Stable flows with restricted edges

Ágnes Cseh*

May 11, 2014

Abstract

The stable marriage problem with its extensions is a widely studied subject. In this paper, we combine two topics related to it, setting up new and generalizing known results in both. The stable flow problem extends the well-known stable matching problem to network flows. Restricted edges have some special properties: forced edges must be in the stable solution, while forbidden edges may not be in it. Free edges are not able to block matchings.

Here we describe a polynomial algorithm that finds a stable flow with forced and forbidden edges or proves its nonexistence. In contrast to this, we also show that determining whether a stable flow with free edges exists, is NP-complete.

Keywords. stable matchings, stable flows, restricted pairs

1 Introduction

Stability is a well-known concept used for matching markets where the aim is to reach social welfare, instead of profit-maximizing. Such markets model resident allocation, university admission decisions or living donor kidney exchange programs.

1.1 Stable matchings

The stable marriage or stable matching problem was first described by Gale and Shapley in 1962 [8]. The instance $I = (G, O)$ contains a bipartite graph $G = (V, E)$, whose vertices are often called men and women. Each vertex has a strictly ordered, but possibly incomplete preference list of the vertices forming the other color class. If man $m$ prefers woman $w_1$ to woman $w_2$, then $w_1$ has a better ranking on $m$’s preference list than $w_2$: $r_m(w_1) < r_m(w_2)$. This is denoted by $w_1 <_m w_2$, and also referred as dominance: $mw_1$ dominates $mw_2$. The second element of the instance, $O$ is the set of preference lists.

A marriage scheme has to ensure that no one is paired to more than one person. Therefore, we are searching for a matching $M \subseteq E(G)$ in the graph-theoretical sense, that is, each vertex of $G$ has at most one incident edge in $M$. Intuitively, a set of marriages is stable, if no cheating may occur. If both the man and the woman of an unmarried couple prefer each other to their current partner (or to staying single, if they are), then they mutually agree to cheat with each other and break the marriage scheme. Such a pair is called a blocking edge. In other terms, $mw$ is an edge that dominates the matching at both end vertices. A matching is stable if no blocking edge occurs. With their well-known deferred acceptance algorithm, Gale and Shapley prove that a stable matching always exists.

*TU Berlin, Institut für Mathematik, Straße des 17. Juni 136, 10623 Berlin, Germany, e-mail: cseh@math.tu-berlin.de
1.2 Restricted pairs

The stable marriage problem has a broad variety of extensions. One of them deals with certain restrictions on edges. An edge \( mw \in E(G) \) of a stable marriage instance may play three essentially different roles regarding a matching \( M \):

1. \( mw \in M \) or
2. \( mw \notin M \) and \( mw \) is dominated by some edge in \( M \) or
3. \( mw \notin M \) and \( mw \) blocks \( M \).

An equivalent definition of stability is that \( M \) is stable iff no edge fulfills Property 3. Introducing restrictions on these properties leads to new concepts that are not only from the theoretical point of view interesting, but are also relevant concerning applications. In the literature, this topic is often referred as stability problems with special edges or restricted pairs.

The first paper [5] investigating special edges discussed the stable marriage problem with a set of forced and a set of forbidden edges. The instance of Dias et al. is given on a graph \( G \) with edge set partitioned in three disjoint subsets: the set of forced edges \( Q \), the set of forbidden edges \( P \) and the remaining edges \( E(G) \setminus (Q \cup P) \). A matching \( M \) is stable on an instance with forced and forbidden edges if it is stable in the classical sense and contains all forced edges and does not contain any forbidden edge. In other words, if \( wm \in Q \), then Property 1 has to be fulfilled for \( wm \), if \( wm \in P \), then Property 2 has to be fulfilled for \( wm \), and if \( wm \in E(G) \setminus (Q \cup P) \), then either Property 1 or Property 2 has to hold for \( wm \), otherwise \( M \) is unstable.

Finding a stable matching with forced and forbidden edges or proving that none exists can be done in \( O(n^2) \) time, where \( n = |V(G)| \). The problem can be reduced to the stable matching with forbidden (and no forced) edges problem. The latter has an algorithmic solution: its main idea is running the Gale-Shapley algorithm regardless on \( P \) and then breaking forbidden marriages.

Four years after these results Fleiner, Irving and Manlove published a comprehensive study [7] of the generalized stable marriage and roommates problems. The latter is the non-bipartite version of the stable marriage problem. Their instance allows partially ordered preference lists, forbidden edges and a non-bipartite graph. Amongst other results, they provide an \( O(n^2) \)-time algorithm to find a stable matching with forbidden (and no forced) edges problem. The latter has an algorithmic solution: its main idea is running the Gale-Shapley algorithm regardless on \( P \) and then breaking forbidden marriages.

Forced and forbidden edges are derived from the first two properties listed earlier in this section. A third type of special edges can be introduced with the help of the third, stability-defining property, hence by relaxing some stability conditions. Namely, if some of the blocking edges (fulfilling Property 3) can be ignored and the matching can regarded as a stable solution, the set of solutions may become larger. In this problem, a set of free edges is given and a matching is stable if it is not blocked by any of the non-free edges. Clearly, the presence of free edges allows a larger set of solutions, therefore only those cases are to combine with free edges that do not certainly have a stable solution in the classical case. One of them is the stable roommates problem. Fleiner and Cechlárová [2] show that the stable roommates problem becomes \( NP \)-complete when free edges appear in the graph.

Very recently, Askalidis et al. published several results on free edges [1]. They call the problem the socially stable matching problem, interpreting free edges as lack of social ties between agents of a many-to-one stable matching instance. After proving that the problem of finding a maximum socially stable matching on the classical stable marriage instance is \( NP \)-hard, they also give an approximation algorithm of factor 3/2.

Besides social networks, free edges (or social stability) can also model special market contacts, controlled by central authorities. Such deals can be made in order to reach stability, but
they also can be left unused. In the latter case, the pair of vendors does not block the matching, since they know that their partnership is controlled by some authority and they are not allowed to cooperate on their own.

1.3 Stable flows

As most matching problems, stable matchings also can be extended to network flows. A stable flow instance is a triple $I = (D, c, O)$. The vertex set of the underlying directed graph $D$ is divided into two disjoint subsets: to the set of terminals $S$ and to the set of non-terminals $V(D) \setminus S$. For the terminal vertices, the Kirchhoff law does not necessarily have to hold, these vertices in $S$ act like sources and sinks. All other vertices are obliged to forward the exact amount of flow they receive. On the edges of the network a non-negative capacity function $c : E(D) \rightarrow \mathbb{R}_{\geq 0}$. As usual, the capacity is the upper bound of the flow value on the given edge in a feasible flow. The network $(D, c)$ may contain parallel edges and loops. The last element $O$ of the triple stands for the preference ordering vertices have. Each non-terminal vendor ranks its incoming and its outgoing edges strictly. Similarly to the matching case, if $v$ prefers edge $vw$ to $vz$, then we write $vw <_v vz$. A feasible network flow $f$ is a function $f : E(D) \rightarrow \mathbb{R}_{>0}$ fulfilling the Kirchhoff law at each non-terminal vertex and the capacity condition for each edge.

Such a network with preferences models a market situation. The vertices are vendors dealing with some goods. The edges connecting them represent possible deals. Through his preference list, each vendor specifies how desirable a specific trade would be to him. Sources and sinks (producers and consumers) are not classified by having only incoming or only outgoing edges. Set $S$ contains all vertices that produce or consume some goods. A feasible network flow is stable, if there is no pair of vendors such that they mutually agree to modify the flow in the same manner.

**Definition 1.1 (stable flow).** A blocking walk of flow $f$ is a directed walk $W = (v_1, e_1, ..., e_{k-1}, v_k)$ such that all of the following properties hold:

1. each edge $e_i$, $i = 1, ..., k - 1$, is unsaturated;
2. $v_1 \in S$ or there is an edge $e' = v_1 u$ such that $f(e') > 0$ and $e_1 <_v e'$;
3. $v_k \in S$ or there is an edge $e'' = wv_k$ such that $f(e'') > 0$ and $e_{k-1} <_v e''$.

A network flow is stable, if there is no blocking walk in the graph.

**Theorem 1.2 (Fleiner, 2010 [6]).** There is a stable flow on every instance $(D, c, O)$ and it can be found in polynomial time.

The best currently known computation time is $O(m \log n)$, where $n = |V(D)|$ and $m = |E(D)|$. This bound is due to Fleiner’s reduction to the stable allocation problem and its fastest solution described by Dean and Munshi [4]. The most common solution method for stability problems, the Gale-Shapley algorithm can also be extended for stable flows [3]. At start, all terminal vertices push as much flow as possible on outgoing edges. Then, the vertices that have received offers, decide whether they are able to forward the flow on their outgoing edges. If they are, then they accept the offer and gain some overflow. Otherwise, they refuse the worst incoming edges, pushing back overflow to the vertex making the offer. In each step of the algorithm, vertices with positive overflow are called active vertices. Active vertices submit offers along their best edges, and receivers accept or refuse them as long as there is any overflow at non-terminal vertices.

Besides this, we emphasize a further theorem that will be used later in our proofs.

**Theorem 1.3 (Fleiner, 2010 [5]).** For a fixed instance, each edge incident to a terminal vertex has the same value in every stable flow.
Stable flows extend the stable marriage problem, moreover they can be seen as a generalization of many-to-one, many-to-many matching and allocation markets. Therefore, all results presented in this paper also hold for the above mentioned instances.

Structure of the paper
In the remaining of the paper, we discuss the stable flow problem combined with all three types of special edges. Forced and forbidden edges can be handled with the help of reductions to the classical stable flow problem. Both Section 2 and Section 3 are organized in the same manner: first, the case with a single special edge is studied, then the solution for the general case is described. Free edges appear in Section 4, we show that the maximum stable flow with free edges problem is \( \text{NP} \)-complete. The key result of this last part is a theorem claiming that determining the maximum cardinality of a stable matching with free edges is \( \text{NP} \)-complete. This result has been proved independently by Askalidis et al. [1].

2 Forced edges
In the flow problem, where edges have capacities, the notion of a forced edge requires to be revised. A new lower capacity function is introduced on instances with forced edges. This function shows how much goods have to be transported along the edge. A feasible flow is stable on a network with forced edges if it is stable in the classical sense and for each edge \( uv \in E(D) \) the inequality \( f(uv) \geq u(uv) \) holds. The stable flow with forced edges problem can be solved by a reduction to the classical stable flow problem. Before proving the correctness of this reduction, we discuss its technical details. In the first subsection, we show how to substitute a single forced edge with normal edges. Then, the method is extended to instances with several several forced edges.

2.1 A single forced edge
We are given a stable flow with forced edges instance on graph \( D \). Let the only forced edge be \( uv \). First, we substitute \( uv \) by two parallel edges, \( uv_1 \) with \( c(uv_1) = u(uv) \) and \( uv_2 \) with \( c(uv_2) = c(uv) - u(uv) \). The preference lists of \( u \) and \( v \) are changed at only one point: instead of \( uv \), \( uv_1 \) and \( uv_2 \) are listed, in this order. This change is made in order to show that handling forced edges with lower capacities is essentially the same procedure as handling completely forced edges. On the modified graph, a stable flow with \( f(uv_1) = c(uv_1) \) can be found if and only if there is a stable flow on the original network with \( f(uv) \geq u(uv) \).

From this point on, it is sufficient to consider a network with a completely forced edge \( uv \). As the next step, we modify graph \( D \) to get a helper graph \( D_{st} \). This modification consists of deleting the forced edge \( uv \) and introducing two new edges to substitute it. One of them starts at a new terminal vertex \( s \) and ends at \( v \), the other edge starts at \( u \) and ends at a new terminal \( t \). They both have capacity \( c(uv) \) and take over \( uv \)’s rank on \( u \)’s and on \( v \)’s preference lists.

\[ u \rightarrow r_u(uv) \rightarrow r_v(uv) \rightarrow v \]

\[ s \rightarrow u \rightarrow r_u(uv) \rightarrow r_v(uv) \rightarrow v \]

\[ u \rightarrow r_u(uv) \rightarrow r_v(uv) \rightarrow v \]

Lemma 2.1. There is a stable flow \( f \) on \( D \) with \( f(uv) = c(uv) \) if and only if there is a stable flow \( f_{st} \) on \( D_{st} \) with \( f_{st}(sv) = f_{st}(ut) = c(sv) = c(ut) \).

Proof. \( \Rightarrow \): The stable flow on \( D_{st} \) is constructed by copying \( f \) to the new graph and replacing \( f(uv) = c(uv) \) by \( f_{st}(sv) = f_{st}(ut) = f(uv) = c(sv) = c(ut) \). Comparing the dominance...
situation at vertices, it is straightforward that there is no blocking walk to \( f_{st} \) that did not block \( f \), because the unsaturated edges are exactly the same in both flows.
\[
\iff: \text{If } f_{st}(sv) = f_{st}(ut), \text{ then feasibility is kept while replacing } sv \text{ and } ut \text{ by } uv \text{ and setting } f(uv) \text{ to } f_{st}(sv). \text{ Just as before, the dominance situation remains unchanged.} \]

2.2 General case

Being able to handle a single forced edge we are ready to step forward and consider the case \( |Q| \geq 2 \). Substituting more than one forced edges in a manner described above can be done independently, without the forced edges impacting each other. Thus, introducing terminal vertices \( s \) and \( t \), deleting all edges in \( Q \) and substituting each one of them by two edges, one from \( s \) and one to \( t \) results in a classical stable flow instance on \( D_{st} \). For this instance, the following theorem holds:

**Theorem 2.2.** There is a stable flow \( f \) on \( D \) with \( f(uv) = c(uv) \) for all \( uv \in Q \) \iff there is a stable flow \( f_{st} \) on \( D_{st} \) with \( f_{st}(sv) = f_{st}(ut) = c(sv) = c(ut) \) for each \( uv \in Q \).

Due to Theorem \ref{t1} testing whether a stable flow with specific values on edges incident to terminals exists can be done in polynomial time. It is sufficient to find any stable flow, the flow value on \( sv \) and \( ut \) edges carries over to all stable flows. Thus, finding a stable flow with forced edges or proving its nonexistence can be done in \( O(m \log n) \) time.

Our results provide a fairly simple method for the stable matching with forced edges problem. After deleting each forced edge \( mw \in Q \) from the graph, we add \( mw_s \) and \( mw_t \) edges to each of the pairs. They take over the ranking of \( mw \). Then, if an arbitrary stable matching covers all of these new vertices \( w_s \) and \( w_t \), then there is a stable matching containing all forced edges. The running time of this algorithm is \( O(n^2) \), since it is sufficient to construct a single stable solution on an instance with at most \( 2n \) vertices. More vertices may not occur, because more than one forced edge incident to a vertex immediately implies infeasibility.

3 Forbidden edges

Just like forced edges, forbidden edges are also defined with the help of a lower capacity function \( u: E(D) \to \mathbb{R}_{\geq 0} \). A feasible flow is stable, if it is stable in the classical sense and \( f(uv) \leq u(uv) \) for each \( uv \in E(D) \). In order to handle the stable flow problem with forbidden edges, we present here an argumentation of the same structure as in the previous section. First, we prove that it is sufficient to consider the case of \( u(uv) = c(uv) \). Then, the problem of stable flows with single forbidden edge is solved. At last, an algorithm for the general case is described.

It is easy to see that lower capacities can be substituted by completely forbidden edges. Each edge \( uw \) with \( u(uw) > 0 \) can be split into two edges, one of them with \( c(uw_1) = u(uw_1) = u(uv) \) and the other one with \( c(uw_2) = c(uv) - u(uw) \) and \( u(uw_2) = 0 \). Both on \( u \)'s and \( v \)'s preference lists, \( uv_1 \) is placed right before \( uv_2 \), so that they preserve \( uv \)'s relation to all other edges. Stable flows on the original instance correspond to stable flows on the new instance, fulfilling the requirement that forbids \( uv_2 \) completely.

3.1 A single forbidden edge

Before we describe the method with which it can be decided if a network with a single forbidden edge admits a stable flow we show two constructions that we will use in the proof. In both cases, a graph \( D \) and a single completely forbidden edge \( uv \) is given. The first construction produces graph \( D_s \), by adding a terminal vertex \( s \) to \( V(D) \) and an edge \( sv \) to \( E(D) \). We set \( u(sv) = c(sv) \) to an arbitrary positive number. The ranking of \( sv \) on \( v \)'s preference list is better than the ranking of \( uv \), but it is worse than all edges being better than \( uv \) on \( D \). The second
construction is similar to the first one. Graph $D_t$ differs from $D$ in one terminal vertex $t$ and an edge $ut$. The capacity of $ut$ is positive and $ut$ is right before $uv$ on $u$’s preference list.

In the following, we characterize networks with forbidden edges with the help of $D_s$ and $D_t$. Our claim is that it can be decided whether there is a stable flow fulfilling the lower quota requirement on $D$ if and only if there is a stable flow for which $sv$ or $ut$ has 0 value. This second existence problem can be solved easily in polynomial time, since all stable flows have the same value on edges incident to terminal vertices.

**Lemma 3.1.** There is a stable flow $f$ on $D$ with $f(uv) = 0$ $\iff$ at least one of the following holds:

1. there is a stable flow $f_s$ on $D_s$ with $f_s(sv) = 0$ or
2. there is a stable flow $f_t$ on $D_t$ with $f_t(ut) = 0$.

**Proof.** $\Rightarrow$: Since $f$ is stable and $uv$ is unsaturated, every unsaturated walk passing through $uv$ dominates the flow at at most one end. Depending on which end it is, we will construct flow $f_s$ or $f_t$ on either of the modified graphs.

Suppose there is no unsaturated walk containing $uv$ that dominates $f$ at its end. In this case, $D_s$ is chosen: $f_s := f$ on all edges, except $f_s(sv) := 0$. Since $sv$ is the only edge that may dominate $f_s$ but does not dominate $f$, all walks that possibly block $f_s$ must contain $sv$. But since there is no unsaturated walk starting at $v$ that dominates $f$ (and $f_s$) at its end, $f_s$ is stable.

In the remaining case, where there is no unsaturated walk that dominates $f$ at its beginning, $f_t$ on $D_t$ is constructed. Similarly as above, $f_t := 0$ on $ut$ and $f_t := f$ on all other edges. If there is a blocking walk to $f_t$, it must pass through $ut$. But our assumption was that there is no dominating walk ending at $u$.

$\Leftarrow$: Suppose the first condition is fulfilled. Since $f_s(sv) = 0$ and $sv$ dominates $uv$, $f_s(uv)$ must be 0 for all stable flows. We construct $f$ from $f_s$ simply by omitting $f_s(sv)$. For this $f$, the equality $f(uv) = 0$ holds. Moreover, $f$ is stable, since no edge has less flow than in $f_s$ and no edge became dominant to $f$ that did not dominate $f_s$. An analogous argumentation holds for the case of the second condition. \qed

### 3.2 General case

The method described above decides in $O(m \log n)$ time whether a stable flow exists on a graph with a single forbidden edge. However, on networks with several forbidden edges, it is not straightforward how to find $sv$ or $ut$ edges to all edges with positive lower capacities. Applying our method greedily for each forbidden edge does not lead to correct results, since the steps can impact each other. An immediate consequence of Lemma 3.1 is that if there is a stable flow on the network with forbidden edges, a set of added $sv, ut$ edges also exists so that $f(sv) = f(ut) = 0$ for all of them. However, finding it with the same procedure may require exponentially many steps. In the following, we outline a polynomial algorithm that finds a stable flow on an instance with several forbidden edges, or terminates with a proof that no such flow exists.
The sketch of the proof is the following. During our algorithm, the forbidden edges are considered one by one. When adding a forbidden edge $uv$, the edge $sv$ is tested first, then edge $ut$, if needed. The first lemma shows that it is enough to check whether $f(sv)$ is positive when giving the first try to $D_s$. Having done this, we also know whether the stable flows of the new graph avoid all forbidden edges regarded so far. Our second lemma claims that if an $f(sv)$ proofs to be positive, then there is no stable flow avoiding all forbidden edges, if $D$ contains this $sv$ edge. In other words, the only opportunity to find a feasible solution is to check $ut$.

Let $P$ denote the set of forbidden edges, namely, $P = \{e \in E(D) : u(e) < c(e)\}$. It has been shown before that it is sufficient to consider only the case $u(e) = 0$. According to the properties mentioned in Lemma 3.1 and depending on the current graph we work with, $P$ is partitioned into three disjoint subsets. If $e \in P$ fulfills the first property in the lemma on the current instance, i.e. if $sv$ can be added to $E(D)$ so that $f(sv) = 0$, then $e \in P_1$. If only the second property holds for a forbidden edge, i.e. if only $ut$ can be added to $E(D)$ so that $f(ut) = 0$, then $e \in P_2$. Finally, if $e$ has not been investigated yet, then $e \in P_3$. Supposed that there is a stable flow with forbidden edges, there is a ‘good’ set $A$ of $sv$ and $ut$ edges, each allocated to one forbidden edge. It delivers us a graph on which $f(sv) = f(ut) = 0$ for all added edges. The set of good allocations is denoted by $A$. Our goal is to construct a set $A \in A$.

First we provide a pseudocode to our algorithm. The lemmas following it elaborate on its correctness. At the initial state of our method, all forbidden edges belong to the untested set $P_3$. In each round, our algorithm chooses an arbitrary forbidden edge $uv$ and constructs $D_s$ to it. If stable flows avoid $sv$, then $uv$ is added to $P_1$, and $sv$ is added to the edge set of the current graph and to the current $A$ set. Otherwise, $D_s$ is checked. If stable flows leave $ut$ empty, then $uv$ is replaced into $P_2$ from $P_3$. At the same time, $ut$ is added to the edge set and to $A$. If neither of the cases occurs, the algorithm terminates with a proof of infeasibility. During the execution of the algorithm, $D$ and $A$ is being constructed step-by-step, always adding one forbidden edge to the instance. To distinguish amongst the different phases, in step $i$, we work on $D_i$ and similarly, when considering some $P' \subseteq P$, the edge set added to the elements in $P'$ is referred as $A(P')$.

**Algorithm** Stable flow with forbidden edges

\[
P_1 := 0, P_2 := 0, P_3 := P, D_0 := D, i := 0
\]

\[\text{while } \exists uv \in P_3 \text{ do}
\]

\[\text{if } \exists \text{ a stable flow } f \text{ with forbidden edges } P_1 \cup P_2 \text{ on } D_i \cup sv \text{ with } f(sv) = 0 \text{ then}
\]

\[P_1 := P_1 \cup uv, P_3 := P_3 \setminus uv
\]

\[D_{i+1} := D_i \cup sv
\]

\[\text{else if } \exists \text{ a stable flow } f \text{ with forbidden edges } P_1 \cup P_2 \text{ on } D_i \cup ut \text{ with } f(ut) = 0 \text{ then}
\]

\[P_2 := P_2 \cup uv, P_3 := P_3 \setminus uv
\]

\[D_{i+1} := D_i \cup ut
\]

\[\text{else}
\]

\[\text{no solution}
\]

\[\text{end if}
\]

\[i := i + 1
\]

\[\text{end while}
\]

We start our proof of correctness with a rather straightforward observation we will refer to several times later.

**Lemma 3.2.** If $f(e) = 0$ for all $e \in A(P')$ in all stable flows on $D_0 \cup A(P')$ for some $P' \subseteq P$, then $f(e) = 0$ for all $e \in A(P'')$ in all stable flows on $D_0 \cup A(P'')$ for every $A(P'') \subseteq A(P')$.

**Proof.** This is simply due to the fact that deleting unused edges from the network may not create new blocking walks. To be more precise: since all edges in $A(P')$ have flow value 0 in
stable flows, these flows remain stable and avoid forbidden edges even if a subset of $A(P')$ is deleted from the graph.

**Lemma 3.3.** Consider a step of the algorithm after edge $uv \in P_3$ is chosen. We are given a stable flow on $D_{i-1}$ that has value 0 on all edges in $A(P_1 \cup P_2)$. Adding $sv$ to $E(D_{i-1})$ results in stable flows that either leave all edges in $A(P_1 \cup P_2) \cup sv$ empty or have positive flow value on $sv$.

*Proof.* On $D_{i-1}$ all stable flows have value 0 on edges in $A(P_1 \cup P_2)$, in particular, the Gale-Shapley flow as well. This flow is the output of the preflow-push algorithm on $D_{i-1}$. Adding an edge from $s$ to the network is equivalent to saturating this edge and to put its end vertex $v$ to the active set. Since the order of proposal steps of active vertices on $D_{i-1} \cup sv$ is arbitrary, the preflow-push version of the Gale-Shapley algorithm may call propose on edges leaving $v$ only if the active set has no other vertex than $v$. Note that a minor technical change may be needed, if the Gale-Shapley algorithm on $D_{i-1}$ already proposed along edges leaving $v$. In this case, splitting the edge $sv$ with a vertex $v'$ assures that the Gale-Shapley flow on $D_i$ can be derived from the Gale-Shapley flow on $D_{i-1}$ with a series of proposal and refusal steps starting at $v$ (or at $v'$).

At the beginning of these Gale-Shapley steps, $f(sv) = c(sv)$ and $v$ has exactly $c(sv)$ amount of overflow. It may forward it to other vertices, but the overall overflow on the network may not exceed $c(sv)$. If no flow pushed from $v$ could reach a different terminal vertex by the termination, then $sv$ is completely rejected and has 0 flow value. In this case, all edges that are incident to a terminal still have the same flow value as in the Gale-Shapley flow on $D_{i-1}$, in particular, all edges in $A(P_1 \cup P_2)$ have 0 flow value. Otherwise, if some of the pushed flow reached a terminal, then $sv$ is not entirely refused, it has a positive flow value, possibly with other edges in $A(P_1 \cup P_2)$.

This lemma becomes important when testing whether the next edge $uv \in P_3$ can receive an incoming edge in $A$. If it destroys the property needed to assure that all forbidden edges are empty, then $sv$ itself is not empty in stable flows. The next lemma completes the picture, showing that in this case, $sv$ may not be in any $A \in \mathcal{A}$.

**Lemma 3.4.** If $f(sv) > 0$ for a stable flow $f$ on $D_i \cup sv$, then there is no stable flow with forbidden edges on any graph consisting of $D_0 \cup sv$ and an sv or ut edge to each $e \in P$. That is: if there is no stable solution on $D_i \cup sv$, then $sv \notin A$ for every $A \in \mathcal{A}$.

*Proof.* We prove the lemma by induction. When $f(sv) > 0$ first occurs during the course of the algorithm, $A(P_1 \cup P_2)$ consists only of $sv$ edges. Now we delete some edges from the network $D_0 \cup A(P_1)$. The remaining graph $D_0 \cup sv$ differs from $D_0 \cup A(P_1)$ only in terminal-nonterminal edges not used by stable flows. Therefore, all stable flows have a positive value on $sv$, just like on $D_0 \cup A(P_1)$. Lemma 3.2 guarantees that if $sv \in A$, then $A \notin \mathcal{A}$, otherwise a subset of a good edge allocation would not be a good edge allocation.

Suppose that the statement is false, i.e. there is a set $A \in \mathcal{A}$ so that $sv \in A$ for the current $uv$. Again we use Lemma 3.2 and delete some edges from the network. In particular, if we consider $D_0 \cup A(P_2)$ and add $sv$ to $uv$, all stable flows have value 0 on all added edges. The inductive assumption ensures that $A(P_1) \cup sv$ consists only of $ut$ edges. Similarly to the previous proof, we investigate the Gale-Shapley flow on this network $D_0$.

The key observation of our argumentation is that during the execution of the Gale-Shapley algorithm, adding more edges from sources to the graph may not result in more flow on edges coming from sources not used so far. The difference between $D_i \cup sv$ and $D_0 \cup sv$ is the set of $sv$ edges added by our algorithm to $P_1$. They are in $D_i \cup sv$, but not in $D_0 \cup sv$. Running the preflow-push version of the Gale-Shapley algorithm on $D_i$ allows us to find the stable flow on $D_0$ and then call propose on the active end vertices of $P_1$. Due to Lemma 3.3 all that remains
to show is that these proposal steps may not turn $f(sv)$ positive. But since $s$ is not an active vertex and $v$ forwarded or rejected all flow it got from $s$, $f(sv)$ can never be increased.

With this induction step we proved an important property of edges $uv \in P_3$ that may not get an $sv$ edge in our algorithm. Since there is no $A \in \mathcal{A}$ allocating $sv$ to this edge, the only possibility to get a stable flow that avoids this edge is to allocate it an $ut$ edge. This is equivalent to the defining property of $P_2$, thus, $uv \in P_2$ or there is no feasible solution.

This lemma finishes the proof of the correctness of our algorithm for finding stable flows when forbidden edges are present. The running time is bounded by $O(|P|m \log n)$, since a stable flow has to be constructed for each forbidden edge. Both forced and forbidden edges on the same instance can be handled by our two algorithms, applying them after each other. Finally, we can conclude the following result:

**Theorem 3.5.** A stable flow with forced and forbidden edges or a proof for its nonexistence can be found in $O(m^2 \log n)$ time.

There is a closely related problem that can be solved in the same time, up to our current knowledge. As mentioned in Section 1.3, stable flow instances can be converted into stable allocation instances. The minimal cost stable allocation problem was solved by Dean and Munshi [4]. By using a well-studied concept of stability, rotations, they prove that an optimal solution can be found in $O(m^2 \log n)$ time. In our problem, if both forced and forbidden edges are present on a stable flow instance, the following strategy leads to a solution. First, the instance is converted into a stable allocation with forced and forbidden edges instance. Then, cost is assigned to each edge: $-u$ to forced edges, $u$ to forbidden edges and $0$ to the all remaining edges. If a stable allocation with forced and forbidden edges exists, it is also a minimum cost stable allocation. Therefore, this indirect method also answers the question whether a stable solution exist.

Just like in the previous section, we finish this segment with the direct interpretation of our results on classical matching instances. To each forbidden edge $mw \in P$ we introduce $mw_s$ or $mt_w$. According to the preference lists, they are slightly better than $mw$ itself. A stable matching with forbidden edges exits, if there is a good allocation of these $mw_s$ and $mt_w$ edges such that all $w_s$ and $mt$ are unmatched. Our algorithm for several forbidden edges runs in $O(mn^2)$ time, with this it is slower than the best known method.

### 4 Free edges

Recall that free edges were the ones that did not block the matching even if they dominated it at both end vertices. This property carries over to flow problems as well: the third type of special edges consists of edges that may have positive flow value but may not be a part of blocking walks. The presence of free edges makes every stability problem easier to solve. Since it is known that stable flows always exist, stable flows with free edges also always exist and can be found with the same methods that are used for the classical problem.

Free edges modify somewhat the properties of the set of solutions. For instance, Theorem 1.3, saying that all stable flows have the same value on edges connected to terminal vertices, does not hold any more, if free edges are present. An example with unit capacities can be seen below. If $uv$ is the only free edge on the graph, stable flows may have total flow value 1 or 2. The question rises naturally: what is the maximal flow value for stable flows with free edges?
As mentioned in Section 1, the maximum cardinality of a stable matching with free edges has not been known until recently. In this section we prove that the problem is \textit{NP}-complete, which immediately implies that finding a maximum flow on a network with free edges is also \textit{NP}-complete. Our proof is not exactly the same as the one in [1], but it is a very similar reduction of the same \textit{NP}-complete problem.

In the following, we provide a reduction of the maximum weakly stable matching problem to the maximum stable matching with free edges problem. The notion of weak stability is related to ties on preference lists. Especially if the concept of stability is applied to real-world instances, it is irrational to expect strictly ordered preference lists. For example, how should a university rank its applicants who reached the same score at the admission test? Yet allowing ties on preference lists brings up questions about the definition of a blocking edge. Three main variants of stability have been studied in the literature: weak, strong and super-stability. According to the first concept, an edge \( w_{m} / m \in M \) blocks the matching \( M \) if \( w \) and \( m \) both strictly prefer \( w_{m} \) to their partners in \( M \) or they are unmatched. Strongly stable matchings allow no edge out of the matching that strictly dominates the matching edge on one side and is not worse than it on the other side. Super-stability requires even more: an edge not in the matching already blocks it if both end vertices would come with that edge at least as good off as in the matching. Here we make use of the most common definition, the weak stability. Manlove et al. [9] show that finding a maximum cardinality weakly stable matching is \textit{NP}-complete, even if ties occur only on one side, have length 2 and stand on the bottom of preference lists.

**Theorem 4.1.** We are given a stable matching instance with free edges and \( K \in \mathbb{Z} \). Determining whether there is a stable matching with free edges that has cardinality at least \( K \) is \textit{NP}-complete.

**Proof.** The problem is obviously in \textit{NP}.

Each weakly stable matching instance \( \mathcal{I} \) satisfying the three properties mentioned above can be converted into an instance \( \mathcal{I'} \) of the maximum stable matching with free edges problem. The two instances differ only at edges that form ties. W.l.o.g. we can assume that ties occur only on the men’s side, let us denote the two edges in a tie by \( m_{w1} \) and \( m_{w2} \). In \( \mathcal{I'} \), \( m_{w1} \) remains unchanged, its ranking on \( m \)’s preference list is \( |N(m)| \), where \( N(m) \) is the set of vertices incident to \( m \). On the other hand, we substitute \( m_{w2} \) with two parallel paths: \( m, a, b, w_{2} \) and \( m, c, d, w_{2} \). The preferences are the following: \( r_{m}(ma) = |N(m)| + 1, r_{m}(mb) = |N(m)| - 1, r_{a}(ab) = 2, r_{b}(ab) = 1, r_{c}(cd) = 1, r_{d}(cd) = 2, r_{w2}(bw_{2}) = r(mw_{2} - 0.5), r_{w2}(dw_{2}) = r(mw_{2}) \). Edges \( mc, dw_{2} \) and \( ab \) are free. Such an instance conversion is illustrated below.
Lemma 4.2. For every $K \in \mathbb{N}$, there is a weakly stable matching $M$ with $|M| \geq K$ on $\mathcal{I}$ iff there is a stable matching $M'$ with free edges on $\mathcal{I}'$ such that $|M'| \geq K + 2$.

Proof. $\Rightarrow$: The construction of $M'$ is as follows. Apart from $mw_2$, we copy all edges of $M$ to $\mathcal{I}'$. For the rest of the graph, three scenarios are possible.

1. If $mw_2 \in M$, then $mc, dw_2, ab \in M'$.
2. If $m$ is unmatched (and $w_1$ and $w_2$ are both matched to better partners than $m$), let $ma, cd \in M'$.
3. If $m$ is matched to a vertex different from $w_2$, let $cd, ab \in M'$.

In all cases, $|M'| = |M| + 2$. Moreover, edges on $\mathcal{I}$ preserve their dominance relationship on $\mathcal{I}$. This is due to the fact that each edge in $M$ is represented in $M'$ by a similar edge. Similar means, in this context, that each edge dominated by the $M$-edge on $\mathcal{I}$ remains dominated by its representative in $M'$, and each edge dominating the $M$-edge also dominates its variant on $\mathcal{I}'$.

We shall now show that $M'$ is a stable matching on $\mathcal{I}'$.

Suppose $M'$ is blocked by an edge. Since $M$ was weakly stable on $\mathcal{I}$, the blocking edge must be incident to the vertices whose position has been changed: $m, w_1, w_2, a, b, c$ and $d$. Three of their edges, $ab, mc$ and $dw_2$, are free, therefore they may not block. If the blocking edge is incident to $m$, it has to be $ma$ or $mw_1$, because only these non-free edges on $m$’s preference list were affected by the transformation. Similarly, if it is incident to $w_2$, it must be $bw_2$. The last edge that may block $M'$ is $cd$. Each of these four edges fulfill at least either of the following properties in each of the two scenarios: (i) it is in $M'$ or (ii) one of its end vertices is matched in $M'$ to a better vertex. In order to avoid elaboration on technical details, we attached a table containing all cases. The notation ii) $v$ means that the edge is dominated by a matching edge at $v$. 

11
With this, we showed that no edge blocks $M'$.

$\Leftarrow$: If $M'$ is given, it is straightforward to construct a stable matching $M$ on $I$. Let $M$ and $M'$ be identical on all edges apart from $mw_2$. Note that $I'$ was constructed such a way that the subgraph spanned by the vertices $m, a, b, c, d$ and $w_2$ contains either three or two edges of any stable matching $M$. In the first case, if $m$ is matched to $a$ or $c$ in $M'$ and $w_2$ is matched to $b$ to $d$, then $mw_2 \in M$, otherwise $mw_2 \notin M$. These operations decrease the cardinality of the matching by exactly 2.

All that remains is to show that no edge blocks $M$. The preference lists were not changed essentially: any dominance between edges in the tie and out of it is still valid. Thus, the only edges that may block $M$ are $mw_1$ and $mw_2$. They are the worst edges of $m$, hence they only block $M$ if $m$ is unmatched and $w_1$ or $w_2$ has a worse partner than $m$ or they are unmatched. Suppose $mw_1$ blocks $M$. Since the same edge did not block $M'$ on $I'$, $mc \in M'$. It implies that $dw_2 \in M$ as well, otherwise $cd$ blocks $M'$. With this, we derived a contradiction, because we supposed that $mw_2 \notin M$. Let us consider the other case, when $mw_2$ blocks $M$. As mentioned above, $m$ is then unmatched in $M$. Because of edge $ma$, $m$ may not be unmatched in $M'$. Thus, if $mw_2 \notin M$, then $bw_2 \notin M'$ and $dw_2 \notin M'$. But then $cd$ blocks $M'$.

With this lemma, we justified the correctness of the reduction.

We remark here that an alternative version of this lemma admits a less complicated proof of the same structure. If two parallel edges are introduced instead of the paths $m, a, b, w_2$ and $m, c, d, w_2$, then most of the technical details can be spared. Though, the reduction holds then for stable matching instances with parallel edges, which is in some sense against the convention.

Using the usual matching-flow transition, it is easy to see that the result also holds for more complex instances, e.g. b-matchings, allocations and flows.

**Theorem 4.3.** Determining whether there is a stable flow with free edges that has flow value at least $K$ is $\mathbf{NP}$-complete.

Manlove et al. establish their complexity result also for minimum cardinality stable matchings. Thus, Theorem 4.1 and Theorem 4.3 both hold when the matching cardinality or the flow value is minimized.

## 5 Conclusion

All three types of special edges on stable flow instances have been discussed in our present work. While stable flows with forced and forbidden edges can be found in polynomial time, free edges make the maximal flow problem $\mathbf{NP}$-complete.

The most riveting open question is probably about approximation algorithms. Even if there is no stable flow containing all forced and avoiding all forbidden edges, how many edge conditions must be harmed by stable flows? Or the other way round: how can stability be relaxed such that all edge conditions are fulfilled? The first question, for example, can certainly be answered for matchings with the help of linear programming.

Stable matchings and stable flows with special edges can be combined with other common notions in stability: ties on preference lists, edge costs, couples, and so on.
References

[1] Georgios Askalidis, Nicole Immorlica, Augustine Kwanashie, David F. Manlove, and Emmanuel Pountourakis. Socially stable matchings in the hospitals/residents problem. In Frank Dehne, Roberto Solis-Oba, and Jörg-Rüdiger Sack, editors, *Algorithms and Data Structures*, volume 8037 of *Lecture Notes in Computer Science*, pages 85–96. Springer Berlin Heidelberg, 2013.

[2] Katarína Cechlárová and Tamás Fleiner. Stable roommates with free edges. Technical Report TR-2009-01, Egerváry Research Group, Budapest, 2009. www.cs.elte.hu/egres.

[3] Ágnes Cseh, Jannik Matuschke, and Martin Skutella. Stable flows over time. *Algorithms*, 6(3):532–545, 2013.

[4] Brian C. Dean and Siddharth Munshi. Faster algorithms for stable allocation problems. *Algorithmica*, 58(1):59–81, 2010.

[5] Vânia M.F. Dias, Guilherme D. da Fonseca, Celina M.H. de Figueiredo, and Jayme L. Szwarcfiter. The stable marriage problem with restricted pairs. *Theoretical Computer Science*, 306(1-3):391 – 405, 2003.

[6] Tamás Fleiner. On stable matchings and flows. In *Proceedings of the 36th International Workshop on Graph-Theoretic Concepts in Computer Science*, WG’10, pages 51–62, Berlin, Heidelberg, 2010. Springer-Verlag.

[7] Tamás Fleiner, Robert W. Irving, and David F. Manlove. Efficient algorithms for generalized stable marriage and roommates problems. *Theoretical Computer Science*, 381(1-3):162 – 176, 2007.

[8] David Gale and Lloyd Shapley. College admissions and the stability of marriage. *American Mathematical Monthly*, 1:9–14, 1962.

[9] David F. Manlove, Robert W. Irving, Kazuo Iwama, Shuichi Miyazaki, and Yasufumi Morita. Hard variants of stable marriage. *Theoretical Computer Science*, 276(1-2):261 – 279, 2002.