ISSN 0965-5425, Computational Mathematics and Mathematical Physics, 2023, Vol. 63, No. 3, pp. 491–503. © Pleiades Publishing, Ltd., 2023.
Russian Text © The Author(s), 2023, published in Zhurnal Vychislitel'noi Matematiki i Matematicheskoi Fiziki, 2023, Vol. 63, No. 3, pp. 517–530.

INFORMATICS

On Stable Flows and Preflows

A. V. Karzanov[a,*]

[a]*Central Institute of Economics and Mathematics, Russian Academy of Sciences, Moscow, 117418 Russia*

**e-mail: akarzanov7@gmail.com*

Received August 11, 2022; revised August 26, 2022; accepted November 17, 2022

Abstract—We propose a new algorithm of finding a stable flow in a network with several sources and sinks. It is based on the idea of *preflows* (applied in the 1970s for a faster solution of the classical maximal flow problem) and has time complexity $O(nm)$ for a network with n vertices and m edges. The obtained results are further generalized to a larger class of objects, the so-called stable *quasi-flows* with bounded deviations from the balanced relations in nonterminal vertices.

Keywords: stable flow in a network, stable allocation, preflow, quasi-flow

DOI: 10.1134/S0965542523030077

1. INTRODUCTION

The field of theoretical and applied problems on stable contracts is a subject of an intensive study in mathematical economics, game theory and combinatorial optimization, and numerous works has been devoted to this field during last several decades. The classical work on *stable marriages* (SM) due to Gale and Shapley [1] has served as a starting point of many subsequent researches.

According to one widespread formulation of that problem, one considers a bipartite graph $G = (V, E)$, and for each vertex v, there is given a (strict) linear order $<_v$ on its incident edges. (A bipartite graph considered in [1] is assumed to be complete, but this is not so important. In a popular old interpretation the edges of G represent possible marriages between "men" and "women", and the order $<_v$ indicates the preferences of a person v, namely: for edges $e = vu$ and $e' = vw$, if $e <_v e'$, then v prefers the alliance with u to that with w.) In the problem one requires to find a matching $M \subseteq E$ which is stable with respect to all these orders. This means that for any edge e in $E - M$, there exists an edge $e' \in M$ such that e and e' share a vertex v and there holds $e' <_v e$. It was shown that a stable matching in a bipartite graph (equipped with linear orders on the vertices) always exists and that it can be constructed by a combinatorial algorithm having a linear upper bound on the number of operations (viz. *running time*, *complexity*) $O(n + m)$, where n and m are the numbers of vertices and edges in G, respectively.

Subsequent works of many authors have explored various generalizations of problem SM. One can distinguish two ways of generalizations related to graphs (leaving aside settings where more than two "agents" may be involved in a contract, or where the preferences could be given in a different manner, e.g. by use of a choice function). One of them is to admit arbitrary, not necessarily bipartite, graphs G. The corresponding analog of problem SM, called the *stable roommates problem*, was well studied by Irving in [2] where a linear complexity algorithm was devised that either constructs a stable matching in G or proves that there is none. Important additional structural and algorithmic results were presented in [3].

Another type of generalizations, which is more interesting for us here, remains the graph $G = (V, E)$ to be bipartite but adds numerical parameters. Among the problems of this type, a quite general one is the so-called *stable allocation problem* (SA) introduced and studied by Baiou and Balinski [4]. Here "allocation" means an assignment to each edge $e \in E$ a value $x(e) \geq 0$ not exceeding a prescribed capacity $c(e)$, and at the same time, for each vertex $v \in V$, the sum of assignments on the edges incident to v should not exceed a prescribed "quota" $q(v)$. (In case of SM, all $c(e)$ and $q(v)$ are ones, and each $x(e)$ takes value 0 or 1. In a general case of SA, the number $x(e)$ on an edge $e = uv$ can be meant, for example, as the share of participation of the "worker" u in the "job" v.)

In [4] it is proved that SA is solvable for any nonnegative real-valued c, q (and has an integral solution x when c, q are integer-valued), and a strongly polynomial solution algorithm is proposed (i.e., an algo-

rithm whose running time depends only on the graph size and is expressed by a polynomial in n, m). Dean and Munshi [5] developed an improved version of the algorithm from [4] that constructs a solution in $O(nm)$ time; moreover, they showed that one can achieve even the better time bound $O(m\log n)$ if one applies, for a number of procedures, certain powerful data structures, such us dynamic and self-adjusting trees elaborated by Sleator and Tarjan [6, 7]. (This gives a theoretical acceleration, but an algorithm using such structures are cumbersome and could hardly be applied for practical purposes.)

In its turn, problem SA can be represented as a special case of the *stable flow problem* (SF). The latter was stated by Fleiner [8] in 2010th (as an extension of Ostrovsky's problem [9] concerning acyclic networks with unit capacities). Here one is given a network consisting of a directed graph $G = (V, E)$ with capacities $c(e) \geq 0$ of edges $e \in E$ and two distinguished vertices ("terminals") s and t. For each *inner* vertex $v \in V - \{s,t\}$, two linear orders are given, one on the set of its entering (incoming) edges, and the other on the set of its leaving (outgoing) edges. A (feasible) *flow* is a nonnegative real function f on the edges satisfying the upper capacity constraints and having zero excesses at all inner vertices, where by an excess at a vertex v one means the difference $\mathrm{ex}_f(v)$ between the total flow on the entering edges and that on the leaving edges at v. (In applications an inner vertex can be interpreted as a "player" or "trader" or "agent" who after having received some amount of goods through the entering edges, is sending this amount further through the leaving ones, relying on his/her utility rule depending on the above mentioned orders.) A flow is regarded as *stable* if, roughly speaking, it does not admit local improvements using "unsaturated paths"; a precise definition will be given in Section 2.

There is a close relation between problems SA and SF. Indeed, problem SA with a bipartite graph $G = (V_1 \sqcup V_2, E)$ is reduced to problem SF with the graph obtained from G (whose edges are directed from V_1 to V_2) by adding two terminals s and t, the edges (s,u) with the capacities $q(u)$ for vertices u from the part V_1, and the edges (v,t) with the capacities $q(v)$ for vertices v from the part V_2.

On the other hand, Fleiner [8] showed that problem SF with a graph $G = (V, E)$ can in turn be reduced to problem SA with some graph obtained by splitting the vertices of G into pairs and adding $O(|V|)$ new edges. As a result, for any network, the existence of a stable flow was established (and an integral one when c is integer-valued), and a possibility of constructing such a flow by use of algorithms for SA, yielding complexity of a similar kind.

Subsequently there appeared direct algorithms of finding a stable flow (without appealing to SA). Recently Cseh and Matuschke [10] proposed a direct algorithm for a network with one source and one sink, which has complexity $O(nm)$ and is based on a combination of ideas of Ford–Fulkerson's method for the maximal flow problem and the "deferred acceptance method" originated in Gale and Shapley [1].

In this work we present an alternative algorithm to find a stable flow in a network $N = (G, S, T, c)$ with arbitrary sets of sources S and sinks T, assuming (for simplicity) that there are no edges entering sources or leaving sinks. This algorithm is direct and purely combinatorial (neither appealing to SA nor using sophisticated data structures); it exploits the idea of preflows. Recall that a *preflow* in a network is a nonnegative function on the edges that is bounded by the capacities and has nonnegative excesses at all inner vertices. (This notion was introduced in [11] and used in the max flow algorithm developed there.) Note that preflows were used earlier in [12] to find a stable flow in a network with integral capacities in a pseudo-polynomial time (i.e., with a time bound depending linearly on the sum of capacities).

Our algorithm has basic and modified (accelerated) versions. Both start with constructing a certain initial stable preflow, and on each subsequent iteration, the current stable preflow is updated with the aim of getting rid of positive excesses in inner vertices. As soon as the excesses of all inner vertices become zero, a required stable flow is constructed (and the flow is integral when the capacities c are integer-valued). The basic algorithm is finite for any nonnegative real capacities c. The modified algorithm is strongly polynomial; it applies additional transformations and constructs a stable flow in $O(nm)$ time (like the algorithm in [10]).

Then we consider a more general problem in a network $N = (G, S, T, c)$ in which for each inner vertex v, there are given two parameters $\beta(v) \geq 0$ and $\gamma(v) \geq 0$ and it is required to find a stable "quasi-flow" f satisfying the constrains of the form $-\beta(v) \leq \mathrm{ex}_f(v) \leq \gamma(v)$. (This turns into a stable flow when $\beta, \gamma = 0$.) We show that such a "quasi-flow" does exist and it also can be found in $O(nm)$ time. (In applications the number $\gamma(v)$ can be thought of as the permission to the "agent" v to manage on his own a part of the received goods not exceeding $\gamma(v)$, while the number $\beta(v)$ as the permission to involve from "outside" additional goods of the amount not exceeding $\beta(v)$.)

This paper is organized as follows. Section 2 contains basic definitions and exact formulations of problems SF and SA. In Section 3 we describe the basic algorithm of finding a stable flow in a network by use of preflows and show its finite convergence (Proposition 1). The modified version of the algorithm, having complexity $O(nm)$, is described in Section 4. Section 5 generalizes the obtained results to stable preflows and quasi-flows with restricted excesses (Propositions 3 and 4). In the final Section 6 we discuss three additional properties, namely: (1) the maximum preflow value (i.e. the total excess in the sinks) among the stable preflows is achieved by a stable flow; (2) all stable flows in a fixed network have the same value; and (3) the stable flows form a lattice. (Properties 2 and 3 are shown in [8] via a reduction to the corresponding properties for problem SA; we give simple proofs using direct constructions.)

2. DEFINITIONS AND SETTINGS

We consider a network $N = (G, S, T, c)$ consisting of a *directed* graph $G = (V, E)$ (without loops and multiple edges), distinguished disjoint subsets of vertices S (*sources*) and T (*sinks*), also called *terminals*, and a function $c : E \to \mathbb{R}_+$ of *capacities* of edges. (Hereinafter $\mathbb{R}_+$ and $\mathbb{Z}_+$ are the sets of nonnegative reals and integers, respectively.) For a vertex $v \in V$, let $\delta^{\text{in}}(v)$ and $\delta^{\text{out}}(v)$ denote the sets of edges *entering* v and *leaving* v, respectively. To simplify our description, we throughout assume that

$$\delta^{\text{in}}(s) = \varnothing \quad \forall s \in S \quad \text{and} \quad \delta^{\text{out}}(t) = \varnothing \quad \forall t \in T. \tag{2.1}$$

Definition 1. A function $f : E \to \mathbb{R}_+$ is called *feasible* (with respect to c) if it satisfies the capacity constraints $f(e) \le c(e)$ for all edges $e \in E$. Define the *excess* of f at a vertex $v \in V$ by

$$\text{ex}_f(v) := \sum_{e \in \delta^{\text{in}}(v)} f(e) - \sum_{e \in \delta^{\text{out}}(v)} f(e).$$

A feasible function f is called a *preflow* in N (following terminology from [11]) if it has a nonnegative excess $\text{ex}_f(v) \ge 0$ at each vertex $v \in V - S$. A *flow* (from S to T) is a preflow f having zero excesses at all *inner* (nonterminal) vertices $v \in V - (S \cup T)$, and its *value* $\text{val}(f)$ is defined to be $\text{ex}_f(T) := \sum_{t \in T} \text{ex}_f(t)$.

A *path* in G is a sequence $P = (v_0, e_1, v_1, \ldots, e_k, v_k)$, where e_i is an edge connecting vertices v_{i-1} and v_i. An edge e_i in P is called *forward* if $e_i = v_{i-1}v_i$, and *backward* if $e_i = v_i v_{i-1}$. (We usually denote an edge leaving u and entering v as uv, rather than (u, v).) A path is called *directed* if all its edges are forward, and *simple* if all its vertices are different. The reversed path $(v_k, e_k, v_{k-1}, \ldots, e_1, v_0)$ is denoted as P^{-1}. A path from a vertex u to a vertex v may be called a *u–v path*. Speaking of a path, unless otherwise is explicitly said, we assume that it is nontrivial, i.e., it has at least one edge.

For a feasible function f, an edge e with $f(e) = c(e)$ ($f(e) < c(e)$; $f(e) = 0$) is called *saturated* (respectively, *unsaturated*; *free* (of f)). A path is regarded as unsaturated if all its edges are such.

In the problem that we consider each inner vertex v of a network N is endowed with a linear (total strict) order $<_v^-$ on the set $\delta^{\text{in}}(v)$, and a linear order $<_v^+$ on the set $\delta^{\text{out}}(v)$. They are interpreted as "preference relations"; namely, $e <_v^- e'$ means that the vertex ("agent") v prefers the edge e to e'; in this case, we will also say that e is located in $\delta^{\text{in}}(v)$ before, or earlier than, or on the left from e', and accordingly, e' is after, or later than, or on the right from e. In accordance with the order $<_v^-$, the set $\delta^{\text{in}}(v)$ is arranged as a double-linked list; thus, the first (last) element in the list is most (respectively, least) preferable. And similarly for the order $<_v^+$.

Definition 2. A flow f in a network N, with order equipments on the vertices as mentioned above, is called *stable* if each unsaturated *directed* path $P = (v_0, e_1, v_1, \ldots, e_k, v_k)$ satisfies at least one of the following two conditions (where the case $v_0 = v_k$ is allowed):

$$\begin{aligned} &\text{the first vertex } v_0 \text{ is inner and } P \text{ is dominated at } v_0; \\ &\text{this means that each edge } e \in \delta^{\text{out}}(v_0) \text{ later than } e_1 \text{ is free of } f; \end{aligned} \tag{2.2}$$

$$\begin{aligned} &\text{the last vertex } v_k \text{ is inner and } P \text{ is dominated at } v_k; \\ &\text{that is, each edge } e \in \delta^{\text{in}}(v_k) \text{ later than } e_k \text{ is free of } f. \end{aligned} \tag{2.3}$$

In particular, there is no unsaturated directed path from S to T. A preflow f is called stable in the analogous case.

We are interested in the *stable flow problem* (SF): given a network N, find a stable flow in it. Fleiner [8] established that this problem is always solvable, and furthermore, the property of integrality takes place: if a network $N = (G = (V, E), S, T, c)$ is integral (in the sense that $c \in \mathbb{Z}_+^E$), then there exists an integral stable flow.

This is proved in [8] by a reduction of problem SF in N to the *stable allocation problem* (SA) in a bipartite graph $G' = (V', E')$ with weights $c'(e) \geq 0$ on the edges $e \in E'$ and "quotas" $q(v) \geq 0$ on the vertices $v \in V'$. (Strictly speaking, [8] deals with the case when $|S \cup T| = 2$ and condition (2.1) is not imposed; however, the above-mentioned facts are valid for our case as well.) Also the reduction is linear in the input graph size, namely: $|V'| = 2|V|$ and $|E'| < |E| + 2|V|$, and preserves the integrality: c' and q are integral when so is c. Problem SA was introduced and studied in [4] where the solvability for any bipartite network and the integrality property were proved and a solution algorithm was devised.

As is mentioned in the Introduction, it is shown in [5] that by use of sophisticated data structures, the so-called dynamic and self-adjusting trees, the stable allocation problem in a graph with n vertices and m edges can be solved in $O(m \log n)$ time (but without using those, one can obtain $O(nm)$ time). This, due to the above-mentioned reduction, leads to a similar algorithmic complexity for problem SF as well. A direct algorithm for SF in case $|S| = |T| = 1$ proposed in [10] has complexity $O(nm)$.

In the present paper we device an alternative direct algorithm of finding a stable flow in a network $N = (G, S, T, c)$ (subject to (2.1)) based on a preflow method. In the next section we describe the basic version of our algorithm (finite for any c), and in Section 4, the modified version (with running time $O(nm)$).

3. BASIC ALGORITHM

Our algorithm of finding a stable flow in a network $N = (G = (V, E), S, T, c)$ consists of a sequence of iterations; as a rule (but not always) an iteration has two *phases*: *balancing* and *pushing* (similar to the structure of a stage ("big iteration") in the max flow algorithm in [11]). Each iteration transforms one blocking preflow into another, and the process terminates when the current preflow becomes a flow. One should emphasize that the term "blocking" here is borrowed from the language in [11] and has another meaning compared with what is usually understood in problems on stability.

Definition 3. For a preflow f in a network N, let us say that a vertex v is *excessive* if $\mathrm{ex}_f(v) > 0$. A preflow f is called *blocking* if every directed path from S to T has a saturated edge. If, in addition, any directed path going from an excessive inner vertex to T has a saturated edge, then we call f *fully blocking*.

Remark 1. A fully blocking preflow is not necessarily stable, and vice versa. At the same time, any stable flow is blocking (and automatically fully blocking). In general, already in the case of ranged acyclic networks, to find a stable flow looks a somewhat more complicated task than the problem of finding a blocking flow. The latter problem is solved on a stage of the algorithm in [11] in $O(n^2)$ time, which is faster than the complexity $O(nm)$ of the algorithm for SF in Section 4.

3.1. Initial Iteration

At this and subsequent iterations, we maintain two sets Old and New which are arranged as double-linked lists and formed by excessive inner vertices of the current preflow. Also for each inner vertex v, there is an element $\tilde{e}(v)$ which either is void, denoted as $\tilde{e}(v) = \{\varnothing\}$, or is a specified unsaturated edge in $\delta^{\mathrm{out}}(v)$, called the *active* edge at v. In the beginning one puts Old := New := $\varnothing$, and for each $v \in V - (S \cup T)$, one assigns as the active edge the first (most preferable) edge in $\delta^{\mathrm{out}}(v)$.

The initial iteration consists of only the pushing phase, which starts with the trivial blocking preflow f, defined as $f(e) := c(e)$ for all edges $e \in \delta^{\mathrm{out}}(s)$, $s \in S$, and $f(e) := 0$ for the remaining edges e. Each inner vertex v connected by an edge $e = sv$ with a source $s \in S$ is inserted in the list New (since the excess at v became positive). Then we scan vertices of the current list New, as follows.

For a current vertex $v \in \mathrm{New}$, we update f on $\delta^{\mathrm{out}}(v)$ so as either to reduce the excess at v to zero, or to saturate all edges in $\delta^{\mathrm{out}}(v)$ (or both). More precisely, the edges $e \in \delta^{\mathrm{out}}(v)$ are handled following the order $<_v^+$ ("from left to right"), starting from the active edge $e = \tilde{e}(v)$. If the current excess $\Delta := \mathrm{ex}_f(v)$ is still nonzero, then we update $f(e) := \min\{c(e), f(e) + \Delta\}$. If the new excess becomes zero, then the treatment of v finishes; otherwise we proceed to the next edge in the list $\delta^{\mathrm{out}}(v)$, and so on. Also when changing f on the edge $e = vw$, if the vertex w (in which the excess increases) is contained neither in Old nor in New, then we add w to the current list New.

Upon termination of the work with v, this vertex is deleted (maybe temporarily) from the list New, and if the excess at v remains nonzero, then v is inserted in the list Old. The new active edge $\tilde{e}(v)$ is assigned to be the leftmost unsaturated edge in $\delta^{\mathrm{out}}(v)$, and if all edges in it are saturated (in particular, when $\mathrm{ex}_f(v) > 0$), then one puts $\tilde{e}(v) := \{\varnothing\}$. Then we proceed to handling another vertex in New, and so on. The iteration terminates when the set New becomes empty.

Note that during the iteration one and the same vertex v may appear and disappear several times in New. However, we shall show later that the initial (and every subsequent) iteration always terminates. One can see the following:

> Upon termination of the initial iteration, the resulting function f is a fully blocking preflow f possessing the property that: for each inner vertex v, if $\tilde{e}(v) = \{\varnothing\}$, then all edges in $\delta^{\mathrm{out}}(v)$ are saturated, whereas if $\tilde{e}(v) \neq \{\varnothing\}$, then all edges in $\delta^{\mathrm{out}}(v)$ earlier than e are saturated, but all edges after e are free of f. Also all excessive inner vertices v (and only these) are included in the list Old, and for such a v, we have $\tilde{e}(v) = \{\varnothing\}$. (3.1)

From (3.1) it is easy to conclude validity of (2.2) for all unsaturated directed paths; therefore, the initial preflow f is stable.

In the rest of this section we first describe the balancing phase for a general iteration. Then we specify the conditions to which a current preflow should satisfy at the moment of beginning the pushing phase. Finally, we describe the pushing phase for a general iteration.

3.2. Balancing

This procedure is performed when the set Old of excessive inner vertices for the current preflow f is nonempty (while $\mathrm{New} = \varnothing$). It applies to one chosen vertex $v \in \mathrm{Old}$. Define $\Delta := \mathrm{ex}_f(v)$ (>0) and let e be the last (rightmost, least preferable) edge in $\delta^{\mathrm{in}}(v)$ with $f(e) > 0$. We decrease f on e by $\min\{\Delta, f(e)\}$. If the excess at v becomes zero (in case $\Delta \le f(e)$), the procedure finishes. Otherwise (when $\Delta > f(e)$) we take the last edge e' with $f(e') > 0$ for the updated f and decrease f on e' in a similar way. And so on until the excess at v becomes zero.

For each edge $e = uv$ where f decreases under the balancing at v, we examine the vertex u. If u is inner and not contained in Old, then we add it to the list New. (Thus, New is formed by the set of new excessive vertices u appeared as a result of decreasing f on edges uv.) The leftmost (chronologically last) edge in $\delta^{\mathrm{in}}(v)$ where f was decreased is called *critical* and denoted by $\hat{e} = \hat{e}(v)$. This edge along with all edges $e = uv$ in $\delta^{\mathrm{in}}(v)$ after it are labeled as *closed* (to avoid any further increase only!). At the same time, if such an edge e is the current active edge in $\delta^{\mathrm{out}}(u)$, then we assign the new active edge in it to be the first after e edge in $\delta^{\mathrm{out}}(u)$ that is not closed (under earlier balancing phases), and if it is absent, then one puts $\tilde{e}(u) := \{\varnothing\}$.

Remark 2. During the algorithm one should maintain the values of excesses at the inner vertices (correcting each of them in $O(1)$ time whenever f changes).

We assume (by induction) that upon termination of the balancing phase at a vertex v, the current function f is a preflow, which need not be fully blocking but satisfies the following three properties.

(C1) Each unsaturated edge in $\delta^{\mathrm{out}}(s)$, $s \in S$, is closed.

(C2) If for an excessive inner vertex u, the set $\delta^{out}(u)$ contains an unsaturated edge which is not closed, then $u \in$ New.

(C3) For each inner vertex u, there holds: (a) if the active edge in $\delta^{out}(u)$ is not void: $\tilde{e}(u) \neq \{\varnothing\}$, then this edge is unsaturated and not closed, and all edges in $\delta^{out}(u)$ after it are free of f, while each edge before it is saturated or closed (becoming so after the last or earlier balancing); (b) if $\tilde{e}(u) = \{\varnothing\}$, then all edges in $\delta^{out}(u)$ are saturated or closed; and (c) if u has ever been balanced, then $\tilde{e}(u) = \{\varnothing\}$, and the set $\delta^{in}(u)$ possesses the critical edge $\hat{e}(u)$, which is unsaturated and all edges in $\delta^{in}(u)$ after it are free of f.

3.3. Pushing (after Balancing at a Vertex v)

This phase consists in increasing the current preflow f on some edges, aiming to make it fully blocking; this is close to the construction of the initial preflow but has some features. The phase immediately terminates when the beginning set New is empty. Otherwise it starts with choosing a vertex $u \in$ New, and we try to reduce the excess at u as much as possible. To this aim, we handle edges in $\delta^{out}(u)$ following the order $<_u^+$ and skipping the closed ones. This starts with the active edge $\tilde{e}(u) \neq \{\varnothing\}$ (if $\tilde{e}(u) = \{\varnothing\}$, then we simply transfer the vertex u from New to Old and the work with u finishes). Similar to the initial iteration, for the current edge $e = uw$, we update $f(e) := \min\{c(e), f(e) + \Delta\}$, where Δ is the current excess at u, and simultaneously add the vertex w to the set New unless it is already contained in Old $\cup$ New (for the excess at w increases). If the new excess at u is still nonzero, then we proceed to the next non-closed edge in $\delta^{out}(u)$, and so on. If the excess becomes zero, then the work with u finishes and we delete u from New. When all edges are already handled but the excess at u remains nonzero, then u is transferred from New to Old. After finishing the work with u and properly updating the active edge $\tilde{e}(u)$ in $\delta^{out}(u)$ (getting $\tilde{e}(u)$ which satisfies (C3a, C3b), we choose another vertex in the current New, and so on. The phase terminates when the current set New becomes empty.

Upon termination of the pushing phase (which is finite by Prop. 1 below) the resulting f is again a preflow, and one can see that f continues to satisfy properties (C1) and (C3), whereas (C2) is replaced by the following property (cf. C3b).

(C2') Each excessive inner vertex u is contained in Old, and $\tilde{e}(u) = \{\varnothing\}$ takes place.

Lemma 1. *The preflow f obtained in the pushing phase is stable and fully blocking.*

Proof. Consider an unsaturated directed path $P = (u_0, e_1, u_1, \ldots, e_k, u_k)$. Suppose that either $u_0 \in S$ or P is not dominated at u_0 (when u_0 is inner). Then the edge e_1 is closed (in view of (C1) and (C3)). This implies that the vertex u_1 was handled at some balancing phase, so it was excessive before that. Applying (C2') and (C3b) to the vertex u_1 and preflow f' at the moment just before the balancing, we conclude that the edge e_2 should be closed to this moment. Therefore, e_2 is closed for f as well. Arguing so, step by step, we can conclude that the edges $e_3, \ldots, e_k$ are closed as well. (The case $u_k \in T$ is impossible since at the moment of balancing at u_{k-1}, this vertex was excessive, and the situation that e_k is unsaturated is impossible.) Since e_k is closed, it must either occur in $\delta^{in}(u_k)$ after the critical edge $\hat{e}(u_k)$ or coincide with the latter. Then it follows from (C3c) that all edges in $\delta^{in}(u_k)$ after e_k are free of f. Therefore, P is dominated at u_k, yielding (2.3). This implies that f is stable.

The fact that f is fully blocking is deduced from (C1), (C3), (C2') by similar reasonings. Q.E.D.

Using the above-mentioned properties of a preflow obtained at the pushing phase, one can conclude that the balancing phase of the next iteration produces a preflow satisfying properties (C1)–(C3). This implies the correctness of the basic algorithm.

3.4. Convergence of the Basic Algorithm

From Lemma 1 we obtain

Corollary 1. If for the current preflow f at an iteration, the excesses of all inner vertices become zero, then f is a stable flow.

In this case f is a required solution of the problem and the algorithm finishes. Since each application of balancing or pushing can either decrease or increase the number of excessive inner vertices, in order to establish the finiteness of the algorithm we need an additional analysis.

For a current f, let us call an edge e *middle* if it is neither free nor saturated: $0 < f(e) < c(e)$. Let $\Gamma = (V_\Gamma, E_\Gamma)$ denote the subgraph of G induced by the middle edges. In particular, Γ contains all non-free critical edges. We also add to Γ each *free* active edge (recall that any active edge is non-closed and unsaturated). We can observe the following.

(1) For an edge $e = uv$, let us call the moment when e becomes saturated the *event S*, and the moment when e becomes free (after reducing a positive value of f to zero) the *event F*. The changes at e are of "single-peak character": at a first stage $f(e)$ is monotone increasing under pushing at u, and after the first decrease of $f(e)$ the edge e becomes closed and further $f(e)$ can only decrease (under balancing at v).

(2) An edge e can be added to the graph Γ at most two times: first when e becomes active, and second when e becomes critical.

Let α_S, α_F, α_M denote, respectively, the numbers of events S, events F, and events M consisting of changes of the graph Γ. From the observations above it follows that

$$\text{each of the numbers } \alpha_S, \alpha_F, \alpha_M \text{ is estimated as } O(m). \tag{3.2}$$

Thus, to analyze the convergence of the algorithm, one should estimate the number of consecutive iterations when none of the events S, F and M happens. To this aim, we notice the following.

(3) During the algorithm, for each inner vertex v, the active edge in $\delta^{\text{out}}(v)$ can be shifted only to the right (when pushing at v), whereas the critical edge in $\delta^{\text{in}}(v)$ only to the left (when balancing at v); also the status of such edges changes: for the old active edge, the event S happened or the edge became closed, and for the old critical edge, the event F did. In view of this, denoting by $E^+ = E^+(f)$ and $E^- = E^-(f)$ the sets of active and critical edges in Γ, respectively, we obtain that

$$\begin{gathered}\text{as a result of a balancing or pushing operation, the graph } \Gamma \\ \text{preserves if and only if none of the sets } E^+ \text{ and } E^- \text{ changes.}\end{gathered} \tag{3.3}$$

One can also see that these sets give a partition of E_Γ:

$$E^+ \cup E^- = E_\Gamma \quad \text{and} \quad E^+ \cap E^- = \varnothing$$

(taking into account (C3) and the fact that a critical edge $e = uv$ becomes closed and could no longer be active in $\delta^{\text{out}}(u)$). Therefore, in case of preserving Γ together with preserving the same partition (E^+, E^-) on consecutive iterations, any change of the preflow f on an iteration consists in only one decrease at some critical edge, or in one or more increases at some active edges (which preserve). Note also that an active edge can be turned into a critical one, but not conversely. These facts together with properties (3.2) and (3.3) allow us to obtain the following

Proposition 1. *The above basic algorithm of finding a stable flow in a network $N = (G, S, T, c)$ is finite, and in case of integer-valued capacities c, it finds an integral stable flow.*

Proof. One should show the finiteness of a sequence of consecutive iterations on which both E^+ and E^- preserve. Let ε be the minimal positive excess among inner vertices in the beginning of this sequence. Assume by induction that before a change of the preflow f on an iteration of this sequence, the set $\text{Old} \cup \text{New}$ of excessive vertices is nonempty and ($*$): each of them has the excess at least ε. If at this moment the balancing operation at a vertex v takes place, then, in view of keeping E^-, this operation is reduced only to decreasing f by the value $\Delta := \text{ex}_f(v) \geq \varepsilon$ at the critical edge $\hat{e}(v) = uv$. This turns the excess at v to zero and causes growing the excess at the vertex u by the same amount Δ; therefore, property ($*$) is valid for the updated f (in case $u \in S$ the set $\text{Old} \cup \text{New}$ simply decreases by one element v). And in case of performing the pushing operation at some inner vertex u with the active edge $\tilde{e}(u) \neq \varnothing$, then, in view of keeping E^+, this operation is reduced to increasing f on $\tilde{e}(u) = uw$ by the value $\Delta := \text{ex}(u) \geq \varepsilon$. This turns the excess at u to zero and increases the excess at the vertex w by Δ; therefore, property ($*$) continues to hold for the new f (in case $w \in T$ the set $\text{Old} \cup \text{New}$ decreases by one element u).

Thus, any change of f consists in decreasing it by $\geq\varepsilon$ at one edge in E^- or increasing by $\geq\varepsilon$ at one edge in E^+. Hence the number of iterations in the given sequence does not exceed $c(E^- \cup E^+)/\varepsilon$, implying the finiteness of the algorithm.

In case of integral c any operation changes the preflow value at an edge by an integer amount, and therefore, the resulting stable flow is integral as well. Q.E.D.

In spite of the finiteness of the basic algorithm, the number of iterations in it can be very large, as the next example shows.

Example. Let G contain vertices u, v, w and edges uv, vw and uw such that $uv <_u^+ uw$ and $uw <_w^- vw$. Assume that the edge vw is critical in $\delta^{\text{in}}(w)$, the edge uv is critical in $\delta^{\text{in}}(v)$, and the edge uw is active in $\delta^{\text{out}}(u)$. We assume that the values $f(uv)$, $f(vw)$ and $c(uw) - f(uw)$ are sufficiently large, that the excess Δ at w is positive and sufficiently small, and that the excesses at u and v are zero. The work of the algorithm can happen as follows: first $f(vw)$ decreases by Δ, second $f(uv)$ decreases by Δ, third $f(uw)$ increases by Δ. So we come to the starting vertex w with the same excess Δ (and with $\text{ex}(u) = \text{ex}(v) = 0$), and then we move along the same cycle $w \to v \to u \to w$ again, and so many times.

In the next section we explain how to modify the above algorithm in order that it could lead to a faster solution.

4. MODIFIED ALGORITHM

In the above algorithm the number of consecutive iterations on which the auxiliary graph Γ is not changed can be very large (as Example in Section 3 shows); for convenience hereinafter we include the partition (E^+, E^-) in the description of Γ. In this section we describe a modified version for which the number of changes of the current preflow f without changing $\Gamma = (V_\Gamma; E^+, E^-)$ is of order $O(n)$. This is equivalent to the fact that $O(n)$ changes of f lead to the event S, F, or M (including the case when some active edge is getting closed and critical).

Let us call a path in Γ *regular* if all active edges in it are forward while all critical edges are backward. A maximal sequence of iterations with a fixed Γ will be called a *big iteration*. It starts with choosing an excessive vertex v_0 in (G, f), and the order of handling the vertices in it is now refined as follows (assuming that v_0 belongs to Γ).

(A) When balancing at an excessive vertex v, which consists in decreasing f at the critical edge uv (which keeps in Γ), if we see that the vertex u is inner and its active edge is void (whence all edges in $\delta^{\text{out}}(u)$ are saturated or closed (see (C3b), implying that pushing at u is impossible), then we proceed to balancing at the vertex u.

(B) When pushing at a vertex u, which consists in increasing f at the active edge $e = uw$, if we see that the vertex w is inner and has a non-void active edge wz, then we proceed to the pushing operation at w; whereas if $\tilde{e}(w) = \{\varnothing\}$, then we go to balancing at w. At this balancing, if the critical edge in $\delta^{\text{in}}(w)$ turns out to be the same edge $e = uw$, then this edge becomes closed and no longer active at u; this implies that the graph Γ changes and the big iteration finishes.

From these refinements it follows that the sequence of handled vertices and edges forms a regular path $P = (v_0, e_1, v_1, \ldots, e_i, v_i, \ldots)$. For a current vertex v_k in P, the following especial situations are possible:

(Q1) v_k coincides with an earlier vertex v_i;

(Q2) v_k is a terminal.

Consider these situations in detail. Note that as a result of updating f on $e_1, \ldots, e_k$, the excesses of all vertices in P, except for v_k, become zero.

(1) In case (Q1), we extract the simple regular cycle $C = (v_i, e_{i+1}, \ldots, v_k)$. Similar to what used to be done with rotations in known algorithms for stable b-matchings, allocations, etc., we uniformly update f along C by "pushing" the amount Δ equal to the minimum of values $f(e)$ for $e \in E_C \cap E^-$ and values $c(e') - f(e')$ for $e' \in E_C \cap E^+$, that is, by increasing f by Δ in the forward edges and decreasing by Δ in the backward edges of the cycle C. As a result, at least one edge in C becomes saturated or free, implying

that event M (along with S or F) happens, completing the big iteration. Also the excesses of all vertices preserve and one can see that the updated preflow f continues to be stable and fully blocking.

(2) In case (Q2), we store P (where now the excesses of all inner vertices are zero) and, in the assumption of preserving Γ, we continue the big iteration, by choosing a new excessive vertex v_0' in (G, f) and constructing a new regular path $P' = (v_0', e_1', v_1', \ldots)$. Consider possible variants.

(a) If the path P' generates a cycle, than we act as described in part 1 above and complete the big iteration.

(b) If P' meets the previous path P (going to a terminal) in a vertex $v_r' = v_i$, then we combine P' with P, obtaining a tree $\mathcal{T}$ (having its root at some source $s \in S$ or "anti-root" at some sink $t \in T$); here all inner vertices occurring in $\mathcal{T}$, except for v_r', have zero excess. We continue the big iteration with a new excessive vertex v_0'' in (G, f).

(c) If P' enters a terminal different from the terminal in P, then we store P' (as well as P) and continue with a new excessive vertex.

(d) In a general case, every new constructed path, unless it contains a cycle or enters a new terminal, meets either a tree $\mathcal{T}$ rooted in S or a tree $\mathcal{T}'$ "anti-rooted" in T, and we combine this path with the given tree.

As a result, when Γ preserves (in particular, when no cycle arises), we obtain the following situation: in Γ there are constructed several pairwise disjoint regular trees $\mathcal{T}_1, \ldots, \mathcal{T}_k$ with the roots in S and trees $\mathcal{T}_1', \ldots, \mathcal{T}_\ell'$ with the anti-roots in T, and all inner vertices of G not contained in these trees have zero excesses.

Note also that the number of updates of the preflow f is equal to $|E_{\mathcal{T}_1}| + \cdots + |E_{\mathcal{T}_k}| + |E_{\mathcal{T}_1'}| + \cdots + |E_{\mathcal{T}_\ell'}| \simeq O(n)$. It remains to explain how to get rid of these trees.

Suppose that $\ell \neq 0$ and let Z be the set of excessive vertices in $\mathcal{T}_1'$ (which is exactly the set of branching points in $\mathcal{T}_1'$). Scanning $\mathcal{T}_1'$, we extract in it the minimal subtree W containing Z and the anti-root $t \in T$ of $\mathcal{T}_1'$, and enumerate the edges of W in a topological order, say, $w_1, \ldots, w_p$ (so that if w_j lies on the path connecting w_i and t, then $i < j$). Scanning the edges of W in this order, we update f in a natural way, obtaining one of the following two situations: either (i) the excesses of all vertices in $\mathcal{T}_1' - \{t\}$ become zero, or (ii) some intermediate edge becomes saturated or free, thus completing the big iteration.

We act with the other trees $\mathcal{T}_i'$ and $\mathcal{T}_j$ in a similar way.

Thus, the total work with the trees takes $O(n)$ time and finishes either with event M (along with S or F) or with elimination of all excessive inner vertices in G, and therefore, obtaining a stable flow f. During the big iteration, each edge in Γ is handled $O(1)$ times. Therefore, the complexity of the big iteration is $O(n)$. This together with (3.2) leads to the following result.

Proposition 2. *The modified algorithm finds a stable flow f in a network $N = (G = (V, E), S, T, c)$ in $O(nm)$ time (and f is integral when so is c).*

Remark 3. With a sufficient certainty one can think that the above algorithm can be accelerated to attain the time bound $O(m \log n)$, by using data structures as in [6, 7] to make operations on the current graphs Γ (such as extracting and rearranging components and cycles in Γ, updating the preflow on cycles and trees, etc.), in a similar spirit as analogous procedures on the subgraph of "weak edges" are accelerated in the algorithm for the allocation problem in [5]. We omit elaboration of such technical details in this paper, trying to keep a sufficient simplicity and possible practical applications of the above method.

5. GENERALIZATIONS

The stable flow problem in a network $N = (G = (V, E), S, T, c)$ can be generalized by introducing for each inner vertex v, an upper bound $\gamma(v) \in \mathbb{R}_+$ on the excess allowed at v. (As before, we impose condition (2.1).)

Definition 4. A preflow $f : E \to \mathbb{R}_+$ in N is called a γ-*preflow* if it satisfies the restrictions

$$0 \leq \mathrm{ex}_f(v) \leq \gamma(v) \quad \forall v \in V - (S \cup T). \tag{5.1}$$

In practical interpretations one can think of restrictions in (5.1) as a permission for each "agent" v do not send further all the goods delivered through the entering edges in $e \in \delta^{\text{in}}(v)$, but store (or send "aside") a part of the goods not exceeding $\gamma(v)$.

We call a γ-preflow *stable* if for each unsaturated directed u–v path P, at least one of the following is valid: (a) the vertex v is inner and P satisfies (2.3) (with $v_k = v$), i.e., P is dominated at v; or (b) the vertex u is inner and P is *strongly dominated* at u, which means validity of (2.2) (with $v_0 = u$) and the absence of excess at u: $\text{ex}_f(u) = 0$. When $\gamma = 0$, this turns into the definition of a stable flow.

A generalization of above results to γ-preflows is viewed as follows.

Proposition 3. *For a network $N = (G = (V, E), S, T, c)$ and a vector $\gamma \in \mathbb{R}_+^{V-(S\cup T)}$, a stable γ-preflow exists and can be found in $O(nm)$ time.*

Proof. The given problem is reduced to the stable flow one in an extended network N' with the graph $G' = (V, E')$ formed from G by adding for each inner vertex v, a new edge vt of capacity $c(vt) := \gamma(v)$, which is put as the last element of the order $\delta^{\text{out}}(v)$; namely, $e <_v^+ vt$ for all $e \in \delta_G^{\text{out}}(v)$. Here t is a distinguished sink in T. Let f' be a stable flow for N', and f its restriction to G. Then f is a γ-preflow and its stability follows from that of f'. (Indeed, if P is an unsaturated directed u–v path in G which, being considered as a path in G', is dominated at u, then there holds $f'(ut) = 0$, whence we obtain $\text{ex}_f(u) = 0$.)

We can further generalize the problem by introducing for each inner vertex v, besides $\gamma(v)$ as above, a bound $\beta(v) \in \mathbb{R}_+$ on the value $-\text{ex}(v)$. In other words, we consider a feasible (w.r.t. c) function $f : E \to \mathbb{R}_+$ satisfying

$$-\beta(v) \le \text{ex}_f(v) \le \gamma(v) \quad \forall v \in V - (S \cup T). \tag{5.2}$$

Such an f is called a (β, γ)-*quasi-flow*. One may think of the restriction $-\beta(v) \le \text{ex}_f(v)$ in (5.2) as a permission for the "agent" v to take from his reserve or attract from "outside" an amount of goods not exceeding $\beta(v)$ to send this further, together with the goods delivered through the entering edges (and, as before, it is allowed to the "agent" to store or send "aside" a part of the goods not exceeding $\gamma(v)$). Roughly speaking, for a (β, γ)-quasi-flow f, if the value $\Delta(v) := \text{ex}_f(v)$ is positive, then the "agent" v stores $\Delta(v)$ units of goods, while if it is negative, then the "agent" take from his reserve $-\Delta(v)$ units. Q.E.D.

We call a (β, γ)-quasi-flow *stable* if for each unsaturated directed u–v path P, at least one of the following is valid: (a) the vertex u is inner and P is strongly dominated at u (relative to γ), in the sense that (2.2) holds (with $v_0 = u$) and the excess at u is nonpositive (and no less than $-\beta(u)$); or (b) the vertex v is inner and P is *strongly dominated* at v (relative to β), which means that (2.3) is valid (with $v_k = v$) and the excess at v is nonnegative (and no more that $\gamma(v)$). When $\beta = \gamma = 0$, we obtain the definition of a stable flow.

For this generalization, we obtain the following

Proposition 4. *For a network $N = (G = (V, E), S, T, c)$ and vectors $\beta, \gamma \in \mathbb{R}_+^{V-(S\cup T)}$, a stable (β, γ)-quasi-flow exists and can be found in time $O(nm)$.*

Proof. Consider the stable flow problem in the extended network N' with the graph $G' = (V', E')$ formed from G by: (a) splitting each inner vertex v into two copies v' and v'', where v' inherits the entering edges from $\delta^{\text{in}}(v)$, and v'' does the leaving edges from $\delta^{\text{out}}(v)$ (keeping the capacities and orders on them); (b) adding edge $v'v''$ with a large capacity; and (c) adding edge $v't$ of capacity $c(v't) := \gamma(v)$ and edge sv'' of capacity $c(sv'') := \beta(v)$, where each of $v't$ and sv'' is assigned to be less preferable compared with the edge $v'v''$ and where s and t are distinguished source and sink, respectively. Let f' be a stable flow in N', and f its "image" in N. One can check that f is a (β, γ)-quasi-flow in N and that its stability follows from that of f'. Here the essential fact is that for each inner vertex $v \in V$, at least one of two values $f'(sv'')$ and $f'(v't)$ must be zero (which follows by considering the unsaturated path $(v', v'v'', v'')$). Q.E.D.

6. ADDITIONAL REMARKS

In this final section we discuss three additional interesting properties of stable flows. As before, we consider a directed network $N = (G = (V, E), S, T, c)$ with linear orders on $\delta^{\text{in}}(v)$ and $\delta^{\text{out}}(v)$ for inner vertices $v \in V - (S \cup T)$ (and subject to (2.1)).

(I) For a preflow f in a network $N = (G, S, T, c)$, we define the value of f to be the total excess at the sinks: $\text{val}(f) := \text{ex}_f(T)\left(= \sum_{t \in T} \text{ex}_f(t)\right)$. The following property is valid:

$$\begin{aligned}&\text{for a stable preflow } f \text{ in } N, \text{ there is a stable flow } f' \text{ in } N \text{ such that}\\ &f(e) \le f'(e) \text{ for all edges entering } T, \text{ and therefore, } \text{val}(f) \le \text{val}(f').\end{aligned} \tag{6.1}$$

Indeed, if f is not a flow (i.e., there is an excessive inner vertex), then f is transformed into a stable flow f' by applying a sequence of iterations of the basic algorithm. Any balancing operation does not change the values on the edges entering T, but any pushing operation can only increase these values, yielding the desired property (6.1).

(II) As to the values of stable flows, note that, generalizing classical results on stable marriages, stable bipartite b-matchings, etc., Fleiner [8, Section 4] established that:

$$\begin{aligned}&\text{for any two stable flows } f \text{ and } g \text{ in a network } N, \text{ the values } f(e)\\ &\text{and } g(e) \text{ coincide for each edge } e \text{ incident to a terminal.}\end{aligned} \tag{6.2}$$

As a consequence, $\text{val}(f) = \text{val}(g)$. Property (6.2) was proved in [8] (where two-terminus networks are considered and arbitrary edges incident to terminals are allowed) by use of a reduction to the corresponding property for stable allocations. One can give a direct and rather simple proof (considering our network $N = (G, S, T, c)$ as before).

For this purpose, let us associate with the function $f - g$ a decomposition on paths and cycles. More precisely, since f and g have no excessive inner vertices, one can form a family $\mathcal{C}$ consisting of simple paths and cycles C with weights $\Delta(C) > 0$ such that:

(i) for each $C \in \mathcal{C}$, the forward edges e in C satisfy $f(e) > g(e)$, and the backward edges e' satisfy $f(e') < g(e')$;

(ii) for each $e \in E$, the sum of weights $\Delta(C)$ on the paths/cycles C containing e is equal to $|f(e) - g(e)|$;

(iii) each path in $\mathcal{C}$ connects two different terminals and each cycle contains at most one terminal.

Suppose that f and g differ in some edge e incident to a terminal and consider the case when e leaves a source $s \in S$ (i.e., $e \in \delta^{\text{out}}(s)$). For definiteness, let $f(e) > g(e)$. Then there is $C \in \mathcal{C}$ containing e (as a forward edge); moreover, either (a) C is a path from s to $t \in T$, or (b) C is a path from s to $s' \in S$ (possibly $s' = s$).

In case (a), there is a sequence $s = v_0, v_1, \ldots, v_k = t$ of vertices in C (where k is odd) such that the portion of C from v_{i-1} to v_i has only forward edges for i odd, and only backward edges for i even. Then

$$\begin{aligned}&\text{for } i \text{ odd, there is a directed path } P_i \text{ from } v_{i-1} \text{ to } v_i \text{ where } f \text{ exceeds } g\\ &\text{on all edges, and therefore, } P_i \text{ is unsaturated for } g \text{ and does not contain}\\ &\text{edges free of } f\text{; in its turn, for } i \text{ even, there is a directed path } Q_i \text{ from}\\ &v_i \text{ to } v_{i-1} \text{ where } g \text{ exceeds } f\text{, and therefore, } Q_i \text{ is unsaturated for } f \text{ and}\\ &\text{does not contain edges free of } g.\end{aligned} \tag{6.3}$$

(Such paths form the concatenation $P_1 \cdot Q_2^{-1} \cdot P_3 \cdot Q_4^{-1} \cdot \ldots \cdot Q_{k-1}^{-1} \cdot P_k$ of C, where Q^{-1} stands for the path reversed to Q.) Let p_i (p_i') denote the first (resp. last) edge in P_i, and let q_j (q_j') denote the first (resp. last) edge in Q_j. One can see that

$$p_i', q_{i+1}' \in \delta^{\text{in}}(v_i) \text{ for each odd } i < k, \text{ and } q_i, p_{i+1} \in \delta^{\text{out}}(v_i) \text{ for } i \text{ even.} \tag{6.4}$$

Now we apply (6.3) and (6.4), moving step by step in the sequence of paths $P_1, Q_2, P_3, \ldots$. Since P_1 begins at the source $v_0 = s$ and is unsaturated for g and since $g(q_2') > 0$, from the stability of g it follows that

$q'_2 <^-_{v_1} p'_1$. Then from the facts that f is stable, Q_2 is unsaturated for f, the inequality $f(p_3) > 0$ holds, and Q_2 is not dominated at v_1 (since $f(p'_1) > 0$ and p'_1 is after q'_2), we obtain $p_3 <^+_{v_2} q_2$. And so forth. Coming to the path Q_{k-1}, we obtain $p_k <^+_{v_{k-1}} q_{k-1}$. But then the last directed path P_k (which ends at the sink $v_k = t$ and is unsaturated for g) is not dominated at the beginning vertex v_{k-1}, contrary to the stability of g.

In case (b), the reasonings are similar. Here we deal with the concatenation of paths of the form $P_1, Q_2^{-1}, \ldots, P_{k-1}, Q_k^{-1}$ (with k even). Now the last directed path Q_k begins at the source $v_k = s'$, is unsaturated for f and not dominated at the end vertex v_{k-1} (in view of $f(p'_{k-1}) > 0$ and $q'_k <^-_{v_{k-1}} p'_{k-1}$); this contradicts the stability of f. By symmetry reasons, the cases when f and g differ at an edge entering T are impossible as well. This completes the proof of (6.2).

(**III**) The previous construction can be extended. Namely, consider two stable flows f and g in a network $N = (G = (V, E), S, T, c)$. Let H be the subgraph of G induced by the edges in $A := \{e : f(e) > g(e)\}$ and $B := \{e : g(e) > f(e)\}$, and endow the edges e in H with the weights $\omega(e) := |f(e) - g(e)|$. By (6.2), H contains no terminal. Moreover, ω is decomposed into a nonnegative linear combination of the characteristic functions of simple regular cycles (and therefore, ω may be regarded as a "circulation").

Here we say that a (not necessarily simple) cycle C in H is *regular* (relative to (A, B)) if all forward edges in it belong to A and the backward ones belong to B. For a vertex v in such a C, we denote by $\Pi_C(v)$ the set of pairs of consecutive edges incident to v and following in the direction of the cycle. We call a pair $\pi \in \Pi_C(v)$ *especial* if π contains edges in both A and B (equivalently, both edges of π either enter v or leave v). In this case we say that a pair $\pi = (e, e')$ is *left* if e is more preferable for v than e'; otherwise the pair is called *right*. Arguing as in the proof of (6.2), from the stability of f and g one can conclude that

> for any regular cycle C, all especial pairs occurring in C have the same orientation, in the sense that all are left or all are right. (6.5)

This property is extended to the components K of H: all especial pairs occurring in regular cycles in a component K are simultaneously either left or right (this follows from the fact that any two such pairs π, π' can be included in one (non necessarily simple) cycle in K). According to this, we specify four types of components K. Namely, we say that: K has *type A* (*type B*) if all edges of K belong to the set A (resp. B); and K has *type L* (*type R*) if K has edges in both sets A and B (in which case we call the component K *rich*) and the orientations of all especial pairs in K are left (resp. right). Equivalently,

> a rich component K has type L if for any vertex v of K admitting an especial pair, in the set $\delta^{\text{in}}(v)$ the edges from A precede (more preferable than) the ones from B, and in the set $\delta^{\text{out}}(v)$ the edges from B precede the ones from A; in case of type R the situation is opposite. (6.6)

Using this, one can represent the set of stable flows in N as a lattice (which is done in [8] by use of a reduction to the stable allocation problem and appealing to the corresponding result in [4]). More precisely, define the following functions h, ℓ on E:

> h coincides with f on the components of types A and R, and coincides with g on the components of types B and L, while ℓ is defined conversely; for the remaining edges e, one puts $h(e) := \ell(e) := f(e) = g(e)$. (6.7)

One can see that h and ℓ are flows. Also for each inner vertex v, the flow h dominates the flows f, g on the set $\delta^{\text{out}}(v)$, and is dominated by these flows on the set $\delta^{\text{in}}(v)$, whereas ℓ behaves conversely. (Here for numerical functions a, b on an ordered set $(S, \prec)$, we say that *a dominates b* if either $a = b$ or there is $e \in S$ such that $a(e) > b(e)$, $a(e') \geq b(e')$ for $e' \prec e$, and $a(e'') \leq b(e'')$ for $e \prec e''$.)

Due to what is said above, one can obtain (we omit details here) that h and ℓ are nothing else than the flows $f \vee g$ and $f \wedge g$, respectively, that are pointed out in [8, Section 4]; in particular, both h, ℓ are stable.

ACKNOWLEDGMENTS

The author thanks the referee for a useful analysis of the initial version of this paper and pointing out the works [10, 12] that were unknown to him in the period of writing this version.

CONFLICT OF INTEREST

The author declares that he has no conflicts of interest.

REFERENCES

1. D. Gale and L. S. Shapley, "College admissions and the stability of marriage," Am. Math. Mon. **69** (1), 9–15 (1962).
2. R. W. Irving, "An efficient algorithm for the 'stable roommates' problem," J. Algorithms **6**, 577–595 (1985).
3. J. Tan, "A necessary and sufficient condition for the existence of a complete stable matching," J. Algorithms **12**, 154–178 (1991).
4. M. Baiou and M. Balinski, "Erratum: The stable allocation (or ordinal transportation) problem," Math. Oper. Res. **27** (4), 662–680 (2002).
5. B. C. Dean and S. Munshi, "Faster algorithms for stable allocation problems," Algorithmica **58** (1), 59–81 (2010).
6. D. D. Sleator and R. E. Tarjan, "A data structure for dynamic trees," J. Comput. Syst. Sci. **26** (3), 362–391 (1983).
7. D. D. Sleator and R. E. Tarjan, "Self-adjusting binary search trees," J. ACM **32** (3), 652–686 (1985).
8. T. Fleiner, "On stable matchings and flows," Algorithms **7**, 1–14 (2014).
9. M. Ostrovsky, "Stability in supply chain networks," Am. Econ. Rev. **98**, 897–923 (2006).
10. Á. Cseh and J. Matuschke, "New and simple algorithms for stable flow problems," Algorithmica **81** (6), 2557–2591 (2019).
11. A. V. Karzanov, "Determining the maximal flow in a network by the method of preflows," Sov. Math. Dokl. **15** (2), 434–437 (1974).
12. Á. Cseh, J. Matuschke, and M. Skutella, "Stable flows over time," Algorithms **6**, 532–545 (2013).