Algorithmica (2019) 81:2557–2591
https://doi.org/10.1007/s00453-018-00544-7

New and Simple Algorithms for Stable Flow Problems

Ágnes Cseh[1] · **Jannik Matuschke**[2]

Received: 19 July 2017 / Accepted: 31 December 2018 / Published online: 8 January 2019
© Springer Science+Business Media, LLC, part of Springer Nature 2019

Abstract
Stable flows generalize the well-known concept of stable matchings to markets in which transactions may involve several agents, forwarding flow from one to another. An instance of the problem consists of a capacitated directed network in which vertices express their preferences over their incident edges. A network flow is stable if there is no group of vertices that all could benefit from rerouting the flow along a walk. Fleiner (Algorithms 7:1–14, 2014) established that a stable flow always exists by reducing it to the stable allocation problem. We present an augmenting path algorithm for computing a stable flow, the first algorithm that achieves polynomial running time for this problem without using stable allocations as a black-box subroutine. We further consider the problem of finding a stable flow such that the flow value on every edge is within a given interval. For this problem, we present an elegant graph transformation and based on this, we devise a simple and fast algorithm, which also can be used to find a solution to the stable marriage problem with forced and forbidden edges. Finally, we study the stable multicommodity flow model introduced by Király and Pap (Algorithms 6:161–168, 2013). The original model is highly involved and allows for commodity-dependent preference lists at the vertices and commodity-specific edge capacities. We present several graph-based reductions that show equivalence to a significantly simpler model. We further show that it is NP-complete to decide whether an integral solution exists.

Keywords Stable flows · Restricted edges · Multicommodity flows · Polynomial algorithm · NP-completeness

A preliminary version of this paper appeared at the 43rd International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2017). The authors were supported by Cooperation of Excellences Grant (KEP-6/2018), by the Ministry of Human Resources under its New National Excellence Programme (UNKP-18-4-BME-331), the Hungarian Academy of Sciences under its Momentum Programme (LP2016-3/2016), its János Bolyai Research Fellowship, OTKA Grant K128611, COST Action IC1205 on Computational Social Choice, and by the Alexander von Humboldt Foundation with funds of the German Federal Ministry of Education and Research (BMBF).

✉ Ágnes Cseh
cseh.agnes@krtk.mta.hu

Extended author information available on the last page of the article

1 Introduction

Stability is a well-known concept used for matching markets without monetary transactions [33]. A stable solution provides certainty that no two agents are willing to selfishly modify the market situation. Stable matchings were first formally defined in the seminal paper of Gale and Shapley [19]. They described an instance of the college admission problem and introduced the terminology based on marriage that since then became wide-spread. Besides this initial application, variants of the stable matching problem are widely used in employer allocation markets [34], university admission decisions [2,4], campus housing assignments [5,32] and bandwidth allocation [18]. A recent honor proves the currentness and importance of results in the topic: in 2012, Lloyd S. Shapley and Alvin E. Roth were awarded the Sveriges Riksbank Prize in Economic Sciences in Memory of Alfred Nobel for their outstanding results on market design and matching theory.

In the classic stable marriage problem, we are given a bipartite graph, where the two classes of vertices represent men and women, respectively. Each vertex has a strictly ordered preference list over his or her possible partners. A matching is *stable* if it is not *blocked* by any edge, that is, no man-woman pair exists who are mutually inclined to abandon their partners and marry each other [19].

In practice, the stable matching problem is mostly used in one of its capacitated variants, which are the stable many-to-one matching, many-to-many matching and allocation problems. The *stable flow* problem can be seen as a high-level generalization of all these settings. As the most complex graph-theoretical generalization of the stable marriage model, it plays a crucial role in the theoretical understanding of the power and limitations of the stability concept. From a practical point of view, stable flows can be used to model markets in which interactions between agents can involve chains of participants, e.g., supply chain networks involving multiple independent companies.

In the stable flow problem, a directed network with preferences models a market situation. Vertices are vendors dealing with some goods, while edges connecting them represent possible deals. Through his preference list, each vendor specifies how desirable a trade would be to him. Sources and sinks model suppliers and end-consumers. A feasible network flow is stable, if there is no set of vendors who mutually agree to modify the flow in the same manner. A blocking walk represents a set of vendors and a set of possible deals so that all of these vendors would benefit from rerouting some flow along the blocking walk.

Literature review The notion of stability was extended to so-called "vertical networks" by Ostrovsky in 2008 [30]. Even though the author proves the existence of a stable solution and presents an extension of the Gale–Shapley algorithm, his model is restricted to unit-capacity acyclic graphs. Stable flows in the more general setting were defined by Fleiner [13], who reduced the stable flow problem to the stable allocation problem. Since then, the stable flow problem has been investigated in several papers [15,16,24,29]. Recently, stable flows have been used to derive conflict-free routings in multi-layer graphs [35].

The best currently known computation time for finding a stable flow is $\mathcal{O}(|E| \log |V|)$ in a network with vertex set V and edge set E. This bound is due to Fleiner's reduction

to the stable allocation problem and its fastest solution described by Dean and Munshi [8]. Since the reduction takes $\mathcal{O}(|V|)$ time, it does not change the instance size significantly, and the weighted stable allocation problem can be solved in $\mathcal{O}(|E|^2 \log |V|)$ time [8], the same holds for the maximum weight stable flow problem. The Gale–Shapley algorithm can also be extended for stable flows [7], but its straightforward implementation requires pseudo-polynomial running time, just like in the stable allocation problem.

It is sometimes desirable to compute stable solutions using certain forced edges or avoiding a set of forbidden edges. This setting has been an actively researched topic for decades [6,9,14,22,28]. This problem is known to be solvable in polynomial time in the one-to-one matching case, even in non-bipartite graphs [14]. Though Knuth presented a combinatorial method that finds a stable matching in a bipartite graph with a given set of forced edges or reports that none exists [28], all known methods for finding a stable matching with both forced and forbidden edges exploit a somewhat involved machinery, such as rotations [22], LP techniques [10,11,23] or reduction to other advanced problems in stability [9,14].

In many flow-based applications, various goods are exchanged. Such problems are usually modeled by multicommodity flows [25]. A maximum multicommodity flow can be computed in strongly polynomial time [36], but even when capacities are integer, all optimal solutions might be fractional, and finding a maximum integer multicommodity flow is NP-hard [21]. Király and Pap [27] introduced the concept of stable multicommodity flows, in which edges have preferences over which commodities they like to transport and the preference lists at the vertices may depend on the commodity. They show that a stable solution always exists, but it is PPAD-hard to find one.

Our contribution and structure In this paper we discuss new and simplified algorithms and complexity results for three differently complex variants of the stable flow problem. Section 2 contains preliminaries on stable flows.

- In Sect. 3 we present a *polynomial algorithm for stable flows*. To derive an efficient solution method operating directly on the flow network, we combine the well-known pseudo-polynomial Gale–Shapley algorithm and the proposal–refusal pointer machinery known from stable allocations into an augmenting path algorithm for computing a stable flow. Besides polynomial running time, the method has the advantage that it is easy to implement and that it provides new insights into the structure of the stable flow problem, which we exploit in later sections.
- Then, in Sect. 4 *stable flows with restricted intervals* are discussed. We provide a simple combinatorial algorithm to find a flow with flow value within a pregiven interval for each edge. Surprisingly, our algorithm directly translates into a very simple new algorithm for the problem of stable matchings with forced and forbidden edges in the classical stable marriage case. Unlike the previously known methods, our result relies solely on elementary graph transformations.
- Finally, in Sect. 5 we study *stable multicommodity flows*. First, we answer an open question posed in [27] by providing tools to simplify stable multicommodity flow instances to a great extent. In particular, we show that it is without loss of generality to assume that no commodity-specific preferences at the vertices and

no commodity-specific capacities on the edges exist. Then, we reduce 3- SAT to the integral stable multicommodity flow problem and show that it is NP-complete to decide whether an integral solution exists even if the network in the input has integral capacities only.

2 Preliminaries

A *network* (D, c) consists of a directed graph $D = (V, E)$ and a capacity function $c : E \to \mathbb{R}_{\geq 0}$ on its edges. The vertex set of D has two distinct elements, also called *terminal vertices*: a source s, which has outgoing edges only and a sink t, which has incoming edges only. Besides differentiating between the source and the sink, we will assume that D does not contain loops or parallel edges, and every vertex $v \in V \setminus \{s, t\}$ has both incoming and outgoing edges. These three assumptions are without loss of generality and only for notational convenience. We denote the set of edges leaving a vertex v by $\delta^+(v)$ and the set of edges running to v by $\delta^-(v)$.

Definition 1 (*flow*) Function $f : E \to \mathbb{R}_{\geq 0}$ is a *flow* if it fulfills both of the following requirements:

1. Capacity constraints: $f(uv) \leq c(uv)$ for every $uv \in E$;
2. Flow conservation: $\sum_{uv \in E} f(uv) = \sum_{vw \in E} f(vw)$ for all $v \in V \setminus \{s, t\}$.

A stable flow instance is a triple $\mathcal{I} = (D, c, r)$. It comprises a network (D, c) and r, a ranking function that induces for each vertex an ordering of their incident edges. Each non-terminal vertex ranks its incoming and also its outgoing edges strictly and separately. Formally, $r = (r_v)_{v \in V \setminus \{s,t\}}$, contains an injective function $r_v : \delta^+(v) \cup \delta^-(v) \to \mathbb{R}$ for each $v \in V \setminus \{s, t\}$. We say that v *prefers* edge e to e' if $\mathrm{r}_v(e) < \mathrm{r}_v(e')$. Terminals do not rank their edges, because their preferences are irrelevant with respect to the following definition.

Definition 2 (*blocking walk, stable flow*) A *blocking walk* of flow f is a directed walk $W = \langle v_1, v_2, \ldots, v_k \rangle$ such that all of the following properties hold:

1. $f(v_i v_{i+1}) < c(v_i v_{i+1})$, for each edge $v_i v_{i+1}$, $i = 1, \ldots, k-1$;
2. $v_1 = s$ or there is an edge $v_1 u$ such that $f(v_1 u) > 0$ and $\mathrm{r}_{v_1}(v_1 v_2) < \mathrm{r}_{v_1}(v_1 u)$;
3. $v_k = t$ or there is an edge $w v_k$ such that $f(w v_k) > 0$ and $\mathrm{r}_{v_k}(v_{k-1} v_k) < \mathrm{r}_{v_k}(w v_k)$.

A flow is *stable*, if there is no blocking walk with respect to it in the graph.

Intuitively, a blocking walk is an unsaturated walk in the graph so that both its starting vertex and its end vertex are inclined to reroute some flow along it. Notice that the preferences of the internal vertices of the walk do not matter in this definition.

Unsaturated walks fulfilling point 2 are said to *dominate* f at start, while walks fulfilling point 3 dominate f at the end. We can say that a walk blocks f if it dominates f at both ends.

Problem 1 SF Input: $\mathcal{I} = (D, c, r)$; a directed network (D, c) and r, the preference ordering of vertices.
Question: Is there a stable flow f?

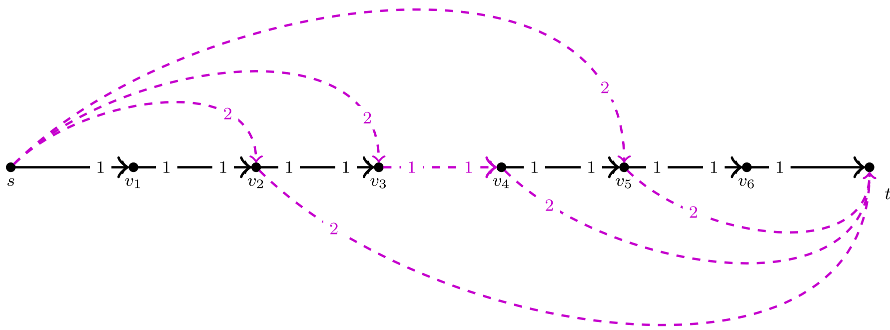

Fig. 1 The edge labels indicate the ranking of each edge at a vertex. For example, $v_3$prefers receiving flow from v_2 to receiving flow from s. The maximum flow (marked by dashed colored edges) has value 3 in this unit-capacity network, while the unique stable flow is of value 1 and is sent along the path $\langle s, v_1, v_2, \ldots, t\rangle$. It is easy to see that this instance can be extended to demonstrate the ratio $\Omega(|E|)$ (Color figure online)

Theorem 1 (Fleiner [13]) sf *always has a stable solution and it can be found in polynomial time. Moreover, for a fixed* sf *instance, each edge incident to s or t has the same value in every stable flow.*

This result is based on a reduction to the stable allocation problem. The second half of Theorem 1 can be seen as the flow generalization of the so-called *Rural Hospitals Theorem* known for stable matching instances [20]. While Theorem 1 implies that all stable flows have equal value, we remark that this value can be much smaller than that of a maximum flow in the network. In Example 1 we demonstrate a gap of $\Omega(|E|)$.

Example 1 (Small stable flow value) Flows with no unsaturated terminal-terminal paths are *maximal* flows. We know that every stable flow is maximal and it is folklore that the ratio of the size of maximal and maximum flows can be of $\mathcal{O}(|E|)$. As the instance in Fig. 1 demonstrates, this ratio can also be achieved by the size of a stable flow versus that of a maximum flow.

3 A Polynomial-Time Augmenting Path Algorithm for Stable Flows

Using Fleiner's construction [13], a stable flow can be found efficiently by computing a stable allocation in a transformed instance instead. Another approach is adapting the widely used Gale–Shapley algorithm to sf. As described in [7], this yields a preflow-push type algorithm, in which vertices forward or reject excessive flow according to their preference lists. While this algorithm has the advantage of operating directly on the network without transformation to stable allocation, its running time is only pseudo-polynomial.

In the following, we describe a polynomial time algorithm to produce a stable flow that operates directly on the network D. Our method is based on the well-known augmenting path algorithm of Ford and Fulkerson [17], also used by Baïou and Balinski [1] and Dean and Munshi [8] for stability problems. The main idea is to introduce proposal and refusal pointers to keep track of possible Gale–Shapley steps and execute

them in bulk. Each such iteration corresponds to augmenting flow along an s-t-path or a cycle in a restricted residual network.

3.1 Our Algorithm

In the algorithm, every vertex (except for the sink) is associated with two pointers, the *proposal pointer* and the *refusal pointer*. Throughout the course of the algorithm, the proposal pointer traverses the outgoing edges of the vertex in order of decreasing preference while the refusal pointer traverses its incoming edges in order of increasing preference. For the source s, we assume an arbitrary preference order. Starting with the 0-flow, the algorithm iteratively augments the flow along a path or cycle in the graph induced by the pointers. This graph consists of the edges pointed at by the proposal pointers and the reversals of the edges pointed at by the refusal pointer.

After each augmentation step, pointers pointing at saturated or refused edges are advanced. The algorithm terminates when the proposal pointer of the source has traversed all its outgoing edges. We prove that when this happens, the algorithm has found a stable flow. As in each iteration, at least one pointer is advanced, the running time of the algorithm is polynomial in the size of the graph. The complete algorithm is listed as Algorithm 1. In the following we describe the individual parts in detail.

Initializing and updating pointers For notational convenience, we introduce two artificial elements, $*$ at the top and $\emptyset$ at the bottom of each preference list with the convention $\mathrm{r}_v(*) = -\infty$ and $\mathrm{r}_v(\emptyset) = \infty$.

Every vertex $v \in V \setminus \{t\}$ is associated with a *proposal pointer* $\pi[v]$ and a *refusal pointer* $\rho[v]$, both pointing to elements on the preference list. Initially, $\pi[v]$ points to the most preferred outgoing edge on v's preference list, i.e., the entry right after $*$, whereas $\rho[v]$ is inactive, which is denoted by $\rho[v] = \emptyset$. We also set $\rho[t] = \emptyset$ for notational convenience (we will never change $\rho[t]$ during the algorithm). Note that this implies $\mathrm{r}_v(\rho[t]) = \infty$.

The pointers at v are advanced through the procedure AdvancePointers(v); see Algorithm 1, lines 11–17 for a formal listing. A call of this procedure works as follows:

- If $\pi[v]$ is active, it is advanced to point to the next less-preferred outgoing edge on v's preference list (lines 12–14). If all of v's outgoing edges have been traversed, $\pi[v]$ reaches its inactive state, i.e., $\pi[v] = \emptyset$, and $\rho[v]$ gets advanced from its inactive state to pointing to the least-preferred incoming edge on v's preference list. Note that in this latter case, the state of $\pi[v]$ changes from active to inactive between line 12 and line 15, and thus both if-conditions are fulfilled in the same call of the procedure.
- If $\pi[v]$ is already inactive, the refusal pointer $\rho[v]$ gets advanced to the next more-preferred incoming edge on the preference list (lines 15–17). Once $\rho[v]$ traversed v's most preferred incoming edge, we set $\rho[v] = *$, denoting all incoming edges of v have been refused (the procedure will not be called again for this vertex after this point).

Algorithm 1: Augmenting path algorithm for stable flows

```
   // Initialize proposal pointers to point at most-preferred
      outgoing edges, refusal pointers inactive.
1  Set π[v] := argmin_{vw∈E} r_v(vw) and ρ[v] := ∅ for all v ∈ V.
2  Set f := 0.
   // Ensure pointers only point to residual, non-refused edges.
3  while ∃uv ∈ E_{H_{π,ρ}} with c_f(uv) = 0 or (π[u] = uv and r_v(uv) ≥ r_v(ρ[v])) do
4  |  AdvancePointers(u)
   // Stop once proposal pointer of source becomes inactive.
5  if π[s] = ∅ then
6  |  return f
   // Augment flow along path/cycle induced by proposal and refusal
      pointers.
7  Let W be an s-t-path or cycle in H_{π,ρ}.
8  Set Δ := min_{e∈W} c_f(e).
9  Augment f by Δ along W.
   // Repeat.
10 Goto line 3.
11 procedure AdvancePointers(v)
   |  // If proposal pointer is active, advance it to next
   |     less-preferred outgoing edge.
12 |  if π[v] ≠ ∅ then
13 |  |  Set P := {vw ∈ E : r_v(vw) > r_v(π[v])} ∪ {∅}.
14 |  |  Set π[v] := argmin_{e∈P} r_v(e).
   |  // If proposal pointer has passed all edges, advance refusal
   |     pointer to next more-preferred incoming edge.
15 |  if π[v] = ∅ and ρ[v] ≠ ∗ then
16 |  |  Set R := {uv ∈ E : r_v(uv) < r_v(ρ[v])} ∪ {∗}.
17 |  |  Set ρ[v] := argmax_{e∈R} r_v(e).
```

The helper graph With any state of the pointers π, ρ, we associate a helper graph $H_{\pi,\rho}$. It has the same vertex set as D and the following edge set:

$$E_{H_{\pi,\rho}} := \Big\{\pi[v] \,:\, v \in V \setminus \{t\},\ \pi[v] \neq \emptyset\Big\} \\ \cup \Big\{ \mathrm{rev}(\rho[v]) \,:\, v \in V \setminus \{t\},\ \pi[v] = \emptyset,\ \rho[v] \neq *\Big\},$$

where $\mathrm{rev}(uv) := vu$ denotes the reversal of a given edge. Hence, for every vertex $v \in V \setminus \{t\}$, the graph $H_{\pi,\rho}$ either contains the edge $\pi[v]$, if the proposal pointer is still active, or it contains the reversal $\mathrm{rev}(\rho[v])$ of the edge $\rho[v]$, if the refusal pointer is active, or neither of these, if both pointers are inactive. Each edge $e \in E_{H_{\pi,\rho}}$ has a *residual capacity* $c_f(e)$ depending on the current flow f, defined by

$$c_f(e) := \begin{cases} c(e) - f(e) & \text{if } e \in E, \\ f(e) & \text{if } e = \mathrm{rev}(e') \text{ for some } e' \in E. \end{cases}$$

At the beginning of each iteration of the algorithm, we ensure that no proposal or refusal pointer points to an edge with residual capacity 0 and that no proposal pointer points to an edge that has already been refused by its head (lines 3–4).

Augmenting the flow The algorithm iteratively augments the flow f along an s-t-path or cycle W in $H_{\pi,\rho}$ by the bottleneck capacity $\min_{e \in W} c_f(e)$ (lines 7–9). Augmenting a flow f along a path or cycle W by Δ means that for every $e \in W$, we increase $f(e)$ by Δ if $e \in E$ and decrease $f(e')$ by Δ if $e = \text{rev}(e')$ for some $e' \in E$. Note that after the augmentation, $c_f(e) = 0$ for at least one edge $e \in W$, implying that at least one pointer is advanced before the next augmentation. Lemma 2 below shows that an augmenting path or cycle in $H_{\pi,\rho}$ exists as long as $\pi[s]$ is still active. The algorithm stops when $\pi[s] = \emptyset$ (lines 5–6).

3.2 Example Run of the Algorithm

Before we analyze the algorithm, we illustrate it by running it on the example instance given in Fig. 2. To each augmentation, the set of pointers is drawn in Fig 3.

Augmentation 1 Initially, the proposal pointers are set to $\pi[s] = sv$, $\pi[v] = vw$, $\pi[w] = [wt]$, while all refusal pointers are inactive (pointing to $\emptyset$). The graph $H_{\pi,\rho}$ consists of the edges sv, vw, and wt, which comprise a unique s-t-path W_1. The algorithm augments f along W_1 by its bottleneck capacity 1, yielding the flow $f(sv) = f(vw) = f(wt) = 1$ and $f(sw) = f(vt) = 0$.

Pointer update Because the residual capacity of wt is 0, AdvancePointers(w) is called. The procedure advances $\pi[w]$ to the inactive state $\emptyset$ and hence immediately

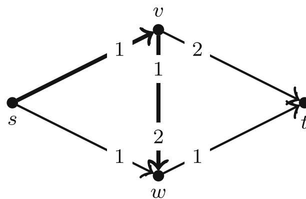

Fig. 2 Example instance for illustrating a run of Algorithm 1. Numbers next to the vertices indicate preferences of incident edges. Edge capacities are $c(sv) = c(vw) = 2$ and $c(sw) = c(vt) = c(wt) = 1$. For the algorithm, we choose the arbitrary preference order of the source s to prefer edge sv over sw

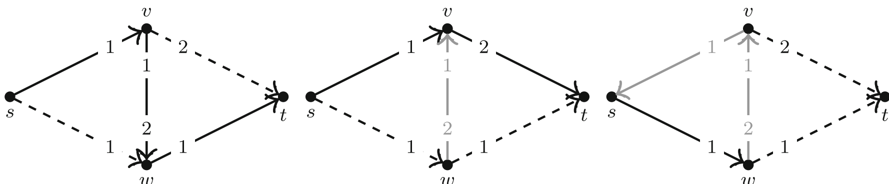

Fig. 3 The proposal and refusal pointers at the beginning of augmentations 1, 2, and 3, respectively. Proposal pointers are marked by solid black edges, while refusal pointers are the solid gray edges. The dashed edges do not belong to the current set of pointers

activates $\rho[w]$ with $\rho[w] = vw$. Because also $\pi[v] = vw$, this pointer is also advanced according to the second criterion of the while loop. It reaches $\pi[v] = vt$.

Augmentation 2 With $\pi[s] = sv$, $\rho[w] = vw$, and $\pi[v] = vt$, the graph $H_{\pi,\rho}$ consists of the edges sv, $\text{rev}(vw) = wv$, and vt. The unique s-t-path $W_2 = \langle s, v, t\rangle$ is chosen, the bottleneck capacity is $c_f(sv) = c_f(vt) = 1$. After augmenting f along W_2 by 1 unit, the new flow is $f(sv) = 2$, $f(vw) = f(vt) = f(wt) = 1$ and $f(sw) = 0$.

Pointer update Because $c_f(sv) = 0$, the pointer $\pi[s]$ is advanced to sw. Because $c_f(vt) = 0$, also $\pi[v]$ is advanced to $\emptyset$ and $\rho[v]$ gets activated with $\rho[v] = sv$.

Augmentation 3 With $\pi[s] = sw$, $\rho[w] = vw$, and $\rho[v] = sv$, the graph $H_{\pi,\rho}$ consists of the edges sw, wv, and vs. These edges comprise the cycle W_3. The residual capacities are $c_f(sw) = c_f(wv) = 1$ and $c_f(vs) = 2$. Augmenting f along W_3 by 1 unit yields the flow $f(sv) = f(sw) = f(vt) = f(wt) = 1$ and $f(vw) = 0$.

Pointer update Because $c_f(wv) = 0$, the pointer $\rho[w]$ is updated to sw, also triggering an update of $\pi[s]$ that was pointing at the same edge. After advancing $\pi[s]$ it reaches $\emptyset$ and hence the algorithm terminates.

3.3 Analysis

In the proof of correctness we utilize the following notation. We say the proposal pointer $\pi[v]$ has *reached* edge vw if $\mathrm{r}_v(\pi[v]) \geq \mathrm{r}_v(vw)$. We say $\pi[v]$ has *passed* the edge vw if $\mathrm{r}_v(\pi[v]) > \mathrm{r}_v(vw)$. We use analogous terms for the refusal pointer $\rho[v]$ with reversed inequality signs, respectively.

We now make a few observations on the behavior of the pointers. We first observe that $\pi[v]$ moves from most-preferred to least-preferred edge and $\rho[v]$ moves from least-preferred to most-preferred edge, the ranks of the two pointers are non-decreasing or non-increasing, respectively, during the course of the algorithm (note that the lowest rank in P is always higher than the current rank of $\pi[v]$ in line 13 and the highest rank in R is always lower than the current rank of $\rho[v]$ in line 16.

Observation 1 *Throughout the algorithm,* $\mathrm{r}_v(\pi[v])$ *never decreases and* $\mathrm{r}_v(\rho[v])$ *never increases for any* $v \in V \setminus \{t\}$.

Also, for each vertex, at most one of its two pointers is active at any time, as the refusal pointer is only advanced once the proposal pointer reaches the inactive state.

Observation 2 *Throughout the algorithm, for each* $v \in V \setminus \{t\}$ *either* $\rho[v] = \emptyset$ *or* $\pi[v] = \emptyset$.

Finally, we observe that proposal/refusal pointers do not skip any outgoing/incoming edge, respectively. This is due to the construction of P in line 13 and R in line 16, which contain every edge that has a rank strictly higher/lower, respectively, than the edge currently pointed at by the pointer.

Observation 3 *Let* $uv \in E$.

- *If* $\mathrm{r}_u(\pi[u]) < \mathrm{r}_u(uv)$ *before a call of* AdvancePointers*(u), then* $\mathrm{r}_u(\pi[u]) \leq \mathrm{r}_u(uv)$ *after that call.*

- *If* $\mathrm{r}_v(\rho[v]) > \mathrm{r}_v(uv)$ *before a call of* ADVANCEPOINTERS*(v), then* $\mathrm{r}_v(\rho[v]) \geq \mathrm{r}_v(uv)$ *after that call.*

We next establish a set of invariants that are useful for analyzing the algorithm.

Lemma 1 *The following invariants hold true for each $uv \in E$ any time the algorithm is in lines* 5–10*:*

1. *If* $\mathrm{r}_v(\rho[v]) \leq \mathrm{r}_v(uv)$ *then* $\pi[u] \neq uv$.
2. *If* $\mathrm{r}_v(\rho[v]) < \mathrm{r}_v(uv)$ *then* $f(uv) = 0$.
3. *If* $\mathrm{r}_u(\pi[u]) < \mathrm{r}_u(uv)$ *then* $f(uv) = 0$.
4. *If* $\mathrm{r}_u(\pi[u]) > \mathrm{r}_v(uv)$ *then* $f(uv) = c(uv)$ *or* $\mathrm{r}_v(\rho[v]) \leq \mathrm{r}_v(uv)$.

Note that due to the monotonicity of the pointers, once the premise of invariant 1, 2, or 4 is fulfilled for an edge, it will stay this way for the rest of the algorithm. Intuitively, the invariants state that 1 a proposal pointer does not point to a refused edge, 2 once a refusal pointer has passed an edge, the edge carries no flow, 3 an edge can only carry flow after it is reached by its proposal pointer, and 4 after a proposal pointer has passed an edge, the edge is fully saturated until the refusal pointer of its end reaches it.

Proof of Lemma 1 Invariant 1: Note that the pointers are only changed in the while loop in lines 3–4. If $\pi[u] = uv$, then $uv \in E_{H_{\pi,\rho}}$. Therefore the while loop does not terminate while $\pi[u] = uv$ and $\mathrm{r}_v(uv) \geq \mathrm{r}_v(\rho[v])$.

Invariant 2: Observe the invariant is true after intialization since $f(uv) = 0$. Note that $f(uv)$ can only increase in line 9 when $\pi[u] = uv$. In that case, Invariant 1 ensures that $\mathrm{r}_v(\rho[v]) > \mathrm{r}_v(uv)$. So the invariant can only become invalid by advancing the pointer $\rho[v]$ past uv. Consider the first time this happens in the algorithm. By Observation 3, this can only happen with a call of ADVANCEPOINTERS(v) when $\rho[v] = uv$. But then $\pi[v] = \emptyset$ by Observation 2 and therefore the call of ADVANCEPOINTERS(v) can only be triggered by the condition $c_f(vu) = 0$ of the while loop. But this implies $f(uv) = 0$, so the invariant did not become invalid.

Invariant 3: Initially, $f(uv) = 0$. The flow can only increase when uv is part of an augmenting path or cycle in line 9. This can only happen while $\pi[u] = uv$ by construction of $E_{H_{\pi,\rho}}$. Because $\mathrm{r}_u(\pi[u])$ is non-decreasing, $\mathrm{r}_u(\pi[u]) \geq \mathrm{r}_u(uv)$ is true at any time after the first increase of $f(uv)$.

Invariant 4: This invariant is true initially because $\rho[v] = \emptyset$. It can only lose its validity by advancing $\pi[u]$ or decreasing $f(uv)$. By Observation 3, $\pi[u]$ can only pass uv when ADVANCEPOINTERS(u) is called in line 4 while $\pi[u] = uv$. This call can be triggered because $\mathrm{r}_v(\rho[v]) \leq \mathrm{r}_v(uv)$ or because $c_f(uv) = 0$ (implying $f(uv) = c(uv)$). In either case, the invariant is not violated. The flow on $f(uv)$ can only decrease when $\mathrm{rev}(uv) \in W \subseteq E_{H_{\pi,\rho}}$. By definition, this can only happen if $\rho[v] = uv$, which is already enough to fulfill the invariant. □

With the following lemma, we show that, at the beginning of each iteration, the algorithm can actually find an s-t-path or cycle.

Lemma 2 *Each time the algorithm reaches line* 7*, the graph $H_{\pi,\rho}$ contains an s-t-path or a cycle.*

Proof Consider any $v \in V \setminus \{s, t\}$ at any time the algorithm reaches line 7. We show that if v has an incoming edge in $H_{\pi,\rho}$, then it also has an outgoing edge in $H_{\pi,\rho}$. Note that by definition of $E_{H_{\pi,\rho}}$, the only situation in which v has no outgoing edge is when $\rho[v] = *$.

Let $uv \in E_{H_{\pi,\rho}}$ be an incoming edge of v. This implies that either $uv \in E$ and $\pi[u] = uv$ or $vu \in E$ and $\rho[u] = vu$ by definition of $H_{\pi,\rho}$.

If $\pi[u] = uv$, Invariant 1 of Lemma 1 ensures that $\mathrm{r}_v(\rho[v]) > \mathrm{r}_v(uv)$ and hence $\rho[v] \neq *$. Therefore v has an outgoing edge in $H_{\pi,\rho}$.

If $vu \in E$ and $\rho[u] = vu$, the termination criterion of the while loop (lines 3-4) guarantees $f(vu) = c_f(\mathrm{rev}(uv)) > 0$. Hence, by flow conservation, v must also have an incoming edge $u'v \in E$ with $f(u'v) > 0$. By Invariant 2 of Lemma 1, this implies $\rho[v] \neq *$.

Thus every non-terminal vertex with an incoming edge also has an outgoing edge. Now observe that $\pi[s] \neq \emptyset$ ensures that s also has an outgoing edge in $H_{\pi,\rho}$. Thus, we can start a walk at s and extend it until we visit a vertex as second time, closing a cycle, or until we reach t having found an s-t-path. This concludes the proof of the lemma. □

Theorem 2 *Algorithm 1 computes a stable flow in polynomial time.*

Proof We first show that the algorithm indeed computes a stable flow. Assume by contradiction there is a walk $W = \langle v_1, v_2, \ldots, v_k \rangle$ blocking f. We use the previously established invariants to prove the following claim.

Claim *For every $i \in \{1, \ldots, k-1\}$, the pointer $\pi[v_i]$ has passed $v_i v_{i+1}$, i.e., $\mathrm{r}_{v_i}(\pi[v_i]) > \mathrm{r}_{v_i}(v_i v_{i+1})$.*

Proof We show the claim by induction on i. First consider the case $i = 1$. Due to point 2 in Definition 2, either $v_1 = s$ or $\mathrm{r}_{v_1}(v_1 v_2) < \mathrm{r}_{v_1}(v_1 w)$ for some $v_1 w \in E$ with $f(v_1 w) > 0$. In the former case, $\pi[s]$ has passed $v_1 v_2$ as the termination criterion of the algorithm implies $\pi[s] = \emptyset$. In the latter case, $f(v_1 w) > 0$ implies that $\pi[v_1]$ has at least reached $v_1 w$ by Invariant 3 of Lemma 1 and thus it has passed $v_1 v_2$.

Now consider any $i \in \{2, \ldots, k-1\}$. Note that by induction hypothesis $\pi[v_{i-1}]$ has passed $v_{i-1} v_i$. Furthermore $f(v_{i-1} v_i) < c(v_{i-1} v_i)$ because no edge of W is saturated. Hence, Invariant 4 of Lemma 1 implies that $\rho[v_i]$ must have reached $v_{i-1} v_i$. In particular, $\rho[v_i] \neq \emptyset$ and hence $\pi[v_i] = \emptyset$ by Observation 2, implying $\pi[v_i]$ has passed all edges. This completes the induction and proves the claim. □

Now consider v_k, the last vertex of W. Note that, due to the claim above, $\pi[v_{k-1}]$ has passed $v_{k-1} v_k$. Furthermore, $f(v_{k-1} v_k) < c(v_{k-1} v_k)$ as the blocking walk W is unsaturated. Hence, by Invariant 4 of Lemma 1, $\rho[v_k]$ has reached $v_{k-1} v_k$, i.e., $\mathrm{r}_{v_k}(\rho[v_k]) \leq \mathrm{r}_{v_k}(v_{k-1} v_k)$.

Observe that this implies $\mathrm{r}_{v_k}(\rho[v_k]) < \infty = \mathrm{r}_t(\rho[t])$ and therefore $v_k \neq t$ (remember that $\rho[t] = \emptyset$ never changes). Now consider any $uv_k \in E$ with $\mathrm{r}_{v_k}(v_{k-1} v_k) < \mathrm{r}_{v_k}(uv_k)$. Then $\mathrm{r}_{v_k}(\rho[v_k]) \leq \mathrm{r}_{v_k}(v_{k-1} v_k) < \mathrm{r}_{v_k}(uv_k)$ implies $f(uv_k) = 0$ by Invariant 2 of Lemma 1. Therefore W does not dominate f at the end, i.e., it does not fulfill point 3 of Definition 2. Thus W is not a blocking walk and the returned flow f is stable.

We now turn to the running time. Note that in every iteration of the while loop (lines 3–4), a pointer of a vertex is advanced. Thus the total number of iterations of the while loop throughout the whole algorithm is bounded by $2|E|$ by monotonicity of the pointers and the fact that each edge appears in at most two preference lists. Since every vertex has at most one incoming and one outgoing edge in $H_{\pi,\rho}$ by construction, finding edges violating the termination criterion of the loop can be done in time $\mathcal{O}(|V|)$. The same is true for finding an augmenting path or cycle in line 7. As after each augmentation, the residual capacity of at least one edge drops to 0, at least one pointer is advanced in line 4 between any two augmentations, limiting the number of augmentations by $2|E|$. Hence the total running time of the algorithm is bounded by $\mathcal{O}(|E||V|)$. We remark that a more sophisticated implementation using the dynamic-tree data structure can reduce this running time to $\mathcal{O}(|E| \log |V|)$. However, since our primary aim in this article is to provide new and simple approaches, we omit further investigation of this complication. □

4 Stable Flows with Restricted Intervals

Various stable matching problems have been tackled under the assumption that restricted edges are present in the graph [9,14]. A restricted edge can be *forced* or *forbidden*, and the aim is to find a stable matching that contains all forced edges, while it avoids all forbidden edges. Such edges correspond to transactions that are particularly desirable or undesirable from a social welfare perspective, but it is undesirable or impossible to push the participating agents directly to use or avoid the edges. We thus look for a stable solution in which the edge restrictions are met voluntarily.

A natural way to generalize the notion of a restricted edge to the stable flow setting is to require the flow value on any given edge to be within a certain interval. To this end, we introduce a *lower* and an *upper bound* function.

Problem 1 SF RESTRICTED Input: $\mathcal{I} = (D, c, r, \mathfrak{l}, \mathfrak{u})$; an SF instance (D, c, r), a lower bound function $\mathfrak{l} : E \to \mathbb{R}_{\geq 0}$ and an upper bound function $\mathfrak{u} : E \to \mathbb{R}_{\geq 0}$.
Question: Is there a stable flow f so that $\mathfrak{l}(uv) \leq f(uv) \leq \mathfrak{u}(uv)$ for all $uv \in E$?

Note that in the above definition, the upper bound $\mathfrak{u}$ does not affect blocking walks, i.e., a blocking walk can use edge uv, even if $f(uv) = \mathfrak{u}(uv) < c(uv)$ holds. In particular, it is not without loss of generality to assume $c(uv) = \mathfrak{u}(uv)$ for all edges uv, as decreasing $c(uv)$ may enlarge the set of stable flows.

In the following, we describe a polynomial algorithm that finds a stable flow with restricted intervals or proves its nonexistence. We start with an instance modification step in Sect. 4.1. Then we prove that restricted intervals can be handled by small network modifications that reduce the problem to the unrestricted version of SF. We show this separately for the case where only forced edges occur, which we call SF FORCED, in Sect. 4.2 and for the case where only forbidden edges occur, called SF FORBIDDEN, in Sect. 4.3. It is straightforward to see that these two results can be combined to solve the general version of SF RESTRICTED.

We mention that it is also possible to solve SF RESTRICTED by transforming the instance first into a weighted SF instance, and then into a weighted stable allocation

instance, both solvable in $\mathcal{O}(|E|^2 \log |V|)$ time [8]. The advantages of our method are that it can be applied directly to the SF RESTRICTED instance and it also gives us insights to solving the stable roommate problem with restricted edges directly, as pointed out at the end of Sects. 4.2 and 4.3. Moreover, our running time is only $\mathcal{O}(|P||E| \log |V|)$, where P is the set of edges with $\mathfrak{u}(uv) < c(uv)$.

4.1 Problem Simplification

SF RESTRICTED generalizes the natural notion of requiring flow to use an edge to its full capacity (by setting $\mathfrak{l}(uv) = c(uv)$) and of requiring flow not to use an edge at all (by setting $\mathfrak{u}(uv) = 0$), which corresponds to the traditional cases of forced and forbidden edges. In fact, it turns out that any given instance of SF RESTRICTED can be transformed into an equivalent instance in which $\mathfrak{l}(uv), \mathfrak{u}(uv) \in \{0, c(uv)\}$ for all $uv \in E$.

First observe that if $\mathfrak{l}(uv) > \mathfrak{u}(uv)$ for some $uv \in E$, then SF RESTRICTED trivially has no solution. Therefore, we henceforth assume $\mathfrak{l}(uv) \leq \mathfrak{u}(uv)$ for all $uv \in E$. We further execute the following technical change to the instance in order to obtain an equivalent instance with the desired properties. As shown in Fig. 4, we substitute each edge $uv \in E$ with three parallel paths (to avoid parallel edges): $\langle u, x, v\rangle$, $\langle u, y, v\rangle$ and $\langle u, z, v\rangle$. While uy and yv take over the rank of uv, ux and xv are ranked just above, uz and zv are ranked just below uy and yv. The capacities and bounds of the introduced edges are as follows.

$$\begin{aligned}
\mathfrak{l}(ux) &= \mathfrak{l}(xv) = \mathfrak{u}(ux) = \mathfrak{u}(xv) = c(ux) = c(xv) = \mathfrak{l}(uv) \\
\mathfrak{l}(uy) &= \mathfrak{l}(yv) = 0 \\
\mathfrak{u}(uy) &= \mathfrak{u}(yv) = c(uy) = c(yv) = \mathfrak{u}(uv) - \mathfrak{l}(uv) \\
\mathfrak{l}(uz) &= \mathfrak{l}(zv) = \mathfrak{u}(uz) = \mathfrak{u}(zv) = 0 \\
c(uz) &= c(zv) = c(uv) - \mathfrak{u}(uv)
\end{aligned}$$

In words, we split each edge uv with lower and upper bounds into three paths: the first path $\langle u, x, v\rangle$ requires an amount of flow exactly equal to its capacity $\mathfrak{l}(uv)$, the middle path $\langle u, y, v\rangle$ has capacity $\mathfrak{u}(uv) - \mathfrak{l}(uv)$ and is unrestricted, the last path $\langle u, z, v\rangle$ with capacity $c(uv) - \mathfrak{u}(uv)$ must not carry any flow.

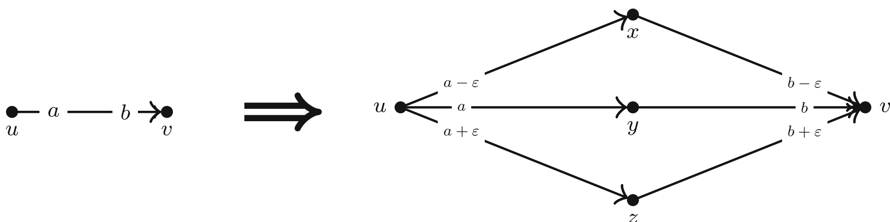

Fig. 4 Splitting an edge with lower and upper bounds. Due to the preferences, capacities and bounds defined on the modified instance, the first $\mathfrak{l}(uv)$ units of flow will saturate $\langle u, x, v\rangle$, then, the coming $\mathfrak{u}(uv) - \mathfrak{l}(uv)$ units of flow will saturate $\langle u, y, v\rangle$, and the remaining $c(uv) - \mathfrak{u}(uv)$ units of flow will use $\langle u, z, v\rangle$

Note that we can map any flow f in original graph to a flow f' in the modified graph by splitting the flow on each edge uv into three parts, setting $f'(ux) = f'(xv) = \min\{f(uv), \mathfrak{l}(uv)\}$, $f'(uy) = f'(yv) = \min\{\max\{f(uv) - \mathfrak{l}(uv), 0\}, \mathfrak{u}(uv)\}$, and $f'(uz) = f'(zv) = \max\{f(uv) - \mathfrak{u}(uv), 0\}$. Conversely, every flow f' in the modified instance induces a flow f in the original instance, simply by aggregating the flow values on the three paths, i.e., setting $f(uv) = f(ux) + f(uy) + f(uz)$.

Note that different flows in the modified instance can map to the same flow f in the original network, but it is easy to check that if f is stable, only a unique stable flow in the modified instance maps to f. Thus there is a one-to-one correspondence between stable flows in the original instance and in the modified instance. Furthermore, it is straightforward to check that f respects the bounds $\mathfrak{l}$ and $\mathfrak{u}$ in the original instance if and only if f' does the same in the modified instance. The modified instance is thus equivalent to the original instance.

Remark 1 Note that the encoding size of the modified instance is within a constant factor of the instance size of the original instance. More precisely, the number of edges in the new instance is $6|E|$ and the number of nodes in the new instance is $|V| + 3|E|$, where V and E are the sets of vertices and edges of the original instance, respectively. Also the set P of edges with $\mathfrak{u}(e) < c(e)$ only grows by a factor of 2. Note that because we assumed the original graph to be simple and connected, $|V| - 1 \leq |E| \leq |V|^2$ and therefore $\log(|V| + 3|E|) = \mathcal{O}(\log |V|)$. Therefore the asymptotic running time of $\mathcal{O}(|P||E| \log |V|)$ which we will establish for our algorithm on the modified instance is the same for the original instance.

Henceforth, we will assume that our instances are of this form and use the notation $Q := \{uv \in E : \mathfrak{l}(uv) = c(uv)\}$ and $P := \{uv \in E : \mathfrak{u}(uv) = 0\}$ for the sets of forced and forbidden edges, respectively.

4.2 Forced Edges

In this section we consider an instance of SF RESTRICTED where $P = \emptyset$. As mentioned earlier, we call this problem SF FORCED. In Sect. 4.2.1 we show how to deal with the case $|Q| = 1$ by reducing the corresponding SF FORCED instance with a single forced edge to an instance of SF without forced edges. Then, in Sect. 4.2.2, we argue that the same technique can be applied to multiple forced edges simultaneously. At last, in Sect. 4.2.3 we elaborate on the application of our technique for stable matching instances.

4.2.1 A Single Forced Edge

Let us first consider a single forced edge uv. We modify graph D to derive a graph D'. The modification consists of deleting the forced edge uv and introducing two new edges sv and ut to substitute it. Both new edges have capacity $c(uv)$ and take over uv's rank on u's and on v's preference lists, respectively, as shown in Fig. 5. The rest of D remains unchanged in D'.

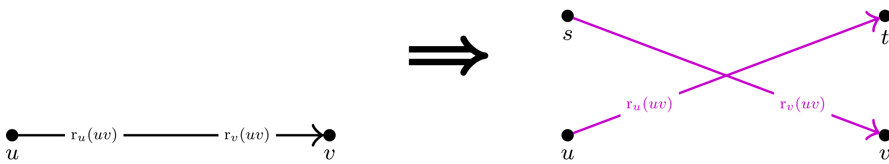

Fig. 5 Substituting forced edge uv by edges sv and ut in D'

In Lemma 3 we show that flows saturating uv in D are equivalent to flows saturating both sv and ut in D'. Then we refer to the extension of the Rural Hospitals Theorem (Theorem 1) to solve the latter problem.

Lemma 3 *Let f be a flow in D with $f(uv) = c(uv)$. Let f' be the flow in D' derived by setting $f'(sv) = f'(ut) = f(uv)$ and $f'(e) = f(e)$ for all $e \in E \setminus \{uv\}$. Then f is stable if and only if f' is stable.*

Proof We prove this lemma by showing that walks blocking f also block f' and vice versa. We first observe that the set of edges not saturated by f in D is the same as the set of edges not saturated by f' in D'. This is because uv is saturated by f, and therefore ut, sv are saturated by f', and all other edges are present in both graphs with identical capacities and flow values, respectively. Note that this implies the set of walks in D not saturated by f and the set of walks in D' not saturated by f' is the same.

Now consider any node $u' \in V$ and any number $r > 0$. Observe that there is an edge $u'v'$ in D with $\mathrm{r}_{u'}(u'v') = r$ and $f(u'v') > 0$ if and only if there is $u'v''$ in D' with $\mathrm{r}_{u'}(u'v'') = r$ and $f'(u'v'') > 0$ (either $u'v'$ itself is in D' or $u'v' = uv$, in which case $u'v'' = ut$ fulfills the requirement). Therefore an unsaturated walk W in D dominates f at the start if and only if it dominates f' at the start. A symmetric argument holds for dominance at the end of an unsaturated walk. This implies that any blocking walk for f in D is a blocking walk for f' in D' and vice versa. □

Checking the existence of a flow in D' that saturates both sv and ut can be done by finding any stable flow in D'. This is because Theorem 1 guarantees that all stable flows have the same value on any edge incident to s or t.

4.2.2 Multiple Forced Edges

We observe that we can replace all edges in Q one after the other, applying Lemma 3 inductively on the resulting graph. This yields the following theorem.

Theorem 3 *Let D_Q be the graph obtained from D when replacing each edge in $uv \in Q$ by edges ut and sv with same rank and capacity. Let $\bar{Q}$ be the set of newly added edges in D_Q. Let f be a flow in D saturating all edges in Q. Then f is stable if and only if the corresponding flow f' in D_Q obtained by setting $f'(sv) = f'(ut) = f(uv)$ for all $uv \in Q$ and $f(e) = f'(e)$ for all $e \in E \setminus Q$ is stable.*

In fact, the Rural Hospitals Theorem (Theorem 1) guarantees that either all stable flows in D_Q saturate all edges in $\bar{Q}$ or none does. Thus we can solve SF FORCED by a single stable flow computation in D_Q.

Theorem 4 SF FORCED *can be solved in time* $\mathcal{O}(|E|\log|V|)$.

Proof As D_Q contains at most twice as many edges as D, we can compute a stable flow f' in D_Q in time $\mathcal{O}(|E|\log|V|)$, as discussed at the end of Sect. 3. If $f'(sv) = f'(ut) = c(uv)$ for all $uv \in Q$, the corresponding flow in D with $f(uv) = f'(sv)$ is a stable flow in D saturating all edges in Q. Now assume $f'(sv) < c(uv)$ or $f'(ut) < c(uv)$ for some $uv \in Q$. Then by Theorem 1, any stable flow in D_Q has this property. Hence, no stable flow in D saturates all edges in Q. □

4.2.3 Stable Matchings with Forced Edges

We shortly discuss the case of forced edges in stable matching instances. Notice that our observations are valid in the so-called stable roommates setting, where the underlying graph is not bipartite. The definition of a blocking edge is exactly the same as in the classical bipartite case. An edge $uv \notin M$ blocks M if both u and v prefer each other to their respective partners in M.

Problem 2 SR FORCED Input: $\mathcal{I} = (G, r, Q)$; a graph G (not necessarily bipartite), the preference ordering r of vertices, and a set of forced edges Q.
Question: Is there a stable matching covering all edges in Q?

The technique described above provides a fairly simple method for solving SR FORCED, because the Rural Hospitals Theorem holds for the stable roommates problem as well [22, Theorem 4.5.2]. After deleting each forced edge $uw \in Q$ from the graph, we add uw_s and $u_t w$ edges to each of the pairs, where w_s and u_t are newly introduced vertices. These edges take over the rank of uw. Unlike in SF, here we need to introduce two separate dummy vertices to each forced edge, simply due to the matching constraints. There is a stable matching containing all forced edges if and only if an arbitrary stable matching covers all of these new vertices w_s and u_t. The proof for this is analogous to that of Lemma 3.

The running time of this algorithm is $\mathcal{O}(|E|)$, since it is sufficient to construct a single stable solution in an instance with at most $2|V|$ vertices. More vertices cannot occur, because in a matching problem more than one forced edge incident to a vertex immediately implies infeasibility. Notice that solving SR FORCED has the same time complexity $\mathcal{O}(|E|)$ as solving the stable roommates problem without any restriction on the edges.

4.3 Forbidden Edges

In order to handle SF FORBIDDEN, we present here an argumentation of the same structure as in the previous section. In Sect. 4.3.1, we show how to solve the problem of stable flows with a single forbidden edge by solving two instances on two different extended networks. Then, in Sect. 4.3.2 we show how these constructions can be used to obtain an algorithm for the case of multiple forbidden edges. Finally, in Sect. 4.3.3 we discuss the implication of our results to stable matching instances.

Now we introduce some notation used in this section. We remind the reader that P is the set forbidden edges, where $\mathfrak{l}(e) = c(e)$. For $e = uv \in P$, we define edges

$e^+ = sv$ and $e^- = ut$. We set $c(e^+) = \varepsilon > 0$ and set $r_v(e^+) = r_v(e) - \varepsilon$, i.e., e^+ occurs on v's preference list exactly before e. Likewise, we set $c(e^-) = \varepsilon$ and $r_u(e^-) = r_u(e) - \varepsilon$, i.e., e^- occurs on u's preference list exactly before e. For $F \subseteq P$ we define $E^+(F) := \{e^+ : e \in F\}$ and $E^-(F) := \{e^- : e \in F\}$.

4.3.1 A Single Forbidden Edge

Assume that $P = \{e_0\}$ for a single edge e_0. First we present two modified instances that will come handy when solving SF FORBIDDEN. The first is the graph D^+, which we obtain from D by adding the edge e_0^+ to E. Similarly, we obtain the graph D^- by adding e_0^- to E. Both graphs are illustrated in Fig. 6.

In the following, we characterize SF FORBIDDEN instances with the help of D^+ and D^-. Our claim is that SF FORBIDDEN in D has a solution if and only if there is a stable flow f^+ in D^+ with $f^+(e^+) = 0$ or there is a stable flow f^- in D^- with $f^-(e^-) = 0$. These existence problems can be solved easily in polynomial time, since all stable flows have the same value on edges incident to terminal vertices by Theorem 1.

We start with a straightforward observation, which follows from the fact that the deletion of an edge that does not carry any flow in a stable flow neither affects flow conservation nor can create blocking walks.

Observation 4 *If $f(e) = 0$ for an edge $e \in E$ and stable flow f in D, then f remains stable in $D - e$ as well.*

Now we are ready to prove the correctness of our transformation.

Lemma 4 *Let f be a flow in $D = (V, E)$ with $f(e_0) = 0$. Then f is a stable flow in D if and only if at least one of the following properties hold:*

Property 1: The flow f^+ with $f^+(e) = f(e)$ for all $e \in E$ and $f^+(e_0^+) = 0$ is stable in (V, D^+).

Property 2: The flow f^- with $f^-(e) = f(e)$ for all $e \in E$ and $f^-(e_0^-) = 0$ is stable in (V, D^-).

Proof Sufficiency of any of the two properties follows immediately from Observation 4 by deletion of e_0^+ or e_0^-, respectively, since there edges carry zero flow.

To see necessity, assume that f is a stable flow in D. By contradiction assume that neither f^+ nor f^- is stable. Then there is a blocking walk W^+ for f^+ and a blocking walk W^- for f^-. Since W^+ is not a blocking walk for f in D, it must contain e_0^+. This is only possible if W^+ starts with e_0^+, because e_0^+ starts at a terminal vertex. Similarly,

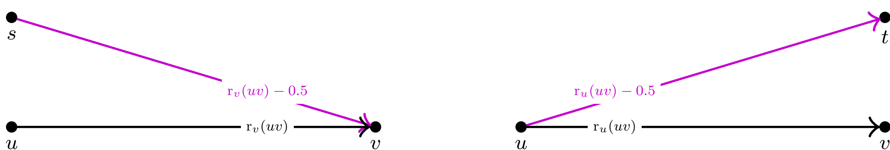

Fig. 6 Adding edges $e_0^+ = sv$ in D^+ and $e_0^- = ut$ in D^- to forbidden edge $E = uv$

since W^- is not a blocking walk for f in D, it must end with e_0^-. Let $W'^+ := W^+ \setminus \{e_0^+\}$ and $W'^- := W^- \setminus \{e_0^-\}$. Consider the concatenation $W := W'^- \circ e_0 \circ W'^+$. Note that W is an unsaturated walk in D. If $W'^- \neq \emptyset$, then W starts with the same edge as W^- and thus dominates f at the start. If $W'^- = \emptyset$, then W starts with e_0, which dominates any flow-carrying edge dominated by e_0^-, and hence it dominates f at the start also in this case. By analogous arguments it follows that W also dominates f at the end. Hence W is a blocking walk, contradicting the stability of f. We conclude that at least one of Properties 1 or 2 must be true if f is stable. □

This method can be used to solve SF FORBIDDEN if $|P| = 1$, by simply computing stable flows f^+ in D^+ and f^- in D^-. Note that by the extension of the Rural Hospitals Theorem (Theorem 1), the flow values $f^+(e_0^+)$ and $f^-(e_0^-)$ do not depend on the choice of f^+ and f^-, since they are the same for all stable flows in an instance. If $f^+(e_0^+) = 0$ or $f^-(e_0^-) = 0$, then we have found a stable flow in f avoiding the forbidden edge e_0. On the other hand, if the flow value is positive in both cases, there is no stable flow avoiding e_0.

4.3.2 Multiple Forbidden Edges

For $|P| > 1$, Lemma 4 guarantees that we can add either e^+ or e^- for each forbidden edge $e \in P$ without destroying any stable flow avoiding the forbidden edges. However, it is not straightforward to decide for which forbidden edges to add e^+ and for which to add e^-. Simply checking the two properties in Lemma 4 and creating either a D^- or D^+ graph for each forbidden edge in an arbitrary order does not lead to correct results, since the modification steps can impact each other. It is possible that the forbidden edge checked first allows for both D^- and D^+, and it turns out at a later forbidden edge that only one of these two choices can be combined with network modifications induced when tackling other forbidden edges, as the following example reveals. The same example demonstrates that adding both e^+ and e^- to all forbidden edges at the same time might lead to an instance that admits no stable flow avoiding all added edges, even though a stable flow avoiding all forbidden edges exists in the original instance. After the example we describe how to resolve this issue and obtain a polynomial time algorithm for SF FORBIDDEN.

Example 2 (Stable flows with forbidden edges) In the unit-capacity network of Fig. 7, the dashed edges u_1v_1 and u_2v_2 form P, while the thin gray edges sv_2 and u_1t are not part of the original graph but are added by the application of Lemma 4. The instance admits two stable flows. Both of them saturate all edges leaving s and all edges entering t. In the rest of the graph, stable flow f_1 is denoted by purple, and it sends one unit of flow along the edges in $\{u_1v_2, u_2v_1, u_3v_3\}$, while stable flow f_2 is denoted by green, and it sends one unit of flow along the edges in $f_2 = \{u_1v_1, u_2v_3, u_3v_2\}$. Since $u_1v_1 \in P$ is used by f_2, only f_1 avoids P. If tested separately, edge u_2v_2 fulfills both Properties 1 and 2 of Lemma 4, while u_1v_1 only fulfills Property 2. Yet requiring Property 1 for u_2v_2 and Property 2 for u_1v_1 by adding sv_1 and u_2t to the graph (as the gray edges indicate) results in a graph where every stable flow uses both sv_2 and u_1t. This is because the only stable flow in the modified network with the edges sv_2 and u_1t saturates edges su_1, su_2, su_3, sv_2, u_2v_1, u_3v_3, v_1t, v_2t, v_3t and u_1t.

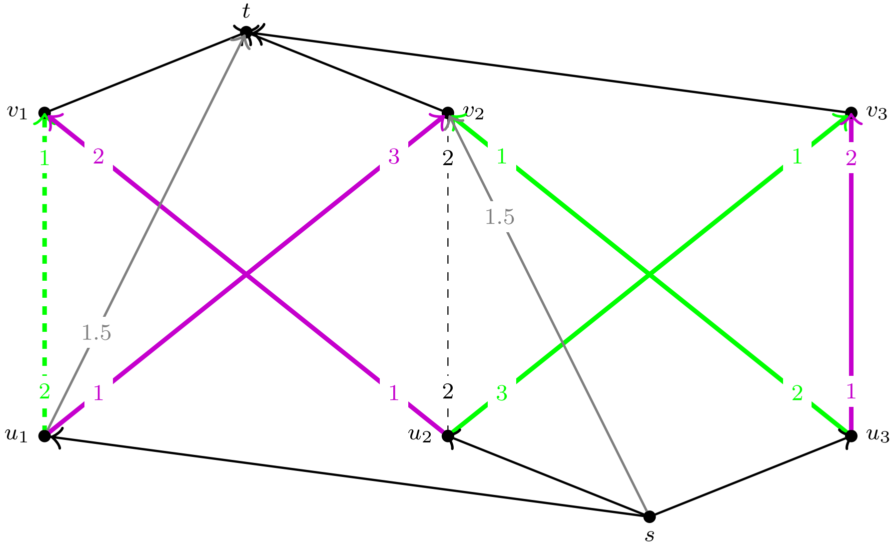

Fig. 7 The greedy algorithm fails to report the existence of a stable solution in this instance

We now sketch our algorithm that can deal with the presence of multiple forbidden edges. For any $A, B \subseteq E$, let us denote by $D[A|B]$ the network with vertices V and edges $E \cup E^+(A) \cup E^-(B)$. We remind the reader that $E^+(A) := \{e^+ : e \in A\}$ and $E^-(B) := \{e^- : e \in B\}$. Our algorithm maintains a partition of the forbidden edges in two groups P^+ and P^-. Initially $P^+ = P$ and $P^- = \emptyset$. In every iteration, we compute a stable flow f in $D[P^+|P^-]$. If $f(e^+) > 0$ for some $e \in P^+$, we move e from P^+ to P^- and repeat. If $f(e^+) = 0$ for all $e \in P^+$ but $f(e^-) > 0$ for some $e \in P^-$, we will show that no stable flow avoiding all forbidden edges exists in D. Finally, if we reach a flow f where neither of these two things happens, then f's restriction to D is a stable flow in D avoiding all forbidden edges, since $f(e^+) = 0$ or $f(e^-) = 0$ implies $f(e) = 0$ by choice of the ranks.

Algorithm 2: Stable flow with forbidden edges

```
1  Initialize P+ = P and P− = ∅.
2  repeat
3  |   Compute a stable flow f in D[P+|P−].
4  |   if ∃ e ∈ P+ with f(e+) > 0 then
5  |   |_ P+ := P+ \ {e} and P− := P− ∪ {e}
6  until f(e+) = 0 for all e ∈ P+;
7  if ∃ e ∈ P− with f(e−) > 0 then
8  |_ return ∅
9  else
10 |_ return f
```

Before proving its correctness, we present our algorithm run on the instance of Fig. 7.

Example 3 (Execution of Algorithm 2) Since $P = \{u_1v_1, u_2v_2\}$, we initialize P^+ to be $\{u_1v_1, u_2v_2\}$ and P^- to be the empty set. This defines the network $D[P^+|P^-]$, which is D complemented by sv_1 and sv_2. The stable flow f computed by Algorithm 1 in $D[P^+|P^-]$ saturates the edges sv_1, v_1t, su_2, u_2v_3, v_3t, su_3, u_3v_2, and v_2t. Since $f(sv_1) > 0$, the edge u_1v_1 is removed from P^+ and added to P^-.

In the second iteration, $D[P^+|P^-]$ is D complemented by u_1t and sv_2. The algorithm computes the stable flow in this network saturating the edges su_1, u_1t, sv_2, v_2t, su_3, u_3v_3, and v_3t. Because $f(sv_2) > 0$, the edge u_2v_2 is moved from P^+ to P^-.

In the third iteration, $D[P^+|P^-]$ is D complemented by u_1t and u_2t. The algorithm computes the stable flow in this network saturating the edges su_1, u_1v_2, v_2t, su_2, u_2v_1, v_1t, su_3, u_3v_3, and v_3t. Since $P^+ = \emptyset$ and $f(e^-) = 0$ for all $e \in P^-$, the algorithm terminates by returning this flow.

For the analysis of Algorithm 2, the following consequence of the augmenting path algorithm presented earlier (Algorithm 1) is helpful. It essentially states that removing an edge leaving s and recomputing a stable flow cannot decrease the flow value on any other edge leaving s. This observation will allow us to prove an important invariant of Algorithm 2.

Lemma 5 *Let f be a stable flow in D. Let f' be a stable flow in $D' = D - e'$ for some edge $e' \in \delta^+(s)$. Then $f'(e) \geq f(e)$ for all $e \in \delta^+(s) \setminus \{e'\}$.*

Proof We run Algorithm 1 on the networks D and D', respectively, to obtain stable flows f and f'. Recall that Algorithm 1 uses an arbitrary but fixed order of the outgoing edges of s. We choose this order such that e' comes last for the run in D. Observe that the algorithms run identically on both instances until $\pi[s]$ reaches e' for the run on D and terminates on D', respectively. Thus the flow $\bar{f}$ computed by the algorithm on D right before $\pi[s]$ is advanced to e' is identical to f'. Further note that the algorithm does not increase the flow value on any edge $e \in \delta^+(s) \setminus \{e'\}$ after $\pi[s]$ has passed e, which comes before e' by our choice of preferences. Hence $f(e) \leq \bar{f}(e) = f'(e)$. □

Lemma 6 *Algorithm 2 maintains the following invariant. There is a stable flow in D avoiding P if and only if there is a stable flow in $D[\emptyset|P^-]$ avoiding $P^+ \cup E^-(P^-)$.*

Proof Clearly, the invariant holds initially as $P^+ = P$ and $P^- = \emptyset$. Now consider any later iteration of the algorithm in which P^+, P^- are changed. Let f_0 be the computed stable flow in $D[P^+|P^-]$ and let e_0 be the edge with $f_0(e_0^+) > 0$ found in that iteration. Let P^+_{old}, P^-_{old} and P^+_{new}, P^-_{new} denote the partition before and after the update, i.e., $P^+_{\text{new}} = P^+_{\text{old}} \setminus \{e_0\}$ and $P^-_{\text{new}} = P^-_{\text{old}} \cup \{e_0\}$.

If there is a stable flow in $D[\emptyset|P^-_{\text{new}}]$ avoiding $P^+_{\text{new}} \cup E^-(P^-_{\text{new}})$, then this flow also avoids P, as for every $e \in P$ either $e \in P^+_{\text{new}}$ or $e^- \in E^-(P^-_{\text{new}})$ (note that in the latter case e^- dominates e at the start and ends at a terminal).

Conversely, if there is a stable flow in D avoiding P, then by induction hypothesis there is a stable flow f in $D[\emptyset|P^-_{\text{old}}]$ avoiding $P^+_{\text{old}} \cup E^-(P^-_{\text{old}})$. Note that e_0^+ starts at a terminal and recall that $f_0(e_0^+) > 0$ for the stable flow f_0 in $D[P^+_{\text{old}}|P^-_{\text{old}}]$. By repeated

application of Lemma 5, deleting every $e^+ \in E^+(P^+_{\text{old}} \setminus \{e_0\})$ from $D[P^+_{\text{old}}|P^-_{\text{old}}]$, we obtain that $f'(e^+_0) > 0$ for every stable flow f' in $D[\{e_0\}|P^-_{\text{old}}]$. In particular, this means that Property 1 of Lemma 4 fails for f and e_0. Therefore, by Lemma 4, Property 2 must hold for f, i.e., the extension of f to $D[\emptyset|P^-_{\text{old}} \cup \{e^-_0\}] = D[\emptyset|P^-_{\text{new}}]$ with $f(e^-_0) = 0$ is a stable flow avoiding $P^+_{\text{old}} \cup E^-(P^-_{\text{old}}) \cup \{e^-_0\}$. As $P^+_{\text{new}} \subseteq P^+_{\text{old}}$ and $E^-(P^-_{\text{new}}) = E^-(P^-_{\text{old}}) \cup \{e^-_0\}$, this completes the induction.

Lemma 7 *If Algorithm 2 returns $\emptyset$, then no stable flow in D avoids P.*

Proof If the algorithm returns $\emptyset$, then the algorithm computed a stable flow f in $D[P^+|P^-]$ with $f(e^+) = 0$ for all $e \in P^+$ but $f(e^-) > 0$ for some $e \in P^-$. Note that by Observation 4, the restriction of f is also stable in $D[\emptyset|P^-]$. As e^- is incident to a terminal, $f(e^-) > 0$ for every stable flow in $D[\emptyset|P^-]$. Therefore, by Lemma 6, there is no stable flow in D avoiding P.

Lemma 8 *If Algorithm 2 returns flow f, then f is stable in D and it avoids P.*

Proof If the algorithm returns flow f then $f(e^+) = 0$ for all $e \in P^+$ and $f(e^-) = 0$ for all $e \in P^-$. Hence the restriction of f to E is stable and avoids $P^+ \cup P^- = P$.

The correctness of Algorithm 2 follows immediately from the above lemmas. The running time of this algorithm is bounded by $\mathcal{O}(|P||E|\log|V|)$, as each stable flow f can be computed in $\mathcal{O}(|E|\log|V|)$ time and in each round either $|P^+|$ decreases by one or the algorithm terminates.

4.3.3 Stable Matchings with Forbidden Edges

Just as earlier, in Sect. 4.2.3, we finish this part with the direct interpretation of our results in the stable marriage instances.

Problem 3 SM FORBIDDEN Input: $\mathcal{I} = (G, r, P)$; a bipartite graph G, the preference ordering r of vertices, and a set of forbidden edges P.
Question: Is there a stable matching avoiding all edges in P?

Let $A \cup B$ be the bipartition of the vertices. One possibility to solve SM FORBIDDEN would be to transform it into an instance of SF FORBIDDEN by the standard transformation of bipartite matching to flow (directing all edges from A to B and augmenting the graph by a super source and a super sink connected to all vertices in A and B, respectively). Running Algorithm 2 on this instance gives a stable flow that can be transformed into a matching in the original instance.

However, we can adapt the Algorithm 2 to directly run on the matching instance as follows. For forbidden each edge $e \in P$ we introduce a new vertex v_e. We maintain a partition of P into sets P_A and P_B, with $P_A = P$ and $P_B = \emptyset$ initially. For each $e = ab \in P_A$ we introduce the edge av_e to the graph with $r_a(av_e) = r_a(ab) - \varepsilon$, and for each edge $e = ab \in P_B$ we introduce the edge bv_e instead with $r_b(bv_e) = r_b(ab) - \varepsilon$. We then compute a stable matching in the resulting graph. If an edge av_e is in the matching for some $e \in P_A$ we remove e from P_A and add it to P_B. We then

again compute a stable matching and repeat this procedure until no edge av_e is in the matching for any $e = ab \in P_A$.

If in the resulting matching the vertices v_e for $e \in P$ are unmatched, i.e., also no edge bv_e is used for any $e = ab \in P_B$, the matching is stable in the original graph and it does not use any edge in P (due to the choice of the ranks). If not, using the same line of argumentation as in the proof of Lemma 6 we can show that no stable matching avoiding P exists. (Here, the bipartite structure of the graph yields a straightforward analogue of Lemma 5. We remark that it is an open problem how to adapt this technique to the stable roommates problem for non-bipartite graphs.)

Our algorithm for several forbidden edges runs in $\mathcal{O}(|P||E|)$ time, because computing stable matchings in each of the at most $|P|$ rounds takes only $\mathcal{O}(|E|)$ time. With this running time, it is somewhat slower than the best known methods [9,14] that require only $\mathcal{O}(|E|)$ time, but it is a reasonable assumption that the number of forbidden edges is small.

4.4 Forced and Forbidden Edges

If both forced and forbidden edges occur in the same instance, then they can be handled by our two algorithms, applying them one after the other. First, all forced edges in the graph D are substituted by the construction discussed in Sect. 4.2.2, obtaining the graph D_Q where the edges in Q are replaced by artificial edges $\bar{Q}$. The following corollary is a direct implication of Theorem 3.

Corollary 1 *There is a stable flow in D saturating all edges in Q and avoiding all edges in P if and only if there is a stable flow in D_Q saturating all edges in $\bar{Q}$ and avoiding all edges in P.*

We now run Algorithm 2 from Sect. 4.3.2 on D_Q. If the algorithm asserts that no stable flow in D_Q avoiding P exists, then by Corollary 1, there is no stable flow in D saturating all edges in Q and avoiding all edges in P. If, instead, the algorithm returns a stable flow f' avoiding P, we check whether it also saturates all edges in $\bar{Q}$. If this is the case, the corresponding flow in D is a stable flow avoiding P and saturating all edges in Q. If there is an edges $e \in \bar{Q}$ with $f'(e) < c(e)$, then this is true for every stable flow in D_Q by the Rural Hospital Theorem (Theorem 1) and hence, no flow saturating all edges in Q exists in D.

The procedure described above runs in time $\mathcal{O}(|P||E| \log |V|)$, as D_Q can be constructed in time linear in $|E|$ and the number of edges and vertices in D_Q is at most twice the number of edges and vertices in D, respectively (remember that we already argued in Remark 1 that the initial transformation of the instance in Sect. 4.1 does not change this asymptotic running time). We conclude the following result:

Theorem 5 SF RESTRICTED *can be solved in $\mathcal{O}(|P||E| \log |V|)$ time.*

5 Stable Multicommodity Flows

In this section we turn our attention to stable multicommodity flows. We first present the original definition of this concept by Király and Pap [27] and outline their results, including the existence of a stable solution. We then proceed to our results: a reduction of the general model to a much simpler special case and a hardness proof for deciding the existence of an integral solution.

5.1 Problem Definition

Multicommodity networks model scenarios in which a common network is used by several commodities. For example, roads serve personal vehicles, and also various sorts of commercial transport vehicles. While each person and each type of goods has its own origin and destination, they all share the same roads, which have a capacity on all vehicles altogether and sometimes also separately on a specific type of vehicle.

A *multicommodity network* (D, c^i, c), $1 \leq i \leq n$ consists of a directed graph $D = (V, E)$, non-negative commodity capacity functions $c^i : E \to \mathbb{R}_{\geq 0}$ for all the n commodities and a non-negative cumulative capacity function $c : E \to \mathbb{R}_{\geq 0}$ on E. For every commodity i, there is a *source* $s^i \in V$ and a *sink* $t^i \in V$, also referred to as the *terminals of commodity* i.

Definition 3 (*multicommodity flow*) A set of functions $f^i : E \to \mathbb{R}_{\geq 0}$, $1 \leq i \leq n$ is a *multicommodity flow* if it fulfills all of the following requirements:

1. Capacity constraints for commodities:
   $f^i(uv) \leq c^i(uv)$ for all $uv \in E$ and commodity i;
2. Cumulative capacity constraints:
   $f(uv) = \sum_{1 \leq i \leq n} f^i(uv) \leq c(uv)$ for all $uv \in E$;
3. Flow conservation:
   $\sum_{uv \in E} f^i(uv) = \sum_{vw \in E} f^i(vw)$ for all $i : 1 \leq i \leq n$ and $v \in V \setminus \{s^i, t^i\}$.

The concept of stability was extended to multicommodity flows by Király and Pap [27]. A stable multicommodity flow instance $\mathcal{I} = (D, c^i, c, r_E, r^i_V)$, $1 \leq i \leq n$ comprises a network (D, c^i, c), $1 \leq i \leq n$, *edge preferences* r_E over commodities, and *vertex preferences* r^i_V, $1 \leq i \leq n$ over incident edges for commodity i. Each edge uv ranks all commodities in a strict order of preference. Separately for every commodity i, each non-terminal vertex ranks its incoming and also its outgoing edges strictly with respect to commodity i. Note that these preference orderings of v can be different for different commodities and they do not depend on the edge preferences r_E over the commodities. If edge uv prefers commodity i to commodity j, then we write $\mathrm{r}_{uv}(i) < \mathrm{r}_{uv}(j)$. Analogously, if vertex v prefers edge vw to vz with respect to commodity i, then we write $\mathrm{r}^i_v(vw) < \mathrm{r}^i_v(vz)$. We denote the flow value with respect to commodity i by $f^i = \sum_{u \in V} f^i(s^i u)$.

Definition 4 (*stable multicommodity flow*) A *blocking walk with respect to commodity* i of a multicommodity flow f is a directed walk $W = \langle v_1, v_2, \ldots, v_k \rangle$ such that all of the following properties hold:

1. $f^i(v_j v_{j+1}) < c^i(v_j v_{j+1})$ for each edge $v_j v_{j+1}$, $j = 1, \ldots, k-1$;
2. $v_1 = s^i$ or there is an edge $v_1 u$ such that $f^i(v_1 u) > 0$ and $\mathrm{r}^i_{v_1}(v_1 v_2) < \mathrm{r}^i_{v_1}(v_1 u)$;
3. $v_k = t^i$ or there is an edge $w v_k$ such that $f^i(w v_k) > 0$ and $\mathrm{r}^i_{v_k}(v_{k-1} v_k) < \mathrm{r}^i_{v_k}(w v_k)$;
4. If $f(v_j v_{j+1}) = c(v_j v_{j+1})$, then there is a commodity i' such that $f^{i'}(v_j v_{j+1}) > 0$ and $\mathrm{r}_{v_j v_{j+1}}(i) < \mathrm{r}_{v_j v_{j+1}}(i')$.

A multicommodity flow is *stable*, if there is no blocking walk with respect to any commodity.

In words, a walk blocks the multicommodity flow with respect to commodity i if both the starting and end vertices of the walk are willing to reroute some units of flow of commodity i along it, moreover, the edges along the walk either have free capacity for forwarding these or they are inclined to drop some units of flow of another commodity. This last point can be seen as a clear difference to single-commodity stable flows. Due to point 4, Definition 4 allows saturated edges to occur in a blocking walk with respect to commodity i, provided that these edges are inclined to trade in some of their forwarded commodities for more flow of commodity i. On the other hand, the role of edge preferences is limited: blocking walks still must start at vertices who are willing to reroute or send extra flow along the first edge of the walk according to their vertex preferences with respect to commodity i.

Problem 2 SMF Input: $\mathcal{I} = (D, c^i, c, r_E, r^i_V)$, $1 \le i \le n$; a directed multicommodity network (D, c^i, c), $1 \le i \le n$, edge preferences over commodities r_E and vertex preferences over incident edges r^i_V, $1 \le i \le n$.
Question: Is there a stable multicommodity flow?

Theorem 6 (Király and Pap [27]) *A stable multicommodity flow exists for any instance, but it is PPAD-hard to find.*

Király and Pap use a polyhedral version of Sperner's lemma [26] to prove the existence result. PPAD-hardness [31] is considered a somewhat weaker evidence of intractability than NP-hardness that applies for problems whose decision versions have a 'yes' answer for sure. Note that SMF is one of the very few problems in stability [3] where a stable solution exists, but no extension of the Gale–Shapley algorithm is known to solve it—not even a variant with exponential running time.

5.2 Problem Simplification

The definition of SMF involves many distinct components and constraints. It is natural to investigate how far the model can be simplified without losing any of its generality. In particular, Király and Pap [27] pose an open question on the PPAD-hardness of the problem if there are no individual capacities. Here we give a positive answer to this and further intuitive questions on possible restricted cases. It turns out that the majority of the commodity-specific input data can be dropped, as shown by Theorem 7. This result not only simplifies the instance, but it also sheds light to the most important characteristic of the problem, which seems to be the preference ordering of edges over commodities.

Theorem 7 *There is a polynomial-time transformation that, given an instance $\mathcal{I}$ of* SMF*, constructs an instance $\mathcal{I}'$ of* SMF *with the following properties:*

1. *All commodities have the same source and sink,*
2. *At each vertex, the preference lists are identical for all commodities,*
3. *There are no commodity-specific edge capacities,*

and there is a polynomially computable bijection between the stable multicommodity flows of $\mathcal{I}$ and the stable multicommodity flows of $\mathcal{I}'$. The bijection preserves integrality.

Proof We present the construction in three steps, each ensuring one of the properties without destroying those established before.

1. *All commodities have the same source and sink* We introduce two new super terminals s^* and t^*. These will substitute all commodity-specific sources and sinks. For every commodity i and its terminals s^i and t^i, we introduce the edges s^*s^i and t^it^* with capacities $c^i(s^*s^i) = c(s^*s^i) = \sum_{e\in\delta^+(s^i)} c(e)$ and $c^i(t^it^*) = c(t^it^*) = \sum_{e\in\delta^-(t^i)} c(e)$. These edges cannot carry any other commodity: $c^j(s^*s^i) = c^j(t^it^*) = 0$ for all $j \neq i$. We assign arbitrary ranks to the edges originally incident to s^i or t^i and put s^*s^i and t^it^* to the end of the preference list of s^i and t^i for all commodities. Finally, we set s^* and t^* as source and sink for every commodity i. It is easy to verify that a flow f is stable in the original network D if and only if the natural extension of f to the added edges is a stable flow.
2. *At each vertex, the preference lists over the edges are identical for all commodities* The main idea here is to substitute every edge by a gadget that separates different commodities. Then the edges can be ranked in a single preference list, since each edge is designated to carry its own commodity only and for edges carrying a specific commodity, the list on other edges is irrelevant.
   For any $e \in E$, we remove $e = uv$ from the graph and replace it by the construction shown in Fig. 8. We introduce two new vertices v'_e and v''_e and add the edge $v'_ev''_e$ with $c(v'_ev''_e) = c^i(v'_ev''_e) = c(e)$ for every commodity i. We also add n new edges e'_i for $1 \le i \le n$ from u to v'_e. We set $c(e'_i) = c^i(e'_i) = c^i(e)$, $c^j(e'_i) = 0$ for $j \neq i$, and $\mathrm{r}_u(e'_i) = |E|i + \mathrm{r}^i_u(e)$. We choose $\mathrm{r}_{v'_e}(e'_i)$ arbitrarily. Likewise, we add n new edges e''_i for $1 \le i \le n$ from v''_e to v. We set $c(e''_i) = c^i(e''_i) = c^i(e)$, $c^j(e''_i) = 0$ for $j \neq i$, and $\mathrm{r}_v(e''_i) = |E|i + \mathrm{r}^i_v(e)$. We choose $\mathrm{r}_{v''_e}(e'_i)$ arbitrarily. Let D' be the network resulting from this modification.
   If f is a stable flow in D, then we define a flow f' in D' as follows. For every commodity i and every $e \in E$, we set $f'^i(e'_i) = f'^i(v'_ev''_e) = f'^i(e''_i) = f^i(e)$ and we set $f'^j(e'_i) = f'^j(e''_i) = 0$ for $j \neq i$. It is easy to check that f' is a stable flow in D' and that the mapping from f to f' is a bijection between stable flows in D and D'.
3. *There are no commodity-specific capacities* Finally we ensure that $c^i(e) = c(e)$ for all i and all $e \in E$, which implies that the commodity-specific capacities do not play any role. To this end, we introduce a new commodity i^*. Each edge will be replaced by a gadget in which the capacity on a specific commodity translates

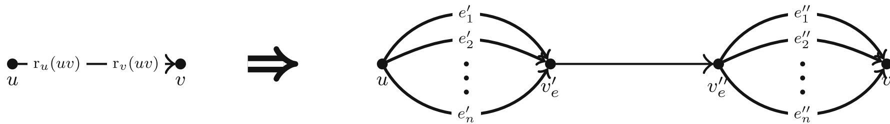

Fig. 8 The gadget ensuring that the preference lists of each vertex are identical for all commodities

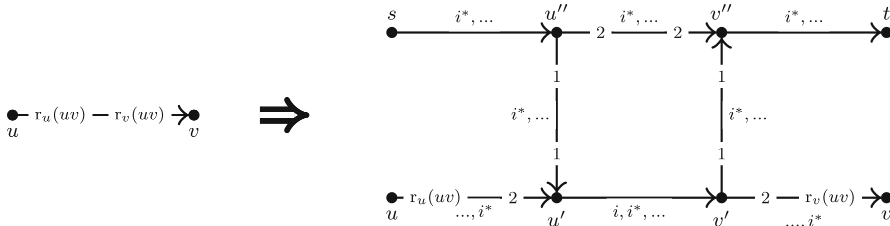

Fig. 9 The gadget ensuring that there are no commodity-specific capacities

into an edge willing to carry i^* rather than forwarding more flow of the specific commodity.

Note that the transformation described in point 2 above already ensures that for every edge $e \in E$ one of the following is true: Either $c^i(e) = c(e)$ for all i, or there is an i such that $c^i(e) = c(e)$ and $c^j(e) = 0$ for all $j \neq i$. We only have to deal with the latter case, that is, edge e being designated to carry commodity i only, up to its full capacity. Let edge e and commodity i be such a pair.

We replace $e = uv$ by the gadget $H_{e,i}$, depicted in Fig. 9. First, four new vertices u', u'', v' and v'' are introduced. We add the edges uu', $u'v'$, $v'v$, su'', $u''v''$, $v''t$, $u''u'$ and $v'v''$, all with capacity $c(e)$. For the edges su'', $u''v''$, $v''t$, $u''u'$ and $v'v''$ the new commodity i^* is on top of their preference list, followed by all other commodities in arbitrary order. For edge $u'v'$ commodity i is first on the list, i^* is second, followed by all other commodities in arbitrary order. For the edges uu' and $v'v$, commodity i^* is last on the list, the rank of the other commodities is arbitrary. For the vertex preferences, we set $\mathrm{r}_{u''}(u''u') < \mathrm{r}_{u''}(u''v'')$ and $\mathrm{r}_{v''}(v'v'') < \mathrm{r}_{v''}(u''v'')$, as well as $\mathrm{r}_{u'}(u''u') < \mathrm{r}_{u'}(uu')$ and $\mathrm{r}_{v'}(v'v'') < \mathrm{r}_{v'}(v'v)$. We further set $\mathrm{r}_u(uu') = \mathrm{r}_u(e)$ and $\mathrm{r}_v(v'v) = \mathrm{r}_v(e)$.

Let us denote the modified network by $\bar{D}$. For a stable flow f in the original network D, we define a flow $\bar{f}$ in $\bar{D}$ as follows. For edges e that were not replaced by a gadget in $\bar{D}$, we set $\bar{f}^i(e) = f^i(e)$ for all i. For every e that was replaced by a gadget (because $c^i(e) = c(e)$ and $c^j(e) = 0$ for all $j \neq i$), we set the flow values within the gadget as follows. For the new commodity i^* we set $\bar{f}^i(uu') = \bar{f}^i(u'v') = \bar{f}^i(v'v) = f^i(e)$, and we set $\bar{f}^{i*}(u''u') = \bar{f}^{i*}(u'v') = \bar{f}^{i*}(v'v'') = c(e) - f^i(e)$, so that $u'v'$ is saturated with its two top-ranked commodities. Furthermore we set $\bar{f}^{i*}(su'') = \bar{f}^{i*}(v''t) = c(e)$, and $\bar{f}^{i*}(u''v'') = f^i(e)$. All other flow values are set to zero within the gadget (recall that $f^j(e) = 0$ for all $j \neq i$).

Claim *The flow $\bar{f}$ is stable in $\bar{D}$.*

Proof We have constructed $\bar{f}$ so that it respects all capacities and fulfills flow conservation in $\bar{D}$. To see that $\bar{f}$ is a stable flow, assume by contradiction that there is an $\bar{f}$-blocking walk $\bar{W}$ for some commodity j.

First assume $\bar{W}$ starts in the interior of a gadget, i.e., with an edge of a gadget $H_{e,i}$ different from uu'. We eliminate the edges of the gadget one by one to show that this is not possible.

- $\bar{W}$ cannot start with su'', as this edge is saturated with its most preferred commodity i^*.
- $\bar{W}$ also cannot start with $u''v''$, $u'v'$, or $v'v$, as these edges are the last-choice outgoing edges on the preference lists of u'', u' and v' respectively.
- If $\bar{W}$ starts at $u''u'$, then $j = i^*$, because this is the only commodity on the dominated edge $u''v''$. But then $\bar{W}$ must end at u' because $u'v'$ is saturated with commodities it ranks at least as high as i^*. However, $\bar{f}^{i^*}(uu') = 0$, so $\bar{W}$ does not dominate f at u'.
- Finally, if $\bar{W}$ starts with $v'v''$, then $j \neq i^*$ because $\bar{f}^{i^*}(v'v) = 0$. But it can neither end at v'' as v'' only receives commodity i^* from $u''v''$, nor can it continue as $v''t$ is saturated with its favorite commodity.

We conclude that $\bar{W}$ cannot start in the interior of a gadget. By a symmetric argument, $\bar{W}$ cannot end in the interior of a gadget, i.e., with an edge of a gadget $H_{e,i}$ different from $v'v$.

Thus, if $\bar{W}$ contains any edge of a gadget $H_{e,i}$, it must traverse all the edges uu', $u'v'$, $v'v$ of the gadget. As $u'v'$ is saturated with commodities i and i^*, we conclude that $j = i$ and $c(e) - f^i(e) = \bar{f}^{i^*}(u'v') > 0$. We replace all such segments uu', $u'v'$, $v'v$ from any traversed gadget $H_{e,i}$ with the corresponding edge e and get a walk W in D. Because $f^i(e) < c(e)$ for all inserted edges, W is a blocking walk for f, contradicting the stability of f. □

It is easy to see that the mapping defined by $\phi(f) = \bar{f}$ is injective, and as argued above, preserves stability. We now show that it is indeed a bijection from stable flows in D to stable flows in $\bar{D}$.

Claim *For any stable flow y in $\bar{D}$, there is a stable flow f in D with $\phi(f) = y$.*

Proof Let y be a stable flow in $\bar{D}$. Consider a gadget $H_{e,i}$. By contradiction assume $y^{i^*}(uu') > 0$. Then $y^{i^*}(su'') = y^{i^*}(u''u') = c(e)$ as otherwise either $\langle s, u'', u' \rangle$ or $\langle u'', u' \rangle$ is a blocking walk for commodity i^*. But then $y^{i^*}(uu') + y^{i^*}(u''u') > c(e) \geq y^{i^*}(u'v')$, contradicting flow conservation. Hence $y^{i^*}(uu') = 0$ and, by a symmetric argument, $y^{i^*}(v'v) = 0$. As no flow of commodity i^* enters or leaves $H_{e,i}$, and the path $\langle s, u'', v'', t \rangle$ is not blocking, we conclude that $y^{i^*}(su'') = y^{i^*}(v''t) = c(e)$. By flow conservation, $y^{i^*}(u''u') = y^{i^*}(u'v') = y^{i^*}(v'v'') = c(e) - y^{i^*}(u''v'')$. Since the path $\langle u'', u', v', v'' \rangle$ is not blocking and i is the only commodity that comes before i^* on an edge of that path, we conclude that $y^{i^*}(u'v') + y^i(u'v') = c(e)$. Hence, by flow conservation, $y^i(uu') = y^i(u'v') = y^i(v'v) = c(e) - y^{i^*}(v'v'')$, and $y^j(e') = 0$ for all $j \notin \{i, i^*\}$ and all edges e' in the gadget $H_{e,i}$.

Now define f by setting $f^i(e) = y^i(u'v')$ for every gadget $H_{e,i}$ in $\bar{D}$ and $f^i(e) = y^i(e)$ for all edges in $E \cap E_{\bar{D}}$ and all commodities i. Using the above observations, it

is easy to check that $\phi(f) = y$ and that f fulfills flow conservation and respects all capacity constraints (in particular $f^j(e) = y^j(u'v') = 0$ for all $j \neq i$ at any gadget $H_{e,i}$). To see that f is a stable flow, assume by contradiction that there is a blocking walk W for f and commodity i. We obtain a walk $\bar{W}$ in $\bar{D}$ by replacing the edges of W with the corresponding gadgets $H_{e,i}$. At any such edge, $f^i(e) < c(e)$ because W is blocking with respect to i and i is the only commodity that can traverse e. Hence, $y^i(uu') = y^i(u'v') = y^i(v'v) < c(e)$. Also, as the preference lists of non-gadget vertices are the same in D and $\bar{D}$, $\bar{W}$ is indeed a blocking walk for y contradicting its stability. □

It is easy to check that all transformations described above can be carried out in polynomial time and that integral stable flows in the original graph correspond to integral stable flows in the transformed graph.

5.3 Integral Multicommodity Stable Flows

First we modify Definition 2 so that it describes the integral version of SMF. Then we carefully analyze an example network with no integral solution. This network is used in the last part of this subsection, in which we present our hardness proof.

Problem 3 ISMF Input: $\mathcal{I} = (D, c^i, c, r_E, r_V^i), 1 \leq i \leq n$; a directed multicommodity network (D, c^i, c), $1 \leq i \leq n$, edge preferences over commodities r_E and vertex preferences over incident edges r_V^i, $1 \leq i \leq n$.
Question: Is there a stable multicommodity flow with integral $f^i(uv)$ values for all $uv \in E$ and $1 \leq i \leq n$?

Király and Pap [27] give, for every integer N, an example instance with N commodities and N vertices, where no stable multicommodity flow exists with denominators at most N. Here we present a small and slightly modified version of that instance as an example and later use it as a gadget in our hardness proof.

Example 4 (ISMF instance with no solution) Consider the network depicted in Fig. 10. We consider two variants of an ISMF instance in this network. In both cases, u is the only terminal vertex in the graph, but the variants differ in that either 3 or only 2 commodities are present:

1. $s^1 = s^2 = s^3 = t^1 = t^2 = t^3 = u$ (see Lemma 9) and
2. $\exists i \in \{1, 2, 3\} : \{s^i, t^i\} = \emptyset$ (see Lemma 10).

We will show below that in the first case, the instance admits no integer multicommodity flow, whereas such a flow exists in the second case.

The edge capacities with respect to commodities are 1 for the commodities that appear in r_E for the specific edge and 0 for the remaining commodities. All edges have cumulative capacity 1. The vertex preferences are the same for all commodities: v_1, v_2 and v_3 are inclined to receive and send the flow along the edges between themselves rather than trading with u. Each commodity i has a unique feasible cycle C^i through u and it is easy to see that due to the choice of the c^i functions, no other cycle or terminal-terminal path exists in the network.

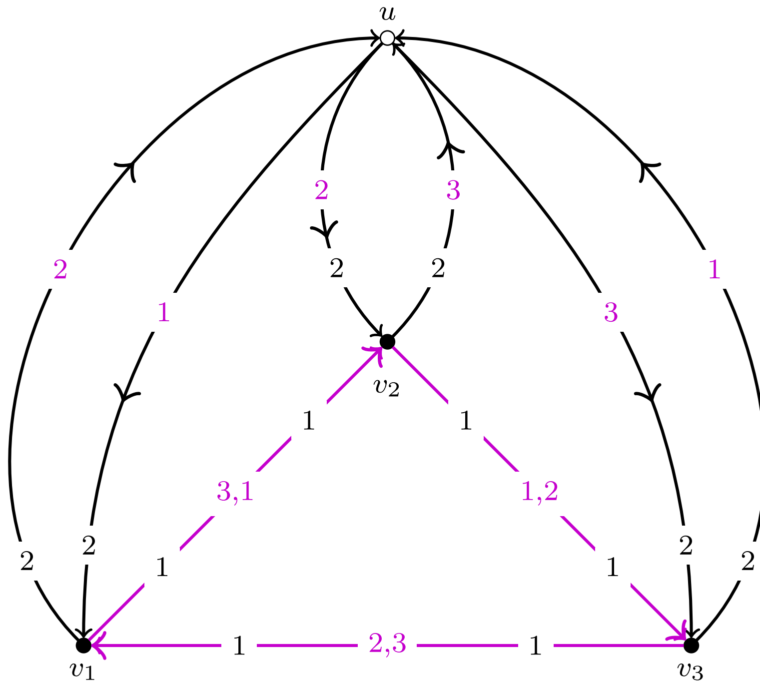

Fig. 10 The edge preferences are marked with colored labels in the middle of edges, while r_V^i is black and closer to the vertices. For all edges, $c = 1$. The purple edges of the triangle can forward two commodities, while the bent black edges can carry only one commodity (Color figure online)

- $C^1 = \langle u, v_1, v_2, v_3, u \rangle$
- $C^2 = \langle u, v_2, v_3, v_1, u \rangle$
- $C^3 = \langle u, v_3, v_1, v_2, u \rangle$

Lemma 9 *If $s^1 = s^2 = s^3 = t^1 = t^2 = t^3 = u$, then there is no integer stable multicommodity flow.*

Proof Assume that there is an integral stable multicommodity flow f in the instance. The empty flow cannot be f, because there is a cycle running through u for each commodity and such cycles block the empty flow. Without loss of generality we can now assume that C^1 is saturated by commodity 1:

$$f^1(uv_1) = f^1(v_1v_2) = f^1(v_2v_3) = f^1(v_3u) = 1,$$

while all other flow values must be 0 due to commodity capacity constraints on edges. This flow is blocked by commodity 3 on the cycle $\langle u, v_3, v_1, v_2, u \rangle$. It is easy to see that analogous arguments work for C^2 and C^3 as well. Thus, no integer stable flow exists in the graph. □

Lemma 10 *If u is a terminal for at most two out of the three commodities, then an integer stable multicommodity flow exists.*

Proof Let us now investigate the same instance with a slight modification: $s^1 = s^2 = t^1 = t^2 = u$, but $\{s^3, t^3\} = \emptyset$. Then, the following integer flow is stable:

$$f^1(uv_1) = f^1(v_1v_2) = f^1(v_2v_3) = f^1(v_3u) = 1.$$

A blocking walk with respect to commodity 1 cannot exist, because all edges that can carry commodity 1 also carry it to their upper capacity. Commodity 2 could block along C^2, but edge v_2v_3 is saturated with its most preferred commodity. It is trivial that the same flow remains stable if we set $s^1 = t^1 = u$ and $\{s^2, t^2\} = \{s^3, t^3\} = \emptyset$. If $\{s^1, t^1\} = \{s^2, t^2\} = \{s^3, t^3\} = \emptyset$, then the empty flow is stable. □

To sum up the established results about Example 4: the instance admits an integer stable flow if and only if u has at most two commodities. This argument will help us prove a claim later in our hardness proof.

Theorem 8 *Deciding whether* ISMF *has a solution is* NP*-complete. This holds even if all commodities share the same set of terminal vertices, all vertices have the same preferences with respect to all commodities, and edges do not have commodity-specific capacities* (*but edges have preferences over different commodities*).

Proof In the following, we show NP-completeness for the general version ISMF. By Theorem 7, this also implies NP-completeness for ISMF restricted to instances with identical terminal sets, commodity-independent vertex preferences, and without commodity-specific edge capacities.

Testing whether a feasible integral multicommodity flow is stable can be done in polynomial time, as pointed out also in [27]. It is sufficient to check the existence of edges fulfilling points 2 and 3 in Definition 4 for every commodity and then execute a breadth-first search for every pair of vertices as v_1 and v_k vertices of the potential blocking walk. Thus ISMF is in NP.

We now describe how to construct an ISMF instance $\mathcal{I}'$ from any given instance $\mathcal{I}$ of 3- SAT with n variables and m clauses, also illustrated in Fig. 11. For each variable i in the Boolean formula we create 2 commodities, i and $\bar{i}$, corresponding to truth values true and false. To simplify notation, we say that $\bar{\bar{i}} = i$. Every clause in the formula is assigned a clause gadget, identical to the instance presented in Example 4, but with u being a non-terminal for all commodities. The three *relevant commodities* are the commodities corresponding to the *negations* of the three literals appearing in the clause. The preferences of u in such a gadget are chosen so that the edges of the gadget are preferred to edges outside of the gadget. The order of the edges at u inside the gadget is irrelevant due to the commodity-specific capacity constraints.

All commodities share the same terminals s and t. There is a long path running from s to t, consisting of three segments. The first and the third segments are two disjoint copies of the same variable gadget, while the second segment consists of the u-vertices of the m clause gadgets. A variable gadget is defined on vertices $\{a, b_1, b_2, \ldots, b_n, d\}$ with edges ab_i and b_id for all i. For each i and each $e \in \{ab_i, b_id\}$ we set the capacities $c^i(e) = c^{\bar{i}}(e) = c(e) = 1$ and $c^j(e) = c^{\bar{j}}(e) = 0$ for $j \neq i$. Edge ab_i ranks commodity i best, and $\bar{i}$ second, while b_id ranks commodity $\bar{i}$ best, and

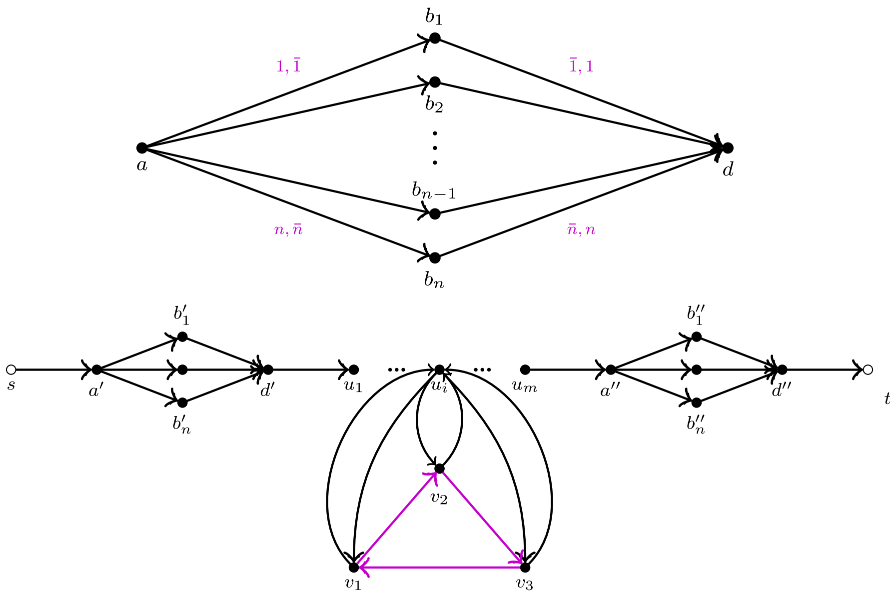

Fig. 11 A variable gadget and the entire construction for ISMF

i second. The vertex preferences of a and d are arbitrary. These three segments are chained together so that the only edge of s ends at a' in the first variable gadget, d' in the same gadget is connected to the first u vertex of the second segment, the last u of the same segment is adjacent to a'' in the second variable gadget and d'' in this gadget has an edge running to t. For the edges connecting the segments and the u-vertices of clause gadgets with each other and with the terminals, the capacities are set to $c^i = c^{\bar{i}} = c = n$ for all $1 \le i \le n$, and edge preferences are chosen arbitrarily.

Having described the full construction we now prove in Lemmas 11 and 12 the equivalence between the existence of an integral stable multicommodity flow in $\mathcal{I}'$ and a satisfying truth assignment in $\mathcal{I}$.

Lemma 11 *If an integral stable multicommodity flow f exists in $\mathcal{I}'$, then there is a satisfying truth assignment in $\mathcal{I}$.*

Proof As defined after Definition 3, f^i denotes the total flow value with respect to commodity i.

Claim *For every commodity i, $f^i + f^{\bar{i}} = 1$.*

Proof If $f^i(ab_i) + f^{\bar{i}}(ab_i) < 1$ for some commodity i and edge ab_i of a variable gadget, then there is an unsaturated s-t path through b_i with respect to commodity i, because the edges ab_i and $b_i d$ are not saturated and all other edges along the main path have capacity n. This path blocks f. Since $c(ab_i) = 1$ for every $1 \le i \le n$, $f^i(ab_i) + f^{\bar{i}}(ab_i) = 1$, thus edges ab_i and $b_i d$ of the variable gadgets are saturated with commodities i and $\bar{i}$. This already implies that $f^i + f^{\bar{i}} = 1$ for every $1 \le i \le n$. □

This claim allows us to assign exactly one truth value to each variable: x_i is true if $f^i = 1$ and it is false if $f^{\bar{i}} = 1$.

Claim *For every clause $C = x_i \vee x_j \vee x_k$, where the variables in C can be in negated or unnegated form, $f^{\bar{i}} + f^{\bar{j}} + f^{\bar{k}} \leq 2$, for every $1 \leq i, j, k \leq n$.*

Proof Since u prefers sending flow along its edges in the gadget over forwarding it to the next u vertex on the path, u can be seen as a terminal vertex with respect to the commodities reaching it. As we have shown in Example 4, if there is a solution to ISMF, then at most two of the three relevant commodities are present at u. □

The latter claim is the reason why we took the negated version of each literal in the clause: at most two literals are false in each clause, thus the clause is satisfied by the truth assignment.

Lemma 12 *If there is a satisfying truth assignment in $\mathcal{I}$, then there is an integral stable multicommodity flow f in $\mathcal{I}'$.*

Proof The constructed flow to the given truth assignment is the following. For every variable i, $f^i = 1$, $f^{\bar{i}} = 0$ if i is true, and $f^i = 0$, $f^{\bar{i}} = 1$ otherwise. This rule obviously determines f on all edges not belonging to clause gadgets. Since we started with a valid truth assignment, each clause gadget has at most two out of the three relevant commodities i_1, i_2 and i_3 reaching u. Commodity i_j corresponds to commodity j in Example 4. If one commodity i_j, $j \in \{1, 2, 3\}$ is not present at u, then we send commodity i_{j+1} (modulo 3) along cycle $C^{i_{j+1}}$ and set all other flow values in the gadget to 0. Note that this also implies that commodity i_{j+2} (modulo 3) is forwarded by u without entering the clause gadget. If two commodities are missing, we send the third along its cycle. If no relevant commodity reaches the gadget, then we leave all edges of the gadget empty.

We need to show now that f is an integral stable flow. Feasibility and integrality clearly follow from the construction. Proceeding from s to t in the graph, we investigate at which vertex a blocking walk W might start.

1. Assume W starts at s. If $a'b'_j$ is the edge saturated by its best commodity, then W cannot proceed through $a'b'_j$. If $a'b'_j$ is not saturated by its preferred commodity, then b'_jd' is and W cannot pass through b'_jd'. Hence W either ends at a' or b'_j for some j. In either case, it ends at a non-terminal vertex with a single incoming edge. Thus a walk W starting at s cannot block f.
2. Similarly, if W starts at a', it has to end at b'_j for some j and thus W cannot block f.
3. For each j, the non-terminal vertex b'_j has a single outgoing edge. Thus it also cannot start a blocking walk.
4. The same holds for d'.
5. The same arguments apply for walks starting at a'', b''_j for some j, or d'', respectively.

6. If W starts at a vertex u_j, then its first edge must be in a clause gadget, because the edge running outside of the clause gadget is the least preferred outgoing edge of u_j.
   Assume now without loss of generality that the first edge of W is $u_j v_1$ in some clause gadget with relevant commodities i_1, i_2 and i_3, in this order. Because $u_j v_1$ only admits flow of commodity i_1, the walk W can only be blocking with respect to commodity i_1, and $f^{i_1}(e) = 1$ on the edge e leaving u_j outside the clause gadget. Thus, $u_j v_1$ is not saturated, which means that commodity i_1 was not chosen to fill C^1. According to our rules above, the only reason for this is that commodity i_2 is not present at u and commodity i_3 saturates C^3. Then the only edge that could be the second edge of W is $v_1 v_2$ in the gadget, but this edge is saturated by its best ranked commodity i_3. We conclude that a blocking walk cannot start at u_j for any j.
7. Now assume W starts at a vertex v in the interior of a clause gadget attached to u_j. Without loss of generality, let this vertex be v_1. Note that v_1 has two outgoing edges $v_1 v_2$ and $v_1 u_j$, but $v_1 v_2$ only supports flow of commodities i_1 and i_3, whereas $v_1 u_j$ only supports flow of commodity i_2. A walk starting with $v_1 u_j$ can only block f with respect to commodity i_2, but then it cannot dominate f at the start because $f^{i_2}(v_1 v_2) = 0$. Likewise, a walk starting with $v_1 v_2$ can only block f with respect to i_1 or i_3, but cannot dominate f at the start because $f^{i_1}(v_1 u_j) = f^{i_3}(v_1 u_j) = 0$.
8. No edge leaves t, so W cannot start with t.

We thus eliminated all possible starting vertices for blocking walks. Since no walk blocks the constructed flow, it is stable. □

6 Conclusion and Open Problems

In this paper we presented four results:

1. A polynomial version of the Gale–Shapley algorithm for stable flows;
2. A direct algorithm for stable flows with restricted intervals;
3. A simplification of the stable multicommodity flow problem;
4. The NP-completeness of the integral stable multicommodity flow problem.

A natural open question regarding the problem of stable flows with restricted edges presented in Sect. 4 is that of approximation. The approximation concept of minimum number of blocking edges or minimum number of violated restrictions [6] can be translated to SF RESTRICTED. Even if there is no stable flow saturating all forced edges or avoiding all forbidden edges, how can stability be relaxed such that all edge conditions are fulfilled? Or the other way round: how many edge conditions must be violated by stable flows?

The big open question of Sect. 5 is clearly algorithms for finding a (possibly fractional) stable multicommodity flow. Even though Theorem 6 states that it is PPAD-hard to find a solution in the general case, it is natural to ask whether this complexity changes when restricting the number of commodities, the maximum degree, or other parameters of the instance. Since the Gale–Shapley algorithm typically executes steps with

integer values if the input is integral and we showed the hardness of ISMF, it is likely that a novel approach is needed. Linear programming is a promising direction, but constructing a description of the SMF polytope seems to be an extremely challenging task. At the moment, the most elaborate structure for which a linear program is known is many-to-many stable matchings [12].

Finally, all stable flow models discussed in this paper can be combined with other common notions in stability or flows, such as ties in preference lists, edge weights, unsplittable flows, and so on.

Acknowledgements We thank Tamás Fleiner for discussions on Lemma 3, and our reviewers for their suggestions that significantly improved the presentation of the paper.

References

1. Baïou, M., Balinski, M.: Many-to-many matching: stable polyandrous polygamy (or polygamous polyandry). Discrete Appl. Math. **101**, 1–12 (2000)
2. Balinski, M., Sönmez, T.: A tale of two mechanisms: student placement. J. Econ. Theory **84**, 73–94 (1999)
3. Biró, P., Kern, W., Paulusma, D., Wojuteczky, P.: The stable fixtures problem with payments. Games Econ. Behav. **108**, 245–268 (2017)
4. Braun, S., Dwenger, N., Kübler, D.: Telling the truth may not pay off: an empirical study of centralized university admissions in Germany. B.E. J. Econ. Anal. Policy (2010). https://doi.org/10.2202/1935-1682.2294
5. Chen, Y., Sönmez, T.: Improving efficiency of on-campus housing: an experimental study. Am. Econ. Rev. **92**, 1669–1686 (2002)
6. Cseh, Á., Manlove, D.F.: Stable marriage and roommates problems with restricted edges: complexity and approximability. Discrete Optim. **20**, 62–89 (2016)
7. Cseh, Á., Matuschke, J., Skutella, M.: Stable flows over time. Algorithms **6**, 532–545 (2013)
8. Dean, B.C., Munshi, S.: Faster algorithms for stable allocation problems. Algorithmica **58**, 59–81 (2010)
9. Dias, V.M.F., da Fonseca, G.D., de Figueiredo, C.M.H., Szwarcfiter, J.L.: The stable marriage problem with restricted pairs. Theor. Comput. Sci. **306**, 391–405 (2003)
10. Feder, T.: A new fixed point approach for stable networks and stable marriages. J. Comput. Syst. Sci. **45**, 233–284 (1992)
11. Feder, T.: Network flow and 2-satisfiability. Algorithmica **11**, 291–319 (1994)
12. Fleiner, T.: On the stable b-matching polytope. Math. Soc. Sci. **46**, 149–158 (2003)
13. Fleiner, T.: On stable matchings and flows. Algorithms **7**, 1–14 (2014)
14. Fleiner, T., Irving, R.W., Manlove, D.F.: Efficient algorithms for generalised stable marriage and roommates problems. Theor. Comput. Sci. **381**, 162–176 (2007)
15. Fleiner, T., Jagadeesan, R., Jankó, Z., Teytelboym, A.: Trading networks with frictions. In: Proceedings of the 2018 ACM Conference on Economics and Computation. ACM, pp. 615–615 (2018)
16. Fleiner, T., Jankó, Z., Schlotter, I., Teytelboym, A.: Complexity of stability in trading networks. arXiv preprint arXiv:1805.08758 (2018)
17. Ford, L.R., Fulkerson, D.R.: Flows in Networks. Princeton University Press, Princeton (1962)
18. Gai, A.T., Lebedev, D., Mathieu, F., de Montgolfier, F., Reynier, J., Viennot, L.: Acyclic preference systems in P2P networks. In: Kermarrec, A., Bougé, L., Priol, T. (eds.) Proceedings of Euro-Par '07 (European Conference on Parallel and Distributed Computing): The 13th International Euro-Par Conference. Lecture Notes in Computer Science, vol. 4641, pp. 825–834. Springer (2007)
19. Gale, D., Shapley, L.S.: College admissions and the stability of marriage. Am. Math. Mon. **69**, 9–15 (1962)
20. Gale, D., Sotomayor, M.: Some remarks on the stable matching problem. Discrete Appl. Math. **11**, 223–232 (1985)
21. Garey, M.R., Johnson, D.S.: Computers and Intractability. Freeman, San Francisco (1979)

22. Gusfield, D., Irving, R.W.: The Stable Marriage Problem: Structure and Algorithms. MIT Press, Cambridge (1989)
23. Irving, R.W., Leather, P., Gusfield, D.: An efficient algorithm for the "optimal" stable marriage. J. ACM **34**, 532–543 (1987)
24. Jagadeesan, R.: Complementary inputs and the existence of stable outcomes in large trading networks. In: Proceedings of the 2017 ACM Conference on Economics and Computation. ACM, pp. 265–265 (2017)
25. Jewell, W.S.: Multi-commodity Network Solutions. Operations Research Center, University of California, California (1966)
26. Király, T., Pap, J.: A note on kernels and Sperner's Lemma. Discrete Appl. Math. **157**, 3327–3331 (2009)
27. Király, T., Pap, J.: Stable multicommodity flows. Algorithms **6**, 161–168 (2013). https://doi.org/10.3390/a6010161
28. Knuth, D.: Mariages Stables. Les Presses de L'Université de Montréal (1976). In: English translation in Stable Marriage and its Relation to Other Combinatorial Problems, vol. 10 of CRM Proceedings and Lecture Notes, American Mathematical Society (1997)
29. Lin, Y.S., Nguyen, T.: On variants of network flow stability. arXiv preprint arXiv:1710.03091 (2017)
30. Ostrovsky, M.: Stability in supply chain networks. Am. Econ. Rev. **98**, 897–923 (2008)
31. Papadimitriou, C.H.: On the complexity of the parity argument and other inefficient proofs of existence. J. Comput. Syst. Sci. **48**, 498–532 (1994)
32. Perach, N., Polak, J., Rothblum, U.G.: A stable matching model with an entrance criterion applied to the assignment of students to dormitories at the Technion. Int. Jo. Game Theory **36**, 519–535 (2008)
33. Roth, A.E.: The evolution of the labor market for medical interns and residents: a case study in game theory. J. Polit. Econ. **92**, 991–1016 (1984)
34. Roth, A.E., Sotomayor, M.A.O.: Two-Sided Matching: A Study in Game-Theoretic Modeling and Analysis. Econometric Society Monographs, vol. 18. Cambridge University Press, Cambridge (1990)
35. Shepherd, F.B., Vetta, A., Wilfong, G.T.: Polylogarithmic approximations for the capacitated single-sink confluent flow problem. In: 2015 IEEE 56th Annual Symposium on Foundations of Computer Science (FOCS). IEEE, pp. 748–758 (2015)
36. Tardos, É.: A strongly polynomial algorithm to solve combinatorial linear programs. Oper. Res. **34**, 250–256 (1986)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Affiliations

Ágnes Cseh[1] · Jannik Matuschke[2]

Jannik Matuschke
jannik.matuschke@tum.de

[1] Institute of Economics, Centre for Economic and Regional Studies, Hungarian Academy of Sciences, Tóth Kálmán u. 4., Budapest 1097, Hungary

[2] TUM School of Management and Department of Mathematics, Technische Universität München, Arcisstraße 21, 80333 Munich, Germany