have a weight that is at least as big. Summing up these weights should result in $\bar{s}$. This however is not the case.

$$\begin{aligned}(\sigma+1)(\frac{\bar{s}}{(\sigma+1)}+\epsilon) &= \bar{s}+(\sigma+1)\epsilon \\ &> \bar{s}\end{aligned}$$

This is a contradiction implying that the lightest color has a weight of at most $\bar{s}/(\sigma+1)$. This gives us a lower bound for s, namely

$$s \geq \bar{s} - \frac{\bar{s}}{\sigma+1} = \frac{\sigma\bar{s}}{\sigma+1}\ .$$

Using this information we can determine the relative performance guarantee.

$$\frac{s_{OPT}}{s} \leq \frac{\bar{s}}{\frac{\sigma\bar{s}}{\sigma+1}} = \frac{\sigma+1}{\sigma}\ .$$

Hence, the solution produced by our algorithm always has a fairness violation δ that is at most off by one and a weight that is at least $\sigma/(\sigma+1)$ times the optimal value.

5.2 Phase 2: Determine the Maximum Weight Subgraph

In this section we describe two reductions to tackle the WDCES problem formulated in Section 4.1 for general graphs. This problem consists of finding a maximum weighted subgraph with node degree constraints. We first reduce it to the *weighted perfect degree constrained edge subset problem* (WPDCES), which can be formulated as follows.

Problem 12 (WPDCES)**.** Given a graph $G = (V, E)$ with edge weights $w : E \to \mathbb{N}\backslash\{0\}$ and a node capacity $c : V \to \mathbb{N}$ such that for each node v the inequality $0 \leq c(v) \leq d(v)$ holds, determine an edge subset of maximum weight such that each node v has degree $c(v)$ in the induced subgraph.

In a second step we reduce this problem to the *weighted perfect matching problem* (WPM), which can be stated as follows.

Problem 13 (WPM)**.** Given a graph $G = (V, E)$ with edge weights $w : E \to \mathbb{N}\backslash\{0\}$ find an edge subset of maximum weight such that each node has degree 1 in the induced subgraph.

These two reductions have already been described by Gabow [6]. We describe them here in the way we have implemented them. We make use of Lemon's [2] matching algorithm for solving the WPM. Gabow has proven that these reductions are correct and therefore we do not prove their correctness here. The goal of this section is to provide enough information to implement these algorithms. Gabow also describes a faster but harder to implement algorithm for solving the WPDCES problem. It uses a slightly different reduction, which can not reuse WPM solvers and therefore requires a lot more implementation work.

In the first reduction we construct a graph $G' = (V', E')$ that contains two disjoint copies of G. The edges have the same weight as their ancestors in the original graph. We set $c(v)$ to maximum number of dates $h(v)$ for each node. Let v be a designated a node in the first copy and let v' be the corresponding node in the second copy. In the next step we insert $h(v) - \ell(v)$ disjoint bridges between each v and v' where $\ell(v)$ is the minimum number of dates. A *bridge* consists of two additional nodes x and y and three edges $\{v, x\}$, $\{x, y\}$, $\{y, v'\}$ with weight 0. We set $c(x)$ and $c(y)$ to 1. Figure 5 shows an example for G and G'.

Once G' is constructed we solve the WPDCES problem with c as the node capacity on it. We get an edge subset E' such that each node in the induced graph has degree $c(v)$. Finally, we remove all bridges and one of the copies of G and obtain a solution to the original problem.

Each bridge can be included in two ways in the edge subset as Figure 6 shows because these are the two only ways to fulfill the node capacity requirement. As every edge has weight 0 this choice does not affect the weight condition. The one version binds one degree capacity of both v and v' to the bridge and the other does not. As both v and v' have capacity $h(v)$ and as there are

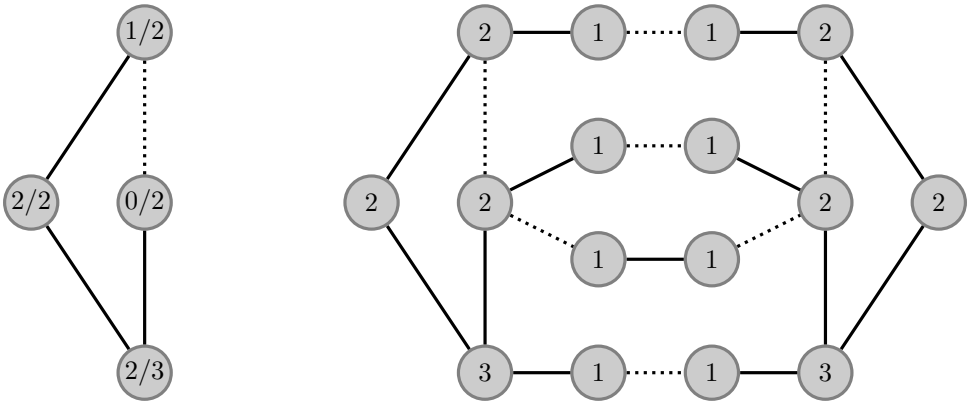


Figure 5: The left graph is the original graph. The labels in the nodes represent its capacity in the form $\ell(v)/h(v)$. The right side shows the transformed graph. The labels show the capacity of the nodes. The weights are omitted for clarity. The solid edges show a valid edge subset. On the right side one can see the two copies and the four bridges.

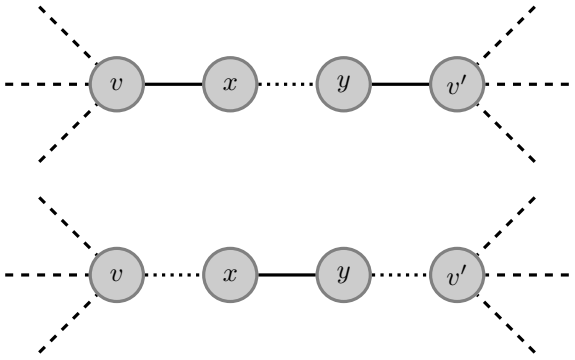


Figure 6: Shows the two ways that a bridge can be included in the subgraph. Solid edges are included in the edge subset and dotted ones are not. Dashed lines represent the rest of the graph.

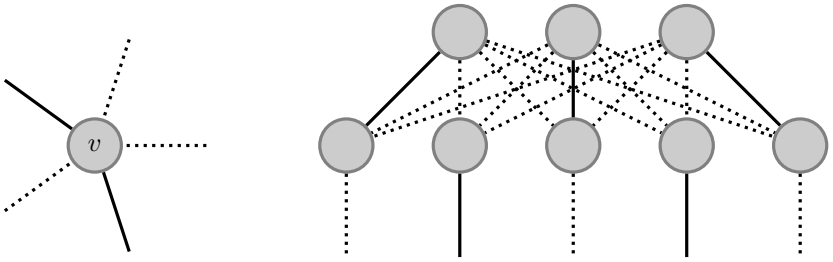


Figure 7: Shows a node and its replacement. The solid edges are included in the edge subset.

$h(v) - \ell(v)$ bridges this allows the capacities of v that are used on the non bridge part of the graph to vary between $\ell(v)$ and $h(v)$ as required.

Let β be the maximal number of bridges, i.e., $\beta = \max_{v \in V} h(v) - \ell(v)$ then $2n(1+\beta)$ is an upper bound for the number of nodes in G' and $2m + 3n\beta$ is an upper bound for the number of edges in G'. An upper bound for the maximum degree is $\triangle + \beta$.

The key idea of the second reduction is to replace each node by a bipartite subgraph. Let G be the graph constructed in the previous reduction. We construct G' by inserting a $K_{d(v),d(v)-c(v)}$ into G' for each node v. This is a complete bipartite graph with $d(v)$ nodes in the one set, which we refer to as *external* nodes, and $d(v) - c(v)$ nodes in the other set called the *internal* nodes. All the edges inserted so far have weight 0. Each external node will now be connected to exactly one external node of another subgraph. This means for each edge $\{u, v\}$ in G we choose a free external node in the subgraph of u and one in the subgraph of v and connect them. We set the weight of this new edge to the weight of the original edge. Next, we execute a weighted perfect matching algorithm on G'. We then contract the subgraphs of each node and obtain a solution to the original WDCES problem. Figure 7 shows a node and its replacement.

In the worst case $c(v)$ is 0 for each node so $2n\triangle$ is an upper bound for the number of nodes and $m + n\triangle^2$ is an upper bound for the number of edges. This transformation itself is fast but it significantly increases the size of the graph and is therefore responsible for most of the time needed by the approximation algorithm being developed. Gabow's other reduction mentioned in Section 5.2 lightens this problem as his sparse node substitute has a size linear in $d(v)$ and not quadratic as the substitute we use.

To perform the feasibility test in the first phase it suffices to check whether a degree constrained subgraphs exists. This can be tested using the exact same reductions but ignoring the weights. A perfect matching exists if and only if a *maximum matching* (MM) algorithm finds a matching with $n/2$ edges.

5.3 Phase 3: Coloring the Edges

The third phase consists of coloring the edges using $\triangle_{G'} + 1$ colors. We do this using Vizing's algorithm [9, 10]. We will not explain the details of the algorithm here. It works similarly to our bipartite coloring algorithm and also has a running time in $O(nm)$. As $\triangle_{G'} \leq \sigma$ we may have used one color more than allowed. If this is the case we correct it in the next phase.

5.4 Phase 4: Removing the Lightest Color

In this phase we dispose of a color if necessary. We first count the colors. At most $\sigma + 1$ colors may have been used. If exactly $\sigma + 1$ colors are used we identify the lightest color and remove all edges of that color.

5.5 Analysis

The WPM matching problem can be solved in $O(nm \log n)$ using the algorithm implemented in Lemon [2]. The MM problem can be solved in $O(m\sqrt{n})$ [8]. To determine the running time of the WPDCES problem let us first recall the size of the transformed graph. We have

$$\begin{aligned} n' &= 2n\triangle \in O(n\triangle) \text{ and} \\ m' &= m + n\triangle^2 \in O(n\triangle^2) \end{aligned}$$

where $\triangle$ is the maximum degree of the input graph. We made use of the fact that $m \leq n\triangle/2$ for every graph. The worst case running time is thus

$$O(n'm' \log n') = O(n^2\triangle^3 \log n\triangle) \subseteq O(n^5 \log n) \ .$$

Testing whether a feasible solution exists is faster because we can reduce it to the maximum matching problem, which can be solved faster, namely in time

$$O(m'\sqrt{n'}) = O(n\triangle^2\sqrt{n\triangle}) \subseteq O(n^4) \ .$$

In the next step, we determine the running time of the weighted degree constrained subgraph algorithm. The characteristical numbers of the transformed graph are

$$\begin{aligned} n' &= \in 2n(1+\beta) \in O(n\beta) \ , \\ m' &= 2m + 3n\beta \in O(m + n\beta) \text{ and} \\ \triangle' &= \triangle + \beta \in O(\triangle) \end{aligned}$$

where $\triangle'$ is the maximum node degree of the transformed graph and β is $\max_{v\in V} h(v) - \ell(v)$. The problem can thus be solved in time

$$n'^2\triangle'^3 \log n'\triangle' \in O(n^2\beta^2\triangle^3 \log n\beta\triangle) \subseteq O(n^2\triangle^5 \log n\triangle) \subseteq O(n^7 \log n)$$

and the corresponding feasibility test runs in time

$$n'\triangle'^2\sqrt{n'\triangle'} \in O(n\sigma\triangle^2\sqrt{n\beta\triangle}) \subseteq O(n\triangle^4\sqrt{n}) \subseteq O(n^{5,5}) \ .$$

Using the binary search over δ we can solve the first and the second phase in time

$$\begin{aligned} O((n^2\beta^2\triangle^3 \log n\beta\triangle) + n\beta\triangle^2\sqrt{n\beta\triangle}\log t) &= O(n^2\beta^2\triangle^3 \log n\beta\triangle) \\ &\subseteq O(n^5\sigma^2 \log n) \ . \end{aligned}$$

This clearly dominates the third phase, which has a running time of $\Theta(mn)$, so this is also the overall running time. The essence of this section can be summed up in the following theorem.

Theorem 14. *The general speed dating problem can be approximated with an absolute performance guarantee of 1 with respect to the weight criterion and with a relative guarantee of* $(\sigma+1)/\sigma$ *with respect to the fairness criterion in time* $O(n^5\sigma^2 \log n)$.

6 Heuristics

The worst-case running time derived in the previous section does not seem satisfactory. We therefore implemented and evaluated a couple of heuristic algorithms. All of these are greedy algorithms and have the same top level structure. They associate a *key* $k(e)$ to each edge e. At each iteration they insert an edge with maximum key that does not violate any constraints. The algorithms differ only in the choice of the key. Algorithm 3 shows this structure in more details. In the following subsections we describe the keys used by the different algorithms.

6.1 δ-absolute

Let $w_{\max}$ be the maximum edge weight and $d'(v)$ the node degree of v in the subgraph induced by the currently selected edge set E'. The key is calculated using the following formula:

$$k(u,v) = w_{\max} \max\{0, \ell(u) - d'(u), \ell(v) - d'(v)\} + w(u,v)$$

The first term tries to measure how well the δ-criterion would be satisfied if the edge e would be added to E'. The second does the same for the weight criterion. We multiply the first term with $w_{\max}$ to make sure that the first term always dominates the second one unless it is 0.

As adding an edge to E' changes $d'(v)$ the key values are not constant during the execution of the algorithm. All of the keys of the edges adjacent to an edge that is added may decrease by that action. We use a heap data structure to quickly determine an edge with maximum key value. The running time is in $O(m\triangle \log m)$.

Algorithm 3 The top level structure of the greedy algorithms considered.

```
H ← E;
E' ← ∅;
For all e ∈ E calculate k(e);
while H ≠ ∅ do
    e ← edge with maximum k(e) in H;
    H ← H\{e};
    if can add e to E' without violating the upper capacity constraint then
        if can color e without violating the coloring constraint in E' then
            E' ← E' ∪ {e};
            update all k(e) that changed;
        end
    end
end
```

6.2 δ-initial

The second algorithm uses a similar formula that does not depend on E' and therefore provides keys that are constant over time. This allows us to sort the edges once at the start of the algorithm. The formula is:

$$k(u,v) = w_{\max} \max\{\ell(u), \ell(v)\} + w(u,v)$$

The running time is in $O(m \log m)$.

6.3 δ-relative

Let c be the least common multiple of all $d(v)$. The third greedy algorithm uses the following key formula:

$$k(u,v) = w_{\max} c \max\{\frac{\ell(v)}{d(v)}, \frac{\ell(u)}{d(u)}\} + w(u,v)$$

The multiplication with c makes sure that the first term is integer. As it is also a multiple of $w_{\max}$ it dominates the second term unless it is 0.

The ratio $\frac{\ell(v)}{d(v)}$ describes how many of v's out edges must be included in E' to achieve a δ value of 0. If we randomly add edges then we will most likely have problems with nodes were this ratio is near 1. The idea of this greedy algorithm is therefore to add edges to these problematic nodes as early as possible.

The keys are constant over time and therefore it is possible to sort the edges once at the start of the algorithm. The running time is in $O(m \log m)$.

7 Evaluation

In the previous sections we have developed a number of algorithms and analyzed their theoretical performances. In this section we evaluate their running time using an implementation and test data.

Our first solver makes use of the ILP formulation we have given in Section 3.2. It simply feeds the input data to an off-the-shelf ILP solver. We have not tried to tune the ILP solver for this specific problem. In this section we refer to this solver of the speed dating problem as *ILP-based* solver. Next, we have implemented the algorithm described in Section 4 which solves the heterosexual speed dating problem. We refer to this solver as the *bipartite* solver as it operates only on bipartite graphs. The min cost flow solver used at the core of this implementation is the network simplex based solver bundled with Lemon [2]. This differs from the algorithm that was used in Section 4.4 to derive a bound for the worst case running time. The author claims that the simplex based approach is generally faster even though it has worse theoretical worst case running time [1]. We also implemented the approximation algorithm for general graphs described in Section 5. At its core it makes use of the solver for the general maximal matching problem provided by Lemon. We will refer to this solver as the *general* solver. Finally we implemented the three heuristic greedy algorithms described in Section 6. We refer to them as the *absolute* solver, the *initial* solver and the *relative* solver.

Unfortunately, we do not have any real world data. Therefore, we use random test instances. There is a quite large set of parameters that make up the instances, hence it is impossible to perform exhaustive testing. Parameters include the number of people, the number of bad dates, the fairness constraints, the date quality distribution and whether we consider homo- or heterosexual dating. We test instances that we suppose could be produced by the original problem statement.

We evaluate how fast our implementations are and how well the produced solution satisfies the fairness and weight criteria. We perform three test series. The first two evaluate the general performance of our algorithms. The last one tries to model a real world situation as closely as possible. As the performances depend on the graph structure we performed each test four times using different graphs of the same size to smoothen outliers.

Our tests have been performed on an Intel Core2 Quad Q6600 processor[1] with 2GB RAM. All tests are single threaded, i.e., make only use of a single core. The test were run under Ubuntu 11.1 and the compiler was GCC 4.4.1 with an optimization level of -O3.

7.1 Heterosexual Dating

In this experiment we evaluate the performance of our algorithms on bipartite graphs. Because of the way we implemented Vizing's algorithm the general solver always manages to color the graph in the third phase using σ colors. It therefore does not have to throw away the lightest color and thus always provides an optimal solution. As we have an optimal and fast solver at our disposal, it makes no sense to display the performance of the ILP-based or general solvers on any plot except the one showing the running times.

[1] See `http://ark.intel.com/Product.aspx?id=29765` for processor details

7.1.1 Varying the Number of People

In this section we vary the number of people n. For each two women there are three men. The number of bad dates is always set to 2/3 of all possible dates. There are $3n/5$ men and $2n/5$ women and therefore a total of $6n^2/25$ possible dates. As 2/3 are bad dates we have $m = \lfloor 2/25 \cdot n^2 \rfloor$. Which dates are bad is chosen at random. For each person we choose ℓ and h such that $\ell(v) \leq h(v) \leq 12$ randomly along a uniform distribution. A dating event with 12 dates and 10 min per date will last at least 2 hours. We suppose therefore that this produces realistic values. The date qualities are chosen randomly from $\{1, 2, \ldots, 100\}$ using a uniform distribution. We call this graph structure *uniform*.

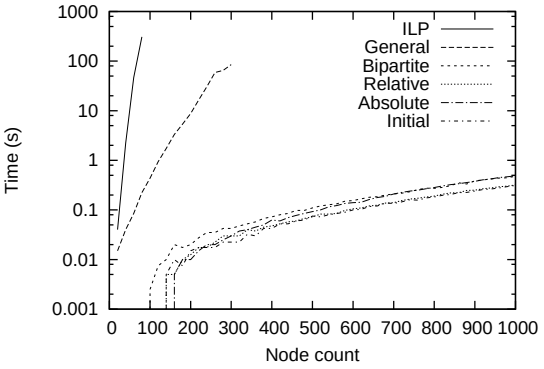
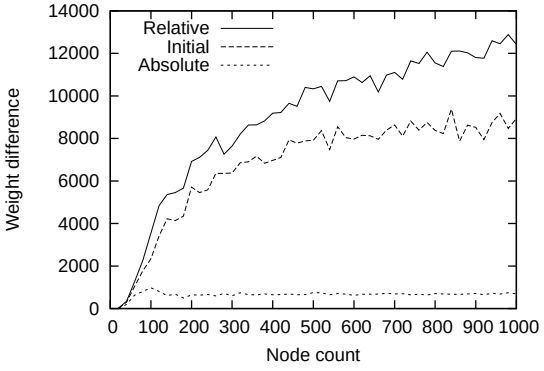


Figure 8: running times while varying n

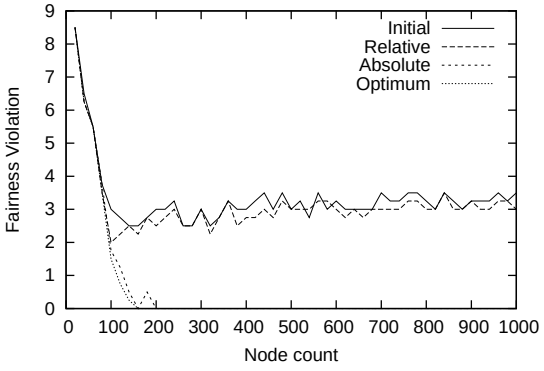
Plot 8 shows the running times of the evaluated algorithms. One can see that the ILP-based solver is not even able to handle small instances. The approximation algorithm for general graphs is not quite as slow as our analysis predicted but it is largely outperformed by the remaining algorithms. What is surprising is that the bipartite algorithm, which produces exact solutions, can rival the speed of the greedy algorithms. The algorithm is by far faster than what our worst case time analysis predicted. This can be explained to some extent by the fact that we used a different min cost flow solver in these tests than in the worst case analysis.

Plot 9 shows the performance of the approximation algorithms with respect to the weight criterion and the fairness criterion. The weight of the solutions is plotted as difference to the optimal value. The absolute Solver works well whereas the initial solver and the relative solver struggle a lot. They only find solutions that have a δ value of about 3 which is quite large considering that solutions with $\delta = 0$ exist. Also the weight criterion is far from being met optimally. We can explain these differences in performance by the fact that the key of the two bad heuristics are constant during the execution and the absolute solver may adjust the keys to react to unforeseen situations.

This experiment clearly shows that the bipartite solver is superior in every way to both the ILP as well as the general solver. We will therefore not consider them in the remaining experiments on bipartite graphs.



(a) weight difference to optimum



(b) fairness violation

Figure 9: performance for bipartite uniform graphs while varying n

7.1.2 Varying the Number of Dates

In the previous experiment the density of the graph was constant and we varied the size of the graph. To evaluate the effect of the density on the performances we perform another experiment, where we set $n = 1000$ and let m vary between 0 and $\lfloor 6/25 \cdot n^2 \rfloor = 240000$. All the other parameters are the same as in the previous experiment.

Figure 10 shows the running times of the four tested algorithms. The graph density is the same as in the previous experiment when $m = \lfloor 2/25 \cdot n^2 \rfloor = 80000$. The results observed at that density are consistent with those of the previous experiment. It is surprising that for large m the exact bipartite solver is fastest. We suppose that this effect is due to the fact that many edges have to be discarded by the greedy algorithms. In our implementation a loop over all colors is needed to discard an edge. Considering that there are only twelve colors this is not prohibitive but it may explain the running times observed.

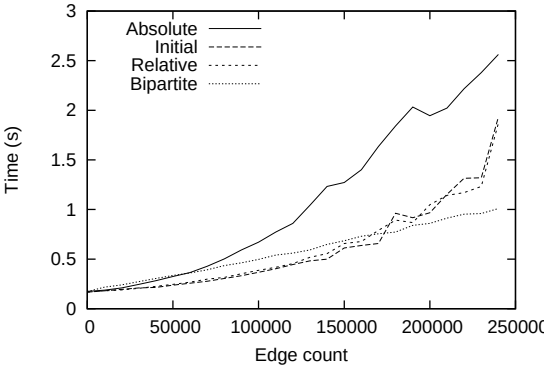
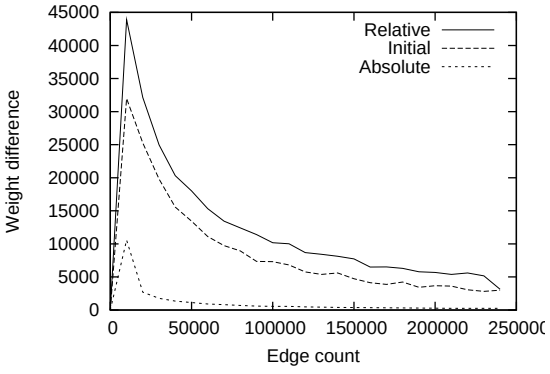


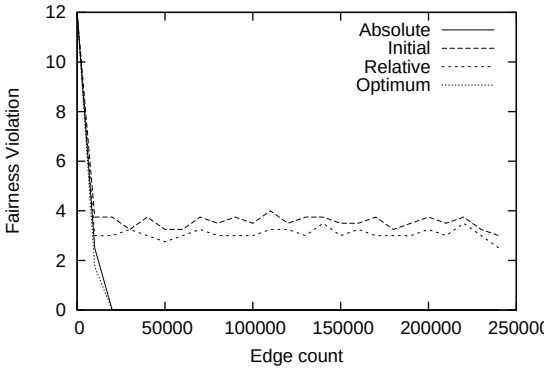
Figure 10: running times while varying m

Figure 11a shows the performance of the greedy algorithm with regard to the weight criterion. One can see, just as in the previous experiment, that absolute solver outperforms the two others. On sparse graphs the algorithms perform rather poorly. Probably there are fewer solutions that are close to the optimal one and therefore including a wrong edge in the subset has worse repercussions than when operating on dense graphs.

Figure 11b shows how well the fairness criterion is satisfied. We can see the same thing as in the previous experiment that the absolute solver is nearly optimal but the two others perform poorly. The peak near the sparse graphs is due to the graph structure as even the optimal fairness violation is not near 0.



(a) weight difference to optimum



(b) fairness violation

Figure 11: performance for bipartite uniform graphs while varying m

7.1.3 Real World Scenario

Our goal in this experiment is to evaluate how well our algorithms perform in models based on filled out forms. We suppose each form contains 30 questions and each of them requires the participant to rate something on a scale ranging from 0 to 5. Every person is therefore represented using a vector v_p with 30 elements chosen from the set $\{0/5, 1/5, \ldots, 5/5\}$ randomly along a uniform distribution. We use the euclidean distance to model the date goodnesses. The maximum distance of two such vectors is the distance of the vectors $(0, 0, 0, \ldots)$ and $(1, 1, 1, \ldots)$, which is $\sqrt{30}$. Using this model good dates have a small distance. We however need a large date goodness. To fix this we use the following formula to calculate the goodness of a date between the persons p and q:

$$w_{p,q} = \left\lfloor (\frac{\sqrt{30}}{2} - |v_p - v_q|_2) \cdot 100 \right\rfloor$$

A date is bad if $w_{p,q} \leq 0$. We still suppose that we have three men for every two women. We try to make sure that every person gets the maximum number of dates possible. Therefore, we set $\ell(v) = h(v) = 8$ for the men and $\ell(v) = h(v) = 12$ for the women. A graph with such a structure is called *form based*.

Figure 12 shows the running times of the different solvers tested. We also tried to evaluate the ILP-based solver but it was not capable of solving even the smallest instances in reasonable time. The running times are generally slower than in the previous experiments. Surprisingly, the running time of the bipartite solver is between the running times of the absolute solver and the other two greedy solvers.

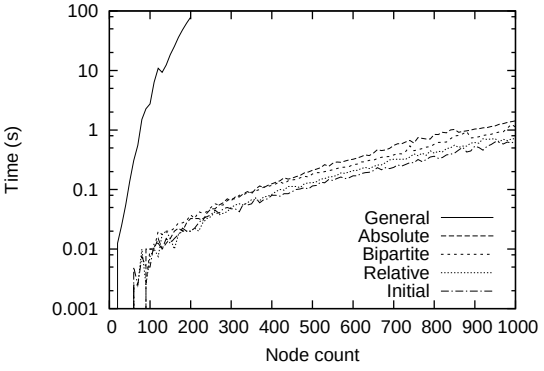
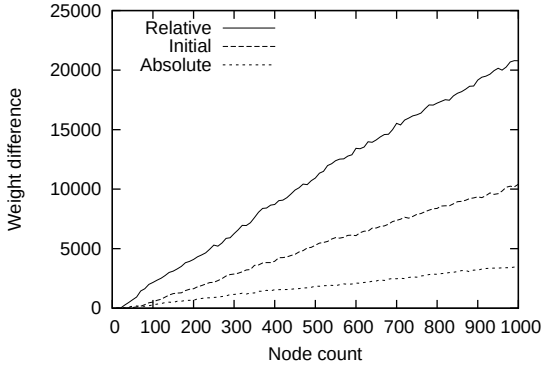


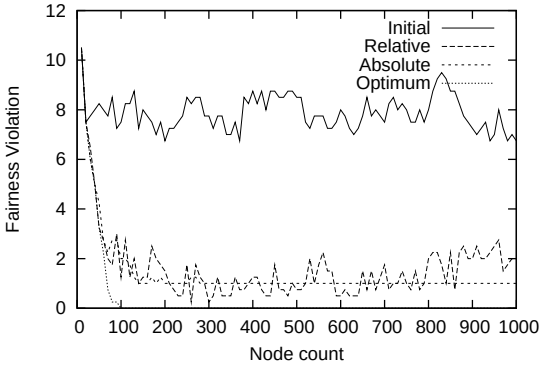
Figure 12: running times for bipartite form graph

Figure 13 show the quality of the solutions. They greatly differ from the results in Section 7.1.1. In terms of total weight the absolute solver still outperforms the two other greedy algorithms but the difference is smaller. The results for the fairness violations are very interesting. The initial solver performs just as bad as in the previous experiment. The relative solver is a lot better and the absolute solver no longer manages to achieve the optimum value but is nearly constantly off by only one.

We suppose that the reason the relative solver performs that well is because it first tries to match the sparse bridge like parts in the graph. Only once this is done it tries to match the edges inside the clusters. Each cluster is a rather dense subgraph and therefore it is simpler to find a matching with many edges inside it. It is however not as easy to find a heavy matching with many edges and therefore the relative solver performs bad in terms of total matching weight.



(a) weight difference to optimum



(b) fairness violation

Figure 13: performance for bipartite form based graphs

7.2 Homosexual Dating

In this section we present similar experiments to the ones in the previous section, however they examine general non-bipartite graph structures. As a consequence it is not possible to make use of the bipartite solver. We therefore no longer have a solver that can compute optimal solutions for large instances. We can therefore not plot the difference to the optimal solution as we did in the previous section but have to resort to plotting the actual values.

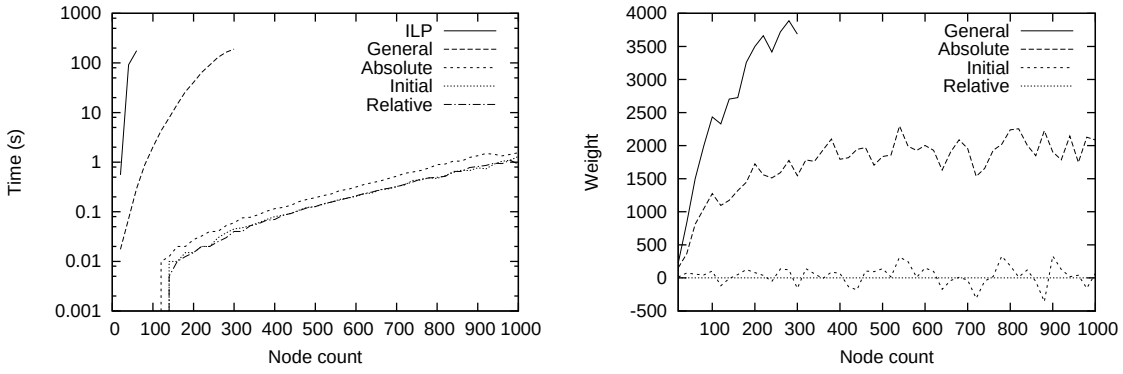
7.2.1 Varying the Number of People

This experiment is very similar to the one in Section 7.1.1. There are $n(n-1)/2$ possible dates and, to make sure that 2/3 of the dates are bad dates, we set $m = \lfloor n(n-1)/6 \rfloor$. The bad dates are again chosen uniformly at random. Figure 14a shows the running times of the different algorithms. They behave nearly the same as in the bipartite case.

We have also tested how well the weight and the fairness criteria are met. Figure 14b displays the result for the total weight. We plot the difference to the relative solver. The general solver produces the best results. The absolute solver is better than the other two greedy algorithms. These two produce solutions that are of the same quality. For the few test cases that the ILP based solver was able to find an optimal solution it found one that has the same quality as the solution found by the general solver. All solvers find nearly always solutions with an optimal fairness violation of zero. We only found a single test instance where the initial solver was off by 1. The plot showing this has been omitted as it would not provide any additional information.

An interesting observation is that these results are different from those in Experiment 7.1.1. Especially, the initial and relative algorithms do not struggle nearly as much as with bipartite graph structures when considering the fairness violation. It is possible that the good solutions for the bipartite graph structures make use of characteristic of that structure that these two solver can not find and therefore they are not able to determine the optimal solution.

Consider for a moment the weighted matching problem i.e. each node should be matched once. In graphs with a uniform weight distribution it is in general best to match the edges along an even cycle in an alternating way. This maximizes the expected total weight sum and provides a perfect matching. As soon as one edge is matched in a way that no such alternating chain can be constructed anymore one misses out on at least two edges. Greedy algorithms are likely to do this. As bipartite graphs only contain even cycles these algorithms often fall into this trap. General graphs on the other hand also contain cycles with an odd length. In those cycles no perfect matching is possible so this matters less. We know that this does not explain this effect very well, for example we do not know why the absolute solver can cope with bipartite graphs, however it is the best attempt at an explanation that we have got.



(a) running time

(b) weight difference to relative

Figure 14: results for uniform graph while varying n

7.2.2 Varying the Number of Dates

In this experiment we will fix $n = 100$ and vary m between 0 and $n(n-1)/2 = 4950$. The remaining parameters are the same as in the previous experiment. As the ILP based solver is incapable of producing solutions of this size we do not consider it.

Figure 15 shows the running time of the general algorithm. We also measured the time that the greedy algorithms take but it was nearly constantly below the resolution of our timer. It therefore makes no sense to plot these results. Figure 16a shows the achieved weights of the different algorithms. We plot the difference to the relative solver. It is interesting that on dense graphs the difference between the general solver and the greedy solvers diminishes as the graphs become denser. This effect is probably due to the fact that as there are more edges it is more likely that there are more edges with high weights and as a consequence the greedy algorithms do not have to use edges with a low weight at the end. The general solver can not make use of that many high weight edges as there is a maximum of edges that may be selected and the number of these good edges is higher than that limit. Figure 16b shows the values for the fairness violation. One should note that all algorithms are capable of finding optimal solutions most of the time when one only considers this criterion. Just as in the previous experiment we can observe that the initial and relative algorithms perform a lot better on non bipartite graphs.

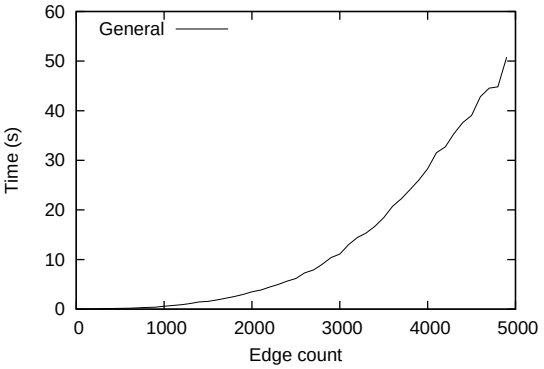
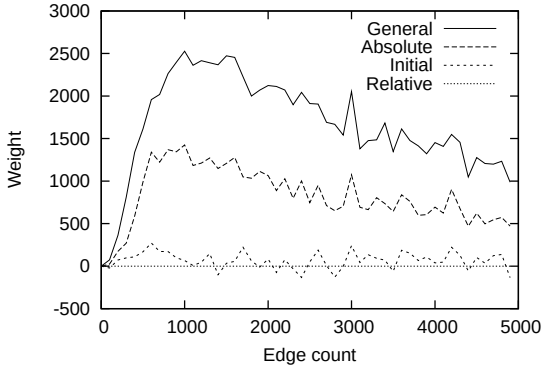
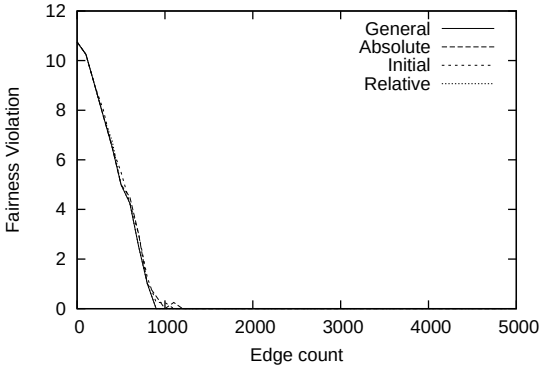


Figure 15: running times while varying m

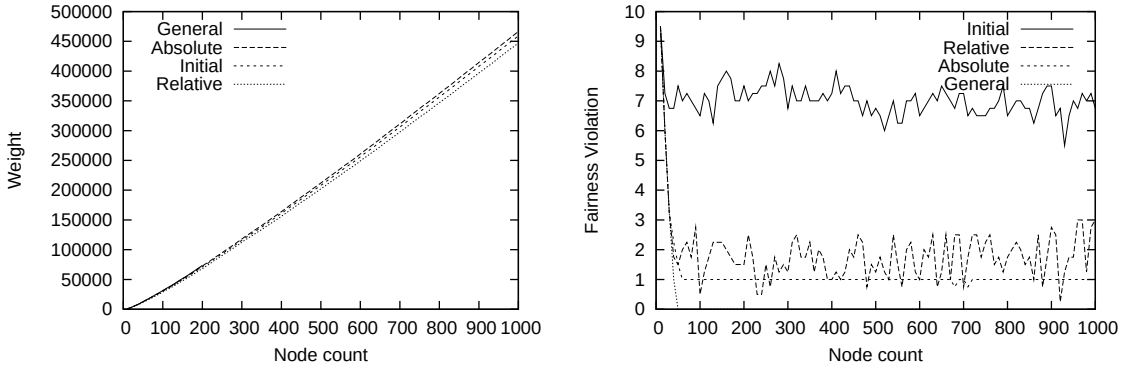


(a) weight difference to relative



(b) fairness violation

Figure 16: performance on uniform graphs while varying m



(a) total weight

(b) fairness violation

Figure 17: performance on form graphs

7.2.3 Real World Scenario

We also tested the solvers for general graph structures on form based graphs. Figure 18 shows the running times of the different algorithms. As we expected the general solver is slow and the rest is fast. Figure 17a shows the total weights of the solutions produced by the different solvers and Figure 17b the values for the fairness violation. It is surprising that the greedy algorithms struggle a lot more than with uniform graphs. The initial solver is always worst with a fairness violation significantly above the optimum. The relative solver is a lot better but its performance varies a lot. The absolute solver is constantly one unit above the optimum. The general solver nearly always finds solutions with an optimal fairness violation of zero.

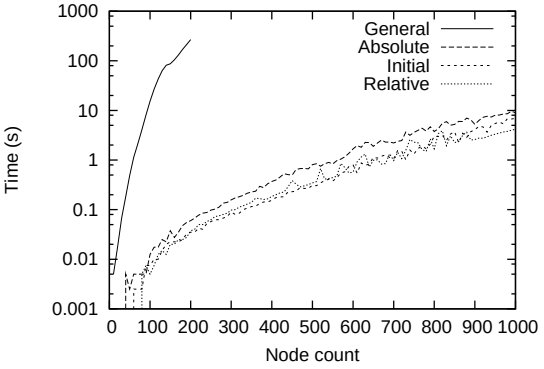


Figure 18: running times for form graphs

8 Conclusion

We have formalized the speed dating problem in graph-theoretic terms and as an integer linear program. We have shown that heterosexual special case can be solved in polynomial time and that the general homosexual problem statement is NP-hard.

We have developed an algorithm that solves the heterosexual special case in at most $O(n^2 m \log(nW))$ running time. Our experiments have shown that even for very large events with 1000 people the computation time is only slightly above a single second. Let us recall that σ is the number of dating rounds. We described an algorithm that computes an approximate solution for the general case in $O(n^5\sigma^2 \log n)$ running time. The weight of the solution found by our approximation algorithm is at least $\sigma/(\sigma+1)$ times the total weight of an optimal solution and the fairness violation is off by at most one unit. Empirical results show that our algorithm nearly always finds a solution with an optimal fairness violation and that the running time is acceptable with 5 minutes for a medium sized event of 200 people. The algorithm will perform better the more dating rounds the event has, as $\sigma/(\sigma+1)$ tends to 1 for $\sigma \to \infty$.

It turned out that the matching aspect can be solved in polynomial time but requires a lot of implementation work. We were only able to solve the scheduling subproblem efficiently for the heterosexual special case. The implementation work was however a lot easier. In the homosexual case the two subproblems can not be treated separately which complicates the problem of finding an optimal solution considerably. If one treats them separately then it is possible that scheduling problem becomes unsolvable.

We also investigated a number of different greedy approaches. It turned out that their performance heavily depends on the graph structure. Interestingly, different approaches perform vastly different on different graph structures. Our tests show that we have found one approach that provides solutions of decent quality even for large homosexual events. The running time is slightly below 10 seconds for 1000 people.

Starting Points for Follow-up Work

Our approximation algorithm does not make use of the best reductions known for solving the weighted degree constrained edge subset subproblem. It is possible to use the other reduction described by Harold [6]. This will certainly speed up the algorithm and lower the worst case running time but it will significantly increase the implementation work. Further our Phase 1 algorithm is optimized for a uniformly distributed fairness violation. However in most cases relevant to the original problem this violation is zero. Making use of this knowledge one can easily achieve a speed up. Instead of determining the minimal violation using a binary search, one can just guess that it is zero and one would almost always be lucky. It would certainly be interesting to investigate to what extent our algorithm can still be optimized.

Our tests have revealed that the performance of the initial and relative greedy algorithms presented here largely depends on the graph structures on which they are run. We have not investigated this effect. An interesting question is which are the exact graph features that trigger this behavior. One could try to derive performance guarantees for these graph classes. Further, it would be interesting to see what the worst case performance for the absolute greedy algorithm actually is.

We have concentrated on determining the actual date rounds and have deliberately ignored a number of aspects of the original problem. For example we have not investigated what the optimal order of the rounds is. One could gather additional requirements imposed by the real world situation and try to formalize these as algorithmic problem. Once this is done an obvious question is how to solve these efficiently.

Acknowledgment

I want to thank my advisors Bastian Katz and Ignaz Rutter for supporting me in a multitude of ways and for having always time and patience for questions and interesting discussions.

References

[1] LEMON minimum cost flow documentation. `http://lemon.cs.elte.hu/pub/doc/1.2/a00527.html`.

[2] Library for efficient modeling and optimization in networks (LEMON). `http://lemon.cs.elte.hu/`.

[3] Thomas L. ; Orlin James B. Ahuja, Ravindra K. ; Magnanti. *Network flows : theory, algorithms, and applications*. Prentice Hall, Upper Saddle River, NJ [u.a.], 1993.

[4] Richard Cole and John Hopcroft. On edge coloring bipartite graphs. *SIAM Journal on Computing*, 11(3):540–546, 1982.

[5] Reinhard Diestel. *Graph Theory (Graduate Texts in Mathematics)*. Springer, Berlin, Germany, August 2005.

[6] H. N. Gabow. An efficient reduction technique for degree-constrained subgraph and bidirected network flow problems. In *STOC '83: Proceedings of the fifteenth annual ACM symposium on Theory of computing*, pages 448–456, New York, NY, USA, 1983. ACM.

[7] Ian Holyer. The NP-completeness of edge-coloring. *SIAM Journal on Computing*, 10(4):718–720, 1981.

[8] S. Micali and V.V. Vazitani. An $O(V^{1/2}E)$ algorithm for finding maximum matching in general graphs. In *Proceedings of the 21st Annual IEEE Symposium on Foundations of Computer Science*, pages 17–27, 1980.

[9] J. Misra and David Gries. A constructive proof of vizing's theorem. *Inf. Process. Lett.*, 41(3):131–133, 1992.

[10] Michele Zito. `http://www.csc.liv.ac.uk/~michele/TEACHING/COMP309/2003/vizing.pdf`. 2003.