A Faster Algorithm for Maximum Flow in Directed Planar Graphs with Vertex Capacities

Julian Enoch
Department of Computer Science, University of Texas at Dallas, TX, USA

Kyle Fox
Department of Computer Science, University of Texas at Dallas, TX, USA

Dor Mesica
Efi Arazi School of Computer Science, The Interdisciplinary Center Herzliya, Israel

Shay Mozes
Efi Arazi School of Computer Science, The Interdisciplinary Center Herzliya, Israel

Abstract

We give an $O(k^3 \Delta n \log n \min(k, \log^2 n) \log^2(nC))$-time algorithm for computing maximum integer flows in planar graphs with integer arc *and vertex* capacities bounded by C, and k sources and sinks. This improves by a factor of $\max(k^2, k \log^2 n)$ over the fastest algorithm previously known for this problem [Wang, SODA 2019].

The speedup is obtained by two independent ideas. First we replace an iterative procedure of Wang that uses $O(k)$ invocations of an $O(k^3 n \log^3 n)$-time algorithm for maximum flow algorithm in a planar graph with k apices [Borradaile et al., FOCS 2012, SICOMP 2017], by an alternative procedure that only makes one invocation of the algorithm of Borradaile et al. Second, we show two alternatives for computing flows in the k-apex graphs that arise in our modification of Wang's procedure faster than the algorithm of Borradaile et al. In doing so, we introduce and analyze a sequential implementation of the parallel highest-distance push-relabel algorithm of Goldberg and Tarjan [JACM 1988].

2012 ACM Subject Classification Theory of computation → Network flows

Keywords and phrases flow, planar graphs, vertex capacities

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2021.72

Related Version *Preprint*: https://arxiv.org/abs/2101.11300

Funding Enoch and Fox were partially supported by NSF grant CCF-1942597. Mesica and Mozes were partially supported by Israel Science Foundation grant No. 592/17.

1 Introduction

The maximum flow problem has been extensively studied in many different settings and variations. This work concerns two related variants of the maximum flow problem in planar graphs. The first variant is the problem of computing a maximum flow in a directed planar network with integer arc *and vertex* capacities, and k sources and sinks. The second variant, which is used in algorithms for the first variant, is the problem of computing a maximum flow in a directed network that is nearly planar; there is a set of k vertices, called apices, whose removal turns the graph planar.

The problem of maximum flow in a planar graph with vertex capacities has been studied in several works since the 1990s [9, 14, 7, 2, 13]. For a more detailed survey of the history of this problem and other relevant results see [13] and references therein. Vertex capacities pose a challenge in planar graphs because the standard reduction from a flow network with vertex capacities to a flow network with only arc capacities does not preserve planarity. The problem can be solved by algorithms for maximum flow in sparse graphs (i.e., graphs with n vertices and $O(n)$ edges that are not necessarily planar). The fastest such algorithms currently known are an $O(n^2/\log n)$-time algorithm [11] for sparse graphs, and an $O(n^{4/3+o(1)}C^{1/3})$-time algorithm for sparse graphs with integer capacities bounded by C [8]. Until recently, there was no planarity exploiting algorithm for the case of more than a single source and a single sink. Significant progress on this problem was recently made by Wang [13]. Wang developed an $O(k^5\Delta n\log^3 n\log^2(nC))$-time algorithm, where k is the number of sources and sinks, Δ is the maximum vertex degree, and C is the largest capacity of a single vertex.[1] This is faster than using the two algorithms for general sparse graphs mentioned above when $k = \tilde{O}(n^{1/5}/(\Delta\log^2 C) + (nC)^{1/15}/\Delta)$.

Wang's algorithm uses multiple calls to an algorithm of Borradaile et al. [2] for computing a maximum flow in a k-apex graph with only arc capacities. The algorithm of Borradaile et al. [2] is based on an approach originally suggested by Hochstein and Weihe [6] for a slightly more restricted problem. In Borradaile et al.'s approach, a maximum flow in a k-apex graph with n vertices is computed by simulating the Push-Relabel algorithm of Goldberg and Tarjan [4] on a complete graph with k vertices, corresponding to the k apices of the input graph. Whenever the Push-Relabel algorithm pushes flow on an arc of the complete graph, the push operation is simulated by sending flow between the two corresponding apices in the input k-apex graph. This can be done efficiently using an $O(n\log^3 n)$ time multiple-source multiple-sink (MSMS) maximum flow algorithm in planar graphs, which is the main result of the paper of Borradaile et al. [2]. Overall, their algorithm for maximum flow in k-apex graphs takes $O(k^3 n\log^3 n)$ time. Flow in k-apex graphs can also be computed using the algorithms for sparse graphs mentioned above. The $O(k^3 n\log^3 n)$-time algorithm of Borradaile et al. is faster than these algorithms when $k = \tilde{O}(n^{1/3}/\log^2 C + (nC)^{1/9})$.

1.1 Our results and techniques

We improve the running time of Wang's algorithm to $O(k^3\Delta n\log n\min(k, \log^2 n)\log^2(nC))$. This is faster than Wang's result by a factor of $\max(k^2, k\log^2 n)$, extending the range of values of k for which the planarity exploiting algorithm is the fastest known algorithm for the problem to $k = \tilde{O}(n^{1/3}/(\Delta\log^2 C) + (nC)^{1/9}/\Delta)$. The improvement is achieved by two main ideas. At the heart of Wang's algorithm is an iterative procedure for eliminating excess flow from vertices violating the capacity constraints. Each iteration consists of computing a circulation with some desired properties. Wang computes this circulation using $O(k)$ calls to the algorithm of Borradaile et al. for maximum flow in k-apex graphs. We show how to compute this circulation using a constant number of invocations of the algorithm for k-apex graphs. This idea alone improves on Wang's algorithm by a factor of k.

To further improve the running time, we modify the algorithm of Borradaile et al. for maximum flow in k-apex graphs [2]. The algorithm of Borradaile et al. uses the Push-Relabel algorithm of Goldberg and Tarjan [4]. We introduce a sequential implementation of the parallel highest-distance Push-Relabel algorithm. In this algorithm, which we call batch-highest-distance, a single operation, Bulk-Push, pushes flow on multiple arcs simultaneously, instead of just on a single arc as in Goldberg and Tarjan's Push operation. More specifically, we simultaneously push flow on all admissible arcs whose tails have maximum height (see Section 3). This is reminiscent of parallel and distributed Push-Relabel algorithms [4, 3],

[1] In a very recent personal communication, Wang informed the authors of a subtle bug in the published version [13] of his result that led to the Δ factor not appearing in the claimed running time. The full version of Wang's paper (which has not been published yet) fixes the bug, and we refer to his corrected claims throughout the current work.

but our algorithm is sequential, not parallel. We prove that the total number of Bulk-Push operations performed by the batch-highest-distance algorithm is $O(k^2)$ (this should be compared to $O(k^3)$ Push operations for the FIFO or highest-distance Push-Relabel algorithms). We then show that, in the case of the k-apex graphs that show up in Wang's algorithm, we can implement each Bulk-Push operation using a constant number of invocations of the $O(n \log^3 n)$-time MSMS maximum flow algorithm for planar graphs [2]. Hence, we can find a maximum flow in such k-apex graphs in $O(k^2 n \log^3 n)$ time, which is faster by a factor of k than the time required by the algorithm of Borradaile et al.

We also give another way to modify the algorithm of Borradaile et al. for maximum flow in k-apex graphs; the second way is better when $k = o(\log^2 n)$. We observe that the structure of the k-apex graphs that arise in Wang's algorithm allows us to implement each of the $O(k^3)$ Push operations of the FIFO Push-Relabel algorithm used by Borradaile et al. in just $O(n \log n)$ time. This is done using a procedure due to Miller and Naor [10] for the case when all sources and sinks lie on a single face.

Roadmap. In Section 2 we provide preliminary background and notations. Section 3 describes the sequential implementation of the parallel highest-distance Push-Relabel algorithm, and its use in an algorithm for finding maximum flow in k-apex graphs. In Section 4 we describe how to use this Push-Relabel variant to obtain an improved algorithm for computing maximum flow in planar graphs with vertex capacities.

2 Preliminaries

All the graphs we consider in this paper are directed. For a graph G we use $V(G)$ and $E(G)$ to denote the vertex set and arc set of G, respectively. For any vertex $v \in V(G)$, let $\deg(v)$ denote the degree of v in G.

For a path P we denote by $P[u, v]$ the subpath of P that starts at u and ends at v. We denote by $P \circ Q$ the concatenation of two paths P, Q such that the first vertex of Q is the last vertex of P.

A *flow network* is a directed graph G with a capacity function $c : V(G) \cup E(G) \to [0, \infty)$ on the vertices and arcs of G, along with two disjoint sets $S, T \subset V(G)$ called *sources* and *sinks*, respectively. We assume without loss of generality that sources and sinks have infinite capacities, and that, for any arc $e = (u, v) \in E(G)$, the *reverse* arc (v, u), denoted $\text{rev}(e)$ is also in $E(G)$, and has capacity $c(\text{rev}(e)) = 0$.

Let $\rho : E(G) \to [0, \infty)$. To avoid clutter we write $\rho(u, v)$ instead on $\rho((u, v))$. For each vertex v let $\rho^{in}(v) = \sum_{(u,v)\in E(G)} \rho(u, v)$, and $\rho^{out}(v) = \sum_{(v,u)\in E(G)} \rho(v, u)$. The function ρ is called a *preflow* if it satisfies the following conservation constraint: for all $v \in V(G) \setminus (S)$, $\rho^{in}(v) \geq \rho^{out}(v)$. The *excess of a vertex v with respect to a preflow ρ* is defined by $\text{ex}(\rho, v) = \rho^{in}(v) - \rho^{out}(v)$. A preflow is *feasible on an arc $e \in E(G)$* if $\rho(e) \leq c(e)$. It is *feasible on a vertex $v \in V(G)$* if $\rho^{in}(v) \leq c(v)$. A preflow is said to be *feasible* if, in addition to the conservation constraint, it is feasible on all arcs and vertices. The value of a preflow ρ is defined as $|\rho| = \sum_{s\in S} \rho^{out}(s) - \rho^{in}(s)$. A preflow f satisfying $\text{ex}(f, v) = 0$ for all $v \in V(G) \setminus (S \cup T)$, is called a *flow*. A flow whose value is 0 is called a *circulation*. A *maximum flow* is a feasible flow whose value is maximum.

▶ Remark 1. The problem of finding a maximum flow in a flow network with multiple sources and sinks can be reduced to the single-source, single-sink case by adding a super source s and super sink t, and infinite-capacity arcs (s, s_i) and (t_i, t) for every $s_i \in S$ and $t_i \in T$. If the original network is planar then this transformation adds two apices to the graph. Throughout the paper, whenever we refer to the graph G, we mean the graph G after this transformation, i.e., with a single source, the apex s, and a single sink, the apex t.

The *violation of a flow f at a vertex v* is defined by $\text{vio}(f, v) = \max\{0, f^{in}(v) - c(v)\}$. Thus, if f is a feasible flow then $\text{vio}(f, v) = 0$ for all vertices v. The violation of the flow f is defined to be $\text{vio}(f) = \max_{v \in V(G)} \text{vio}(f, v)$.[2]

A preflow ρ is *acyclic* if there is no cycle C such that $\rho(e) > 0$ for every arc $e \in C$. A preflow *saturates* an arc e if $\rho(e) = c(e)$.

The sum of two preflows ρ and η is defined as follows. For every arc $e \in E(G)$, $(\rho+\eta)(e) = \max\{0, \rho(e) + \eta(e) - \rho(\text{rev}(e)) - \eta(\text{rev}(e))\}$. Multiplying the preflow ρ by some constant c to get the flow $c\rho$ is defined as $(c\rho)(e) = c \cdot \rho(e)$ for all $e \in E(G)$.

The *residual capacity* $c_\rho(e)$ of an arc e with respect to a preflow ρ is $c(e) - \rho(e) + \rho(\text{rev}(e))$. The *residual graph* of a flow network G with respect to a preflow ρ is the graph G where the capacity of every arc $e \in E(G)$ is set to $c_\rho(e)$. It is denoted by G_ρ. A path of G is called *augmenting* or *residual* (with respect to a preflow ρ) if it is also a path of G_ρ.

Suppose G and H are flow networks such that every arc in G is also an arc in H. If f' is a (pre)flow in H then the *restriction* of f' to G is the (pre)flow f in G defined by $f(e) = f'(e)$ for all $e \in E(G)$.

3 An algorithm for maximum flow in k-apex graphs

In this section we introduce a sequential implementation of the parallel highest-distance Push-Relabel algorithm of Goldberg and Tarjan [4], and use it in the algorithm of Borradaile et al. [2] for maximum flow in k-apex graphs. We first give a high-level description of the Push-Relabel algorithm.

3.1 The Push-Relabel algorithm [4]

Let H be a flow network (not necessarily planar) with source s and sink t, arc capacities $c : E(H) \to \mathbb{R}$, and no finite vertex capacities. The Push-Relabel algorithm maintains a feasible *preflow* function, ρ, on the arcs of H. A vertex u is called *active* if $\text{ex}(\rho, u) > 0$. The algorithm starts with a preflow that is zero on all arcs, except for the arcs leaving the source s, which are saturated. Thus, all the neighbors of s are initially active. When the algorithm terminates, no vertex is active and the preflow function is guaranteed to be a maximum flow. The algorithm also maintains a label function h (also known as distance or height function) over the vertices of H. The label function $h : V(H) \to \mathbb{N}$ is *valid* if $h(s) = |V(H)|$, $h(t) = 0$ and $h(u) \leq h(v) + 1$ for every residual arc $(u, v) \in E(H_\rho)$.

The algorithm progresses by performing two basic operations, Push and Relabel. A Push operation applies to an arc (u, v) if (u, v) is residual, $\text{ex}(\rho, u) > 0$, and $h(u) = h(v) + 1$. The operation moves excess flow from u to v by increasing the flow on e by $\min\{\text{ex}(\rho, u), c(e) - \rho(e)\}$.

The other basic operation, Relabel(u), assigns u the label $h(u) = \min\{h(v) : (u, v) \in E(H_\rho)\} + 1$ and applies to u only if u is active and $h(u)$ is not greater than the label of any neighbor of u in H_ρ. In other words, Relabel applies to an active vertex u only if the excess flow in u cannot be pushed out of u (because $h(u)$ is not high enough). The algorithm performs applicable Push and Relabel operations until no vertex is active.

[2] We define violations only with respect to flows (rather than preflows) because we will only discuss preflows in the context of flow networks without finite vertex capacities.

To fit our purposes, we think of the algorithm as one that only maintains explicitly the excess $\mathrm{ex}(\rho, v)$ and residual capacity $c_\rho(e)$ of each vertex v and arc e of H. The preflow ρ is implicit. In this view, a Push(u, v) operation decreases $\mathrm{ex}(\rho, u)$ and $c_\rho(u, v)$ by $\min\{\mathrm{ex}(\rho, u), c_\rho(u, v)\}$ and increases $\mathrm{ex}(\rho, v)$ and $c_\rho(v, u)$ by the same amount.

We reformulate Goldberg and Tarjan's correctness proof of the generic Push-Relabel algorithm to fit this view.

▶ **Lemma 2** ([4]). *Any algorithm that performs applicable Push and Relabel operations in any order satisfies the following properties and invariants:*
(1) $\mathrm{ex}(\rho, \cdot)$ *and* $c_\rho(\cdot)$ *are non-negative.*[3]
(2) *The function h is a valid labeling function.*
(3) *For all $v \in V$, the value of $h(v)$ never decreases, and strictly increases when Relabel(v) is called.*
(4) $h(v) \leq 2|V(H)| - 1$ *for all* $v \in V(H)$.
(5) *Immediately after Push(u, v) is performed, either (u, v) is saturated or u is inactive.*

Proof. Properties (1) and (5) are immediate from the definition of Push and the fact that excess and residual capacities only change during Push operations. Property (2) corresponds to Lemma 3.1 in [4], Property (3) is proved in Lemma 3.6 in [4], and Property (4) in Lemma 3.7 in [4]. ◀

▶ **Lemma 3** ([4, Lemma 3.3]). *Properties (1), (2) imply that there is no augmenting path from s to t at any point of the algorithm.*

▶ **Lemma 4** ([4, Lemma 3.8]). *Properties (3), (4) imply that the number of Relabel operations is at most $2|V(H)| - 1$ per vertex and at most $2|V(H)|^2$ overall.*

▶ **Lemma 5** ([4, Lemmas 3.9, 3.10]). *Properties (1)-(5) imply that the number of Push operations is $O(|V(H)|^2|E(H)|)$.*

By Lemmas 4 and 5, the algorithm terminates. Upon termination no vertex is active, so the implicit preflow ρ is in fact a feasible flow. By Lemma 3 ρ is a maximum flow from s to t.

Variants of the Push-Relabel algorithm differ in the order in which applicable Push and Relabel operations are applied. Some variants, such as FIFO, highest-distance, maximal-excess, etc., guarantee faster termination than the $O(|V(H)|^2|E(H)|)$ guarantee given above.

3.2 A sequential implementation of the parallel highest-distance Push-Relabel algorithm

We present a sequential implementation of the parallel highest-distance Push-Relabel algorithm, which we call Batch-Highest-Distance. This algorithm attempts to push flow on multiple edges simultaneously in an operation called Bulk-Push. In that sense, it resembles the parallel version of the highest-distance Push-Relabel algorithm. It is important to note, however, that Bulk-Push is a sequential operation and not a parallel/distributed one.

We define Bulk-Push, a batched version of the Push operation. Bulk-Push(U, W) operates on two sets of vertices, U and W. It is applicable under the following requirements:
(i) $\mathrm{ex}(u) > 0$ for all $u \in U$.
(ii) There exists an integer h such that $h(u) = h$ and $h(w) = h - 1$ for all $u \in U$ and $w \in W$.
(iii) There is a residual arc (u, w) for some $u \in U$ and $w \in W$.

[3] This corresponds to the function ρ being a feasible preflow.

Note that in a regular Push-Relabel algorithm, conditions (i) and (ii) imply that Push(u,w) is applicable to any residual arc (u,w) with $u \in U$ and $w \in W$. Condition (iii) guarantees there is at least one such arc. Bulk-Push pushes as much excess flow as possible from vertices in U to vertices in W so that after Bulk-Push the following property holds:

(5*) Immediately after Bulk-Push(U,W) is called, for all $u \in U$ and $w \in W$, either (u,w) is saturated or u is inactive.

We replace property (5) with the more general property (5*) in Lemma 2 and Lemma 5. With this modification, Lemmas 2, 3, 4 and 5 apply to our sequential implementation. The proofs from [4] need no change except replacing Push with Bulk-Push. Hence, our variant terminates correctly with a maximum flow from s to t.

▶ Remark. One may think of Bulk-Push(U,W) as performing in parallel all Push operations on arcs whose tail is in U and whose head is in W. However, not every maximum flow with sources U and sinks W can be achieved as the sum of flows pushed by multiple Push operations. For example, consider the case where U consists of a single vertex u, with $\mathrm{ex}(u) = 2$, $W = \{w_1, w_2\}$, and the residual capacities of (u,w_1) and (u,w_2) are both 2. Bulk-Push(U,W) may push one unit of excess flow from u on each of (u,w_1) and (u,w_2), but Push(u,w_i) would push 2 units of flow on (u,w_i), and no flow on the other arc. Therefore, the correctness of this variant cannot be argued just by simulating Bulk-Push by multiple Push operations. Instead we chose to argue correctness by stating the generalized property (5*).

We now discuss a concrete policy for choosing which Bulk-Push and Relabel operations to perform in the above algorithm. This policy is similar, but not identical, to the highest-distance Push-Relabel algorithm [4, 3]. As long as there is an active vertex, the algorithm repeatedly executes the following two steps, which together are called a *pulse*. Let h_{max} be the maximum label of an active vertex. That is, $h_{max} = \max\{h(v) : \mathrm{ex}(\rho, v) > 0\}$. Let H_{max} be the set of all the active vertices whose height is h_{max}. In the first step of the pulse, the algorithm invokes Bulk-Push(H_{max}, W) where W is the set of all vertices $w \in V$ such that $h(w) = h_{max} - 1$.[4] In the second step of the pulse, the algorithm applies the Relabel operation to all remaining active vertices in H_{max} in arbitrary order.

Algorithm 1 Batch-Highest-Distance(G, c).

```
1: Initialize $h(\cdot)$, $c_\rho(\cdot)$ and $\mathrm{ex}(\cdot)$
2: while there exists an active vertex do
3:     $h_{max} \leftarrow \max\{h(v) : \mathrm{ex}(\rho, v) > 0\}$
4:     $H_{max} \leftarrow \{v \in V(H) : \mathrm{ex}(\rho, v) > 0,\ h(v) = h_{max}\}$
5:     $W \leftarrow \{w \in V(H) : h(w) = h_{max} - 1\}$
6:     Bulk-Push$(H_{max}, W)$
7:     Relabel all active vertices in $H_{max}$ in arbitrary order
8: end while
```

▶ Remark. The crucial difference between this policy and the highest-distance Push-Relabel algorithm [4, 3] is that in the highest-distance algorithm a vertex u with height h_{max} is relabeled as soon as no more Push operations can be applied to u. In contrast, our variant first pushes flow from all vertices with height h_{max} and only then relabels all of them.

[4] Formally it may be that Bulk-Push(H_{max}, W) is not applicable because condition (iii) is not satisfied, e.g., when $W = \emptyset$. In such cases Bulk-Push does not push any flow. Condition (iii) is essential for the termination of the generic generalized algorithm, which may repeat such empty calls to Bulk-Push indefinitely. However, we prove in Lemma 6 that in our specific policy there are $O(|V(H)|^2)$ pulses, regardless of the flow pushed (or not pushed) by Bulk-Push in each pulse.

We partition the pulses into two types according to whether any vertices are relabeled in the relabel step of the pulse. A pulse in which at least one vertex is relabeled is called *saturating*. All other pulses are called *non-saturating*.[5]

By Lemma 4, the total number of Relabel operations executed by the batch-highest-distance algorithm is $O(|V(H)|^2)$. We now prove the same bound for Bulk-Push operations.

▶ **Lemma 6.** *The number of pulses (and hence also the number of calls to Bulk-Push) executed by the batch-highest-distance algorithm is $O(|V(H)|^2)$.*

Proof. Note that the Relabel step of a saturating pulse consists of at least one call to Relabel which strictly increases the height of an active vertex v whose height (before the increase) was h_{max}. Hence, a saturating pulse strictly increases the value of h_{max}. The fact that the height of each vertex never decreases and is bound by $2|V(H)|$ implies that (i) there are $O(|V(H)|^2)$ saturating pulses, and (ii) the total increase in h_{max} over all saturating Bulk-Push operations is $O(|V(H)|^2)$.

As for non-saturating pulses, note that since excess flow is always pushed to a vertex with lower height, the push step of a pulse does not create excess in any vertex with height greater than or equal to h_{max}, so all vertices with height greater than h_{max} remain inactive during the pulse. By property (5*), for every $u \in H_{max}$ and $w \in W$, either (u, w) is saturated, or u is inactive. Since the pulse is non-saturating, it follows that all the vertices in H_{max} become inactive during the pulse. Hence, the value of h_{max} strictly decreases during a non-saturating pulse. Since $h_{max} \geq 0$, the total decrease in h_{max} is also $O(|V(H)|^2)$, so there are $O(|V(H)|^2)$ non-saturating pulses. ◀

Note that we do not claim that implementing the Bulk-Push operation by applying applicable Push(u, w) operations for $u \in U$, $w \in W$ until no more such operations can be applied would result in fewer Push operations than the $O(|V(H)|^2|E(H)|)$ bound of Lemma 5 for the generic Push-Relabel algorithm. However, in Section 4 we will show a situation where each call to Bulk-Push can be efficiently implemented using a single invocation of a multiple-source multiple-sink algorithm in a planar graph.

3.3 The algorithm of Borradaile et al. for k-apex graphs [2]

The algorithm of Borradaile et al. [2, Section 5] uses the framework of Hochstein and Weihe [6]. Let H be a graph with a set $V^\times$ of k apices. Denote $V_0 = V(H) \setminus V^\times$. The goal is to compute a maximum flow in H from a source $s \in V(H)$ to a sink $t \in V(H)$. We assume that s and t are apices. This is without loss of generality since treating s and t as apices leaves the number of apices in $O(k)$. Let $K^\times$ be a complete graph over $V^\times$. The algorithm computes a maximum flow ρ from s to t in H by simulating a maximum flow computation from s to t in $K^\times$ using the Push-Relabel algorithm. Whenever a Push operation is performed on an arc (u, v) of $K^\times$ it is implemented by pushing flow from u to v in the graph H_{uv}, induced by $V_0 \cup \{u, v\}$ on the residual graph of H with respect to the flow computed so far. Note that, because no vertex of V_0 is an apex of H, H_{uv} is a 2-apex graph with apices u, v. Borradaile et al. use this fact to compute a maximum flow from u to v in H_{uv} as follows. They split u into multiple copies, each incident to a different vertex w for which (u, w) is an arc of H_{uv}. A similar process is then applied to v. Note that the resulting graph is planar. A maximum flow from u to v in H_{uv} is equivalent to a maximum flow with sources

[5] This is a generalization of the notions of saturating and non-saturating Push operations in [4].

the copies of u and sinks the copies of v in the resulting graph. This flow can be computed by the multiple-source multiple-sink maximum flow algorithm (the main result in [2]) in $O(|V(H)|\log^3|V(H)|)$ time.

The correctness of implementing the Push-Relabel algorithm on $K^\times$ in this way was proved by Hochstein and Weihe [6] by proving essentially that the algorithm satisfies the properties in Lemma 2. Borradaile et al. used the FIFO policy of Push-Relabel, which guarantees that the number of Push operations is $O(k^3)$, so the overall running time of their algorithm is $O(k^3|V(H)|\log^3|V(H)|)$.

3.4 An algorithm for maximum flow in k-apex graphs

We use the algorithm of Borradaile et al. for maximum flow in k-apex graphs from the previous section, but use our new batch-highest-distance Push-Relabel algorithm instead of the FIFO Push-Relabel algorithm to compute the maximum flow in $K^\times$. In order to implement the batch-highest-distance algorithm on $K^\times$ we only need to maintain the excess $\text{ex}(\rho, v)$ and labels $h(v)$ of each vertex $v \in K^\times$, and to be able to implement Bulk-Push so that after the execution, property (5*) is fulfilled. We do not define a flow function in $K^\times$ nor do we explicitly maintain residual capacities of arcs of $K^\times$. Instead, we maintain a preflow ρ in H, and say an arc (u, v) of $K^\times$ is residual if and only if there exists a residual path from u to v in H_ρ that is internally disjoint from the vertices of $V^\times$. Under this definition, there is no path of residual arcs in $K^\times$ starting at s and ending at t if and only if there is no such path in H. Since $K^\times$ has $O(k)$ vertices, by Lemma 6, the algorithm performs $O(k^2)$ pulses.

We next describe how a Bulk-Push(U, W) operation in $K^\times$ is implemented. Let $A = U \cup W$. Let H_A be the graph obtained from H_ρ by deleting the vertices $V^\times \setminus A$. Bulk-Push(U, W) in $K^\times$ is implemented by pushing a maximum flow in H_A with sources the vertices U and sinks the vertices W, with the additional restriction that the amount of flow leaving each vertex $u \in U$ is at most the excess of u. The efficiency of the procedure depends on how fast we can compute the maximum flow in H_A. We denote the time to execute a single Bulk-Push operation in the graph H_A by T_{BP}. Note that $T_{BP} = \Omega(k)$, as it takes $\Omega(k)$ time to construct H_A from H_ρ.

The proof of correctness is an easy adaptation of the proof of Hochstein and Weihe [6]. We cannot use their proof without change because Hochstein and Weihe considered only Push operations along a single arc of $K^\times$ rather than the Bulk-Push operations which involves more than a single pair of vertices of $K^\times$.

▶ **Lemma 7.** *Maximum flow in k-apex graphs can be computed in $O(k^2 \cdot T_{BP})$ time.*

Proof. We first show that the properties (1)-(4) in the statement of Lemma 2, and the generalized property (5*) from Section 3.2 hold. Property (1) holds since Bulk-Push(U, W) limits the amount of flow pushed from each vertex $u \in U$ by the excess of u. Properties (3) and (4) hold without change since Relabel is not changed.

To show property (5*) holds, recall that an arc (u, w) of $K^\times$ is residual if there exists, in the residual graph H_ρ with respect to the current preflow ρ, a residual path from u to w that is internally disjoint from any vertex of $V^\times$. With this definition it is immediate that property (5*) holds, since our implementation of Bulk-Push(U, W) pushes a maximum flow in H_A from U that is limited by the excess flow in each vertex of U. Hence, after Bulk-Push(U, W) is executed, for every $u \in U$ and $w \in W$, either there is no residual path from u to w in H_ρ that is internally disjoint from $V^\times$, or u is inactive.

As for property (2), since we did not change Relabel, h remains valid after calls to Relabel. It remains to show that h remains a valid labeling after Bulk-Push(U, W). Consider two vertices $a, b \in V^\times$. We will show that after Bulk-Push(U, W), either the arc (a, b) of $K^\times$ is

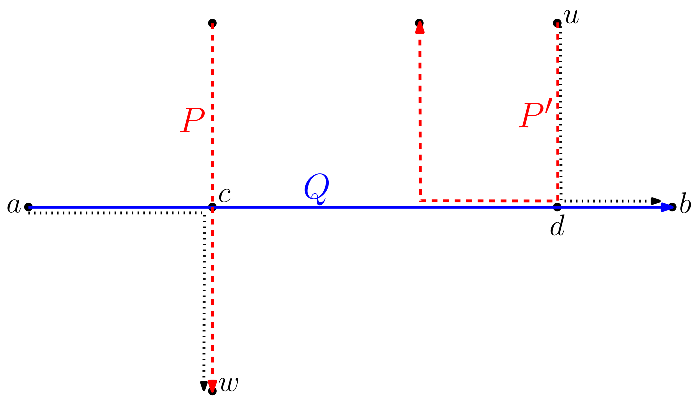

Figure 1 Illustration of property (2) in the proof of Lemma 7. A Bulk-Push(U, W) operation pushes flow along paths P and P' (dashed red paths). If Q (blue solid path) is residual after the Bulk-Push operation then the dotted black paths were residual before.

saturated (i.e., is no residual path from a to b in H_ρ), or $h(a) \leq h(b) + 1$. The flow pushed (in H_A) by the call Bulk-Push(U, W) can be decomposed into a set $\mathcal{P}$ of flow paths, each of which starts at a vertex of U and ends at a vertex of W.

Assume that after performing Bulk-Push(U, W) there is an augmenting a-to-b path, Q in H_ρ. If Q does not intersect any path $P \in \mathcal{P}$ then Q was residual before Bulk-Push(U, W) was called, so $h(a) \leq h(b) + 1$ because h was a valid labeling before the call. Otherwise, Q intersects some path in $\mathcal{P}$. Let c, d be the first and last vertices of Q that also belong to paths in $\mathcal{P}$. Let $P, P' \in \mathcal{P}$ be paths such that $c \in P$ and $d \in P'$. Let $w \in W$ be the last vertex of P and let $u \in U$ be the first vertex of P'. See Figure 1. Then, before Bulk-Push(U, W) was called, $Q[a, c] \circ P[c, w]$ was a residual path from a to w, and $P'[u, d] \circ Q[d, b]$ was a residual path from u to b. Since h was a valid labeling before the call, we have

$$h(u) \leq h(b) + 1 \quad \text{and} \quad h(a) \leq h(w) + 1.$$

Since $h(u) = h(w) + 1$ it follows that

$$h(a) \leq h(w) + 1 = h(u) \leq h(b) + 1,$$

showing property (2).

We have shown that properties (1)-(4) and (5*) hold. Hence, by Lemmas 6 and 4, the algorithm terminates after performing $O(|V^\times|^2) = O(k^2)$ Bulk-Push and Relabel operations. Since each Relabel takes $O(k)$ time, and each Bulk-Push takes $\Omega(k)$ time, the total running time of the algorithm is $O(k^2 \cdot T_{BP})$. By Lemma 3, when the algorithm terminates there is no residual path from s to t in $K^\times$. By our definition of residual arcs of $K^\times$ this implies that there is no residual path from s to t in H_ρ, so ρ is a maximum flow from s to t in H. ◀

4 A faster algorithm for maximum flow with vertex capacities

In this section, we give a faster algorithm for computing a maximum flow in a directed planar graph with integer arc and vertex capacities bounded by C, parameterized by the number k of terminal vertices (sources and sinks). The fastest algorithm currently known for this problem is by Wang [13] and runs in $O(k^5 \Delta n \text{ polylog}(nC))$ time. We first sketch Wang's algorithm, only detailing the parts that will be modified in our algorithm in Section 4.2.

4.1 Wang's algorithm

Wang's algorithm uses some auxiliary graphs, in which only the arcs are capacitated.

▶ **Definition 8** (The graph G°). *For a flow network G, the network G° is obtained by the following procedure. For each vertex $v \in V(G)$, replace v with an undirected cycle C_v with $d = \deg(v)$ vertices $v_1, ..., v_d$.*[6] *Each arc in C_v has capacity $c(v)/2$. Connect each arc incident to v with a different vertex v_i, preserving the clockwise order of the arcs so that the graph remains planar.*

Recall that if H and G are two graphs such that every arc of G is also an arc of H, then the restriction of a flow f' in H to G is a flow f in G such that $f(e) = f'(e)$ for all $e \in E(G)$. Thus we can speak of the restriction of a flow f° in G°, to a flow f in G.

Kaplan and Nussbaum [7] used the reduction from G to G° to compute a maximum flow in directed planar graphs with vertex capacities and a single source and a single sink. Their algorithm computes a maximum flow f° in the graph G°. Let f be the restriction of f° to G. Kaplan and Nussbaum then proved that, provided there is only a single source and a single sink, the values of the maximum flow in G° and in G are the same and that if f is acyclic then it is a feasible maximum flow in G.

However, when dealing with multiple source and sinks this approach fails because the resulting flow f may violate vertex capacities in G. In fact, the value of the maximum flow in G° may be greater than the value of the maximum flow in G. Wang proves that the set X of vertices whose capacities are violated by f (after canceling flow cycles) has size at most $k - 2$, and that the sum of the violations of the vertices in X is at most $(k - 2)C$. In order to overcome this obstacle, Wang's algorithm uses binary search to find the value $\lambda^\star$ of the maximum flow in G.

Let λ be the current candidate value for λ^*. The algorithm computes a flow f° with value λ in the graph G° (recall, G° has a super source s connected to all sources, and a super sink t, connected to all sinks. Thus G° has an apex set of size 2). Let f be the restriction of f° to G. As long as $\text{vio}(f) > 2k\Delta$, the algorithm improves f. This improvement phase, which will be described shortly, is the crux of the algorithm. If $\text{vio}(f) \leq 2k\Delta$, then $O(k^2\Delta)$ iterations of the classical Ford-Fulkerson algorithm suffice to get rid of all the remaining violations.

The improvement phase of the algorithm is based on finding a circulation g that cancels the violations on the infeasible vertices and does not create too much violations on other vertices. It can then be shown that adding $1/(k\Delta) \cdot g$ to the flow f decreases $\text{vio}(f)$ by a multiplicative factor of roughly $1 - 1/(k\Delta)$. After $O(k\Delta \log(kC))$ iterations of the improvement step, $\text{vio}(f)$ is at most $2k\Delta$.

In order to find the circulation g Wang defines the following auxiliary graph.

▶ **Definition 9** (The graph $G^\times$). *Let f be a flow in G. Let X be the set of infeasible vertices. I.e., vertices $x \in V(G)$ s.t. $f^{in}(x) > c(x)$. The graph $G^\times$ is defined as follows. Starting with G°, for each vertex $x \in X$, replace the cycle representing x with two vertices x^{in}, x^{out} and an arc (x^{in}, x^{out}) of capacity $c(x)$.*

Every arc of capacity c going from a vertex $u \notin C_x$ to a vertex in C_x becomes an arc (u, x^{in}) of capacity c. Similarly, every arc of capacity c going from a vertex of C_x to a vertex $u \notin C_x$ becomes an arc (x^{out}, u) with capacity c.

[6] By undirected cycle we mean that there are directed arcs in both directions between every pair of consecutive vertices of the cycle C_v.

Since arcs of G° can be identified with arcs of $G^\times$, the flow f° can be viewed as a flow $f^\times$ in $G^\times$, setting $f^\times(x^{in}, x^{out}) = \sum_{(u,x_i)\in E(G^\circ)} f^\circ(u, x_i)$. The flow $f^\times$ can then be restricted to a flow in G.

Wang proves that in order to find the circulation g described above, it suffices to compute a circulation $g^\times$ in $G^\times$ that satisfies the following properties:

1. $f^\times + g^\times$ is feasible in $G^\times$.
2. The restriction of $f^\times + g^\times$ to G has no violations on vertices of X.
3. The restriction of $f^\times + g^\times$ to G has at most $(k-2)\Delta \cdot \mathrm{vio}(f)$ violation on any vertex in $V(G) \setminus X$.

The desired circulation g is the restriction of $g^\times$ to G. If no such $g^\times$ exists then g does not exist, which implies that $\lambda > \lambda^*$. Otherwise, $\lambda \le \lambda^*$. The binary search for λ^* then continues with a different value of λ.

Wang essentially shows that any algorithm for finding $g^\times$ in $O(T)$ time, where $T = \Omega(n)$, yields an algorithm for maximum flow with vertex capacities in $O(k\Delta T \log(kC)\log(nC))$ time. The additional terms stem from the $O(k\Delta \log(kC))$ iterations of the improvement step, and the $\log(nC)$ steps of the binary search. Wang shows how to compute $g^\times$ in $T = O(k^4 n \log^3 n)$ time, by eliminating the violation at each vertex of X one after the other in some auxiliary graph obtained from $G^\times$. Thus, the overall running time of his algorithm is $O(k^5 \Delta n \log^3 n \log(kC)\log(nC))$.

4.2 A faster algorithm for computing $g^\times$

We propose a faster way of computing the circulation $g^\times$ by eliminating the violations in all the vertices of X in a single shot. Doing so correctly requires some care in defining the appropriate capacities in the auxiliary graph, since we only know that for each $x \in X$, $g^\times$ should eliminate at least $\mathrm{vio}(f, x)$ units of flow from x, but the actual amount of flow eliminated from x may have to be larger. This issue does not come up when resolving the violations one vertex at a time as was done by Wang.

Define an auxiliary graph H as follows. Starting with $G^\times_{f^\times}$, the residual graph of $G^\times$ with respect to $f^\times$,

- For each $x \in X$, set the capacity of the arc (x^{in}, x^{out}) to be 0 and the capacity of (x^{out}, x^{in}) to be $c(x)$.
- Add a super source s' and arcs (s', x^{in}) with capacity $\mathrm{vio}(f, x)$ for every $x \in X$.
- Add a super sink t' and arcs (x^{out}, t') with capacity $\mathrm{vio}(f, x)$ for every $x \in X$.

Note that $\{s,t\} \cup \bigcup_{x\in X}\{x^{in}, x^{out}\} \cup \{s', t'\}$ is an apex set of size $O(k)$ in H (recall from Remark 1 that s and t are the super source and super sink of the original graph G).

The circulation $g^\times$ can be found using the following algorithm. Find a maximum flow h' from s' to t' in H using Lemma 7. Convert h' to an acyclic flow h of the same value using the algorithm of Sleator and Tarjan [12] (cf. [13, Lemma 2.5]). If h does not saturate every arc incident to s' and t', return that the desired circulation g does not exist. Otherwise, h can be extended to the desired circulation $g^\times$ by setting $g^\times(x^{out}, x^{in}) = h(x^{out}, x^{in}) + \mathrm{vio}(f, x)$ for every $x \in X$ and $g^\times(e) = h(e)$ for all other arcs.

The following lemma shows that any single Bulk-Push operation in the algorithm of Lemma 7 on H can be implemented by a constant number of calls to the $O(n \log^3 n)$-time multiple-source multiple-sink maximum flow algorithm in planar graphs of Borradaile et al. [2]. There are two challenges that need to be overcome. First, the graph H is $O(k)$-apex graph rather than planar. Second, the algorithm of Borradaile et al. computes a maximum flow from multiple sources to multiple sinks, not a maximum flow under the restriction that each source sends at most some given limit. This is not a problem in the case of a single source, or a limit on just the total value of the flow, since then some of the flow pushed can be "undone". When each of the multiple sources has a different limit, undoing the flow from one source can create residual paths from another source that did not yet reach its limit.

▶ **Lemma 10.** *Any single Bulk-Push operation in the execution of the algorithm of Lemma 7 on the graph H defined above can be implemented in $O(n \log^3 n)$ time.*

Proof. Let $V^\times = \{s,t\} \cup \bigcup_{x \in X} \{x^{in}, x^{out}\} \cup \{s', t'\}$ be the set of apices of H. Recall that the algorithm of Lemma 7 invokes the batch-highest-distance Push-Relabel algorithm on a complete graph $K^\times$ over $V^\times$, and maintains a corresponding preflow in H. Consider a single Bulk-Push(U,W) operation from a set of apices U to a set of apices W. Let ρ denote the preflow pushed in H up to this Bulk-Push operation. Let $A = U \cup W$. To correctly implement Bulk-Push(U,W), we find a flow ρ' with sources U and sinks W in the graph H, which satisfies the following properties:

(i) For every $u \in U$, $\text{ex}(\rho + \rho', u) \geq 0$, and

(ii) For every $u \in U$ and $w \in W$, either $\text{ex}(\rho + \rho', u) = 0$ or there is no residual path in $H_{\rho+\rho'}$ from u to w that is internally disjoint from $V^\times$.

Condition (i) guarantees that ρ' does not push more flow from a vertex $u \in U$ than the current excess of u. Condition (ii) is condition (5*) from Section 3.2.

Let H'' be the graph obtained from H_ρ by deleting the vertices $V^\times \setminus A$. Note that the absence of residual paths that are internally disjoint from $V^\times$ in H'' is equivalent to the absence of such paths in H. We will compute ρ' using a constant number of invocations of the $O(n \log^3 n)$-time multiple-source multiple-sink maximum flow algorithm in planar graphs of Borradaile et al. [2]. Instead of invoking this algorithm on H'', which is not planar, we shall invoke it on modified versions of H'' which are planar.

Starting with H'', we split each vertex $w \in W$ into $\deg(w)$ copies. Each arc e that was incident to w before the split is now incident to a distinct copy of w, and is embedded so that it does not cross any other arc in the graph. Let H' denote the resulting graph, and let W' denote the set of vertices created as a result of splitting all the vertices of W.

The set W' replaces W as the set of sinks of the flow ρ' we need to compute. Note that U is an apex set in H'. We then build the flow ρ' gradually, by computing the following steps, each using a single invocation of the multiple-source multiple-sink maximum flow algorithm of Borradaile et al. in $O(n \log^3 n)$ time. In what follows, when we say that the flow ρ' satisfies condition (ii) for a subset U' of U we mean that for every $u \in U'$ and $w \in W$, either $\text{ex}(\rho + \rho', u) = 0$ or there is no residual path in $H_{\rho+\rho'}$ from u to w that is internally disjoint from $V^\times$.

(1) If $s \in U$, starting with H', we split s into $\deg(s)$ copies. Each arc e that was incident to s before the split is now incident to a distinct copy of s, and is embedded so that it does not cross any other arc in the graph. We also delete all other vertices of U. We compute in the resulting graph, which is planar, a maximum flow with sources the copies of s and the sinks W'. Let ρ'_s be the flow computed. If $|\rho'_s| > \text{ex}(\rho, s)$, we decrease $|\rho'_s|$ by pushing $|\rho'_s| - \text{ex}(\rho, s)$ units of flow back from W' to the copies of s. This can be done in $O(n)$ time in reverse topological order w.r.t. ρ'_s (cf. [2, Section 1.4]). We view ρ'_s as a flow in H, and set $\rho' = \rho'_s$. By construction ρ' satisfies condition (i), and satisfies condition (ii) for the subset $\{s\}$.

(2) If $t \in U$, starting with $H'_{\rho'}$, we repeat step (1) with t taking the role of s to compute a flow ρ'_t. Set $\rho' \leftarrow \rho' + \rho'_t$ By construction of ρ'_t, ρ' now satisfies satisfies condition (i), and satisfies condition (ii) for the subset $U \cap \{s,t\}$.

(3) Let U^{in} be the set $U \cap \{x^{in} : x \in X\}$. If $U^{in} \neq \emptyset$, starting with $H'_{\rho'}$, we delete all the vertices of U that are not in U^{in}. Note that, since the resulting graph does not contain s, t, s', t', nor any x^{out} for any $x \in X$, and since arcs incident to x^{in} only cross those

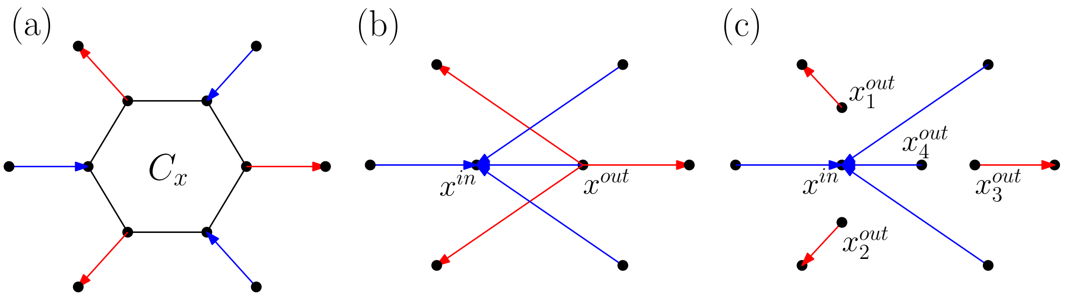

Figure 2 Illustration of the auxiliary graphs used in the algorithm of Lemma 10. Only a portion of the graphs around some vertex $x \in X$ is shown. (a) the graph G°. (b) the graph H. Note that the only crossings are between arcs incident to x^{in} and arcs incident to x^{out}. (c) the graph H' in the case that x^{out} belongs to W. x^{out} is split into multiple copies, eliminating all arc crossings.

incident to x^{out}, the resulting graph is planar. For every $x^{in} \in U^{in}$ we add a vertex x' and an arc (x', x^{in}) with capacity $\text{ex}(\rho, x^{in})$. The resulting graph is still planar. We compute a maximum flow ρ'_{in} with sources $\{x' : x^{in} \in U^{in}\}$ and sinks W'. We view ρ'_{in} as a flow in H, and set $\rho' \leftarrow \rho' + \rho'_{in}$. By construction of ρ'_{in}, ρ' now satisfies condition (i), and satisfies condition (ii) for the subset $U \cap (\{s,t\} \cup U^{in})$.

(4) We repeat step (3) with *out* taking the role of *in* to compute a flow ρ'_{out}. By construction of ρ'_{in}, ρ' now satisfies condition (i), and satisfies condition (ii) for $U \cap (\{s,t\} \cup U^{in} \cup U^{out})$.

Since s' and t' are the source and sink of the flow computed by the Push-Relabel algorithm, they are never active vertices, so they never belong to U. Hence $\{s,t\} \cup U^{in} \cup U^{out} \supseteq U$, and conditions (i) and (ii) are fully satisfied by ρ'. ◀

Using Lemma 10 and Lemma 7, we get the following lemma.

▶ **Lemma 11.** *The algorithm described above finds a circulation $g^{\times}$ such that*
1. *$f^{\times} + g^{\times}$ is feasible in $G^{\times}$.*
2. *The restriction of $f^{\times} + g^{\times}$ to G has no violations at vertices of X.*
3. *The restriction of $f^{\times} + g^{\times}$ to G has violation at most $(k-2)\Delta \cdot \text{vio}(f)$ at any vertex in $V(G) \setminus X$.*

in $O(k^2 n \log^3 n)$ time if such a circulation exists.

Proof. We first analyze the running time. Computing the graph H can be done in $O(n)$ time. Computing the flow h' in H using the algorithm of Lemma 7 takes $O(k^2 \cdot T_{BP})$ time. By Lemma 10, $T_{BP} = O(n \log^3 n)$ for the graph H. Transforming h' to an acyclic flow h using the algorithm of Sleator and Tarjan [12] takes $O(n \log n)$ time. Finally, computing $g^{\times}$ from h takes $O(n)$ time. Hence, the total time to compute $g^{\times}$ is $O(k^2 n \log^3 n)$.

In order to prove the correctness of the algorithm, we will first prove that there exists a feasible flow h in H that saturates every arc incident to s' and t' if and only if there exists a circulation $g^{\times}$ in $G^{\times}$ that satisfies conditions (1) and (2) in the statement of the lemma.

($\Leftarrow$) Assume the circulation $g^{\times}$ exists in $G^{\times}$. Define a flow h in H as follows. For every arc $e \in E(H)$ not of the form (x^{out}, x^{in}) set $h(e) = g^{\times}(e)$. For every $x \in X$, set $h(x^{out}, x^{in}) = g^{\times}(x^{out}, x^{in}) - \text{vio}(f, x)$, $h(s, x^{in}) = \text{vio}(f, x)$ and $h(x^{out}, t) = \text{vio}(f, x)$. Since the restriction of $f^{\times} + g^{\times}$ to G has no violations on the vertices of x, $g^{\times}(x^{out}, x^{in}) \geq \text{vio}(f, x)$, so $h(x^{out}, x^{in}) \geq 0$ and h is a well defined flow. By definition, the flow h saturates every arc incident to s' and t'.

To show that h is feasible in H it is enough to show that $h(x^{out}, x^{in}) \leq c(x)$ for every $x \in X$ (on all other arcs h is feasible because $g^\times$ is feasible in $G^\times_{f^\times}$). Let $x \in X$. Since $f^\times + g^\times$ is feasible in $G^\times$, $g^\times(x^{out}, x^{in}) \leq f^\times(x^{in}, x^{out})$. Since $h(x^{out}, x^{in}) = g^\times(x^{out}, x^{in}) - \text{vio}(f, x)$, $h(x^{out}, x^{in}) \leq f^\times(x^{in}, x^{out}) - \text{vio}(f, x) = c(x)$.

($\Rightarrow$) Assume there exist a feasible flow h in H that saturates every arc incident to s' and t', and let $g^\times$ be the circulation obtained from h as described above. We show that $f^\times + g^\times$ is feasible in $G^\times$. On all arcs e not of the form (x^{out}, x^{in}), $g^\times(e) = h(e)$ and the capacity of e in $G^\times_{f^\times}$ equals the capacity of e in H. Therefore, since h is a feasible flow in H, $g^\times$ is feasible on e in $G^\times_{f^\times}$, so $f^\times + g^\times$ is feasible on e in $G^\times$. We now focus on the arcs (x^{out}, x^{in}) for each $x \in X$. Let $e = (x^{out}, x^{in})$. Observe that $0 \leq h(e) \leq c(x)$. Since $g^\times(e) = h(e) + \text{vio}(f, x)$ we have that $\text{vio}(f, x) \leq g^\times(e) \leq c(x) + \text{vio}(f, x) = f^\times(e)$. Since $(f^\times + g^\times)(x^{in}, x^{out}) = f^\times(x^{in}, x^{out}) - g^\times(x^{out}, x^{in})$, we have $0 \leq (f^\times + g^\times)(x^{in}, x^{out}) \leq c(x)$, so $f^\times + g^\times$ is feasible in $G^\times$.

To finish proving the ($\Rightarrow$) direction, we show that the restriction of $f^\times + g^\times$ to G has no violations on the vertices of X. By definition of $G^\times$ and of residual graph, the only arcs in $G^\times_{f^\times}$ that can carry flow out of x^{in} are the reverses of the arcs that carry flow into x in f, and the only arcs that can carry flow into x^{out} are the reverses of the arcs that carry flow out of x in f. We will show that $(f + g)^{in}(x) \leq c(x)$ by considering separately the contribution of the flow on arcs of G that in $G^\times$ are incident to x^{out}, and arcs of G that in $G^\times$ are incident to x^{in}.

The only arc of $f^\times$ that carries flow into x^{out} is (x^{in}, x^{out}). Thus, there is no arc e of G such that $f^\times(e)$ carries flow into x^{out}. Since $g^\times$ only carries flow into x^{out} along the reverses of arcs that carry flow out of x^{out} in $f^\times$ and since for every such arc e', $g^\times(e') \leq f^\times(\text{rev}(e'))$, there is also no arc e of G such that $(f^\times + g^\times)(e)$ carries flow into x^{out}.

The total flow that $f^\times$ carries into x^{in} is $c(x) + \text{vio}(f, x)$. Let z denote the total amount of flow that $g^\times$ carries into x^{in}. Since the only arc incident to x^{in} that carries flow in $g^\times$ and does not belong to G is (x^{out}, x^{in}), the total amount of flow that $g^\times$ carries into x^{in} on arcs that belong to G is $z - g^\times(x^{out}, x^{in})$. On the other hand, $g^\times$ carries z units of flow out of x^{in}, and all of this flow is pushed along the reverses of arcs that carry flow into x^{in} in $f^\times$ (and also belong to G). Hence, the total amount of flow that $f^\times + g^\times$ carries into x^{in} on arcs that belong to G is $c(x) + \text{vio}(f, x) + (z - g^\times(x^{out}, x^{in})) - z$. Therefore,

$$\begin{aligned}
(f+g)^{in}(x) &= c(x) + \text{vio}(f, x) - g^\times(x^{out}, x^{in})) \\
&= c(x) + \text{vio}(f, x) - (h(x^{out}, x^{in}) + \text{vio}(f, x)) \\
&= c(x) - h(x^{out}, x^{in}) \\
&\leq c(x).
\end{aligned}$$

We have thus shown that the algorithm computes a flow $g^\times$ satisfying conditions (1) and (2) in the statement of the lemma. To see that condition (3) is also satisfied, note that the value of the flow h is $\sum_{x \in X} \text{vio}(f, x) \leq (k-2) \cdot \text{vio}(f)$. Since h is acyclic, $h^{in}(v) \leq (k-2) \cdot \text{vio}(f)$ for all $v \in H$. Since for all $v \in V(G) \setminus X$, $f^{in}(v) \leq c(v)$, and $h^{in}(v) \leq \Delta(g^\times)^{in}(v)$, it follows that the violation of $f^\times + g^\times$ at v is at most $(k-2)\Delta \cdot \text{vio}(f)$. ◀

Using the $O(k^2 n \log^3 n)$-time algorithm of Lemma 11 in the improvement phase of Wang's algorithm instead of using Wang's $O(k^4 n \log^3 n)$-time procedure for this phase results in a running time of $O(k^3 \Delta n \text{ polylog}(nC))$ for finding a maximum flow in G.

4.3 The case $k = o(\log^2 n)$

We now provide an algorithm that is faster for small $k = o(\log^2 n)$. Specifically, we describe an algorithm for computing the circulation $g^\times$ that runs in $O(k^3 n \log n)$ time instead of the $O(k^2 n \log^3 n)$ time required by the algorithm of Lemma 11.

We use the same auxiliary graph H as defined above and again compute a maximum flow h' from s' to t' in H. Let $V^\times = \{s,t\} \cup \bigcup_{x \in X}\{x^{in}, x^{out}\} \cup \{s', t'\} \cup S \cup T$ be the set of apices of H *along with the original sources* S *and sinks* T *of* G, and let $K^\times$ be the complete graph on $V^\times$. Instead of using the batch-highest-distance Push-Relabel algorithm as in Lemma 7, we more directly follow the strategy of Borradaile et al. [2] by simulating a maximum flow computation from s to t in $K^\times$ using the FIFO Push-Relabel algorithm. We do not wish to directly use the multiple-source multiple-sink flow algorithm of Borradaile et al. [2], because then each of the $O(k^3)$ Push operations would take $O(n \log^3 n)$ time.

▶ **Lemma 12.** *Any single (individual arc)* Push *operation in the graph H defined above can be implemented in $O(n \log n)$ time.*

Proof. Consider a single Push(u, v) operation where $u, v \in V^\times$. Let ρ denote the preflow pushed in H by the FIFO Push-Relabel algorithm up to this Push operation. We find a flow ρ' with source u and sink v in the graph H such that either $\text{ex}(\rho + \rho', u) = 0$ or $\text{ex}(\rho + \rho', u) > 0$ and there is no residual path in $H_{\rho+\rho'}$ from u to v that is internally disjoint from $V^\times$.

Let H' be the graph obtained from H_ρ by deleting vertices $V^\times \setminus \{u, v\}$. Instead of invoking the $O(n \log^3 n)$-time multiple-source multiple-sink maximum flow algorithm of Borradaile et al. [2], we will compute ρ' as follows. As before, we must consider a few different cases.

- If $u = s$ or $v = s$, then $v \in S$ or $u \in S$, respectively. We push up to $\text{ex}(\rho, u)$ units of flow directly along the arc (u, v) in constant time, either saturating the arc or reducing the excess flow in u to 0. We may similarly push directly along the arc (u, v) in constant time if one of u or v is one of t, s', or t' instead.
- If $u \in \{x_1^{in}, x_1^{out}\}$ and $v \in \{x_2^{in}, x_2^{out}\}$ for two distinct vertices $x_1, x_2 \in X$, then the graph H' is planar. We add a vertex u' and an arc (u', u) with capacity $\text{ex}(\rho, u)$ and compute the maximum flow ρ' with source u' and sink v using the single-source single-sink maximum flow algorithm in planar graphs of Borradaile and Klein [1].
- If neither of the above cases apply, then $u, v \in \{x^{in}, x^{out}\}$ for some $x \in X$. If arc (u, v) has positive residual capacity, we push up to $\text{ex}(\rho, u)$ units of flow directly along it in constant time. Similar to Step 1 in the proof of Lemma 10, starting with H', we split u into $\deg(u)$ copies so that each arc that was incident to u is now incident to a distinct copy of u. Similarly, we split v into $\deg(v)$ copies so each arc that was incident to v is now incident to a distinct copy of v. The resulting graph is planar, and all copies of u and v lie on a common face. As mentioned by Borradaile et al. [2, p. 1280], we can then plug the linear time shortest paths in planar graphs algorithm of Henzinger et al. [5] into a divide-and-conquer procedure of Miller and Naor [10] to compute a maximum flow ρ'_u with sources the copies of u and sinks the copies of v in $O(n \log n)$ time. Again, if the value of this flow is greater than the excess of u, we push the appropriate amount of flow back to the copies of u in $O(n)$ time. Finally, we view ρ'_u as a flow in H to set $\rho' = \rho'_u$. ◀

We immediately get a variation of Lemma 11 with a running time of $O(k^3 n \log n)$. By using the better of the two procedures for computing $g^\times$, we get our main theorem.

▶ **Theorem 13.** *A maximum flow in an n-vertex planar flow network G with integer arc and vertex capacities bounded by C can be computed in $O(k^3 \Delta n \log n \min(k, \log^2 n) \log(kC) \log(nC))$ time.*

References

1 Glencora Borradaile and Philip N. Klein. An $O(n \log n)$ algorithm for maximum st-flow in a directed planar graph. *J. ACM*, 56(2):9:1–9:30, 2009. doi:10.1145/1502793.1502798.

2 Glencora Borradaile, Philip N. Klein, Shay Mozes, Yahav Nussbaum, and Christian Wulff-Nilsen. Multiple-source multiple-sink maximum flow in directed planar graphs in near-linear time. *SIAM J. Comput.*, 46(4):1280–1303, 2017. doi:10.1137/15M1042929.

3 Joseph Cheriyan and S. N. Maheshwari. Analysis of preflow push algorithms for maximum network flow. *SIAM J. Comput.*, 18(6):1057–1086, 1989. doi:10.1137/0218072.

4 Andrew V. Goldberg and Robert Endre Tarjan. A new approach to the maximum-flow problem. *J. ACM*, 35(4):921–940, 1988. doi:10.1145/48014.61051.

5 Monika Rauch Henzinger, Philip N. Klein, Satish Rao, and Sairam Subramanian. Faster shortest-path algorithms for planar graphs. *J. Comput. Syst. Sci.*, 55(1):3–23, 1997. doi:10.1006/jcss.1997.1493.

6 Jan M. Hochstein and Karsten Weihe. Maximum s-t-flow with k crossings in $O(k^3 n \log n)$ time. In Nikhil Bansal, Kirk Pruhs, and Clifford Stein, editors, *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2007, New Orleans, Louisiana, USA, January 7-9, 2007*, pages 843–847. SIAM, 2007. URL: http://dl.acm.org/citation.cfm?id=1283383.1283473.

7 Haim Kaplan and Yahav Nussbaum. Maximum flow in directed planar graphs with vertex capacities. *Algorithmica*, 61(1):174–189, 2011. doi:10.1007/s00453-010-9436-7.

8 Tarun Kathuria, Yang P. Liu, and Aaron Sidford. Unit capacity maxflow in almost $O(m^{4/3})$ time. In *61st IEEE Annual Symposium on Foundations of Computer Science, FOCS 2020, Durham, NC, USA, November 16-19, 2020*, pages 119–130. IEEE, 2020. doi:10.1109/FOCS46700.2020.00020.

9 Samir Khuller and Joseph Naor. Flow in planar graphs with vertex capacities. *Algorithmica*, 11(3):200–225, 1994. doi:10.1007/BF01240733.

10 Gary L. Miller and Joseph Naor. Flow in planar graphs with multiple sources and sinks. *SIAM J. Comput.*, 24(5):1002–1017, 1995. doi:10.1137/S0097539789162997.

11 James B. Orlin. Max flows in $O(nm)$ time, or better. In Dan Boneh, Tim Roughgarden, and Joan Feigenbaum, editors, *Symposium on Theory of Computing Conference, STOC'13, Palo Alto, CA, USA, June 1-4, 2013*, pages 765–774. ACM, 2013. doi:10.1145/2488608.2488705.

12 Daniel Dominic Sleator and Robert Endre Tarjan. A data structure for dynamic trees. *J. Comput. Syst. Sci.*, 26(3):362–391, 1983. doi:10.1016/0022-0000(83)90006-5.

13 Yipu Wang. Maximum integer flows in directed planar graphs with vertex capacities and multiple sources and sinks. In Timothy M. Chan, editor, *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*, pages 554–568. SIAM, 2019. doi:10.1137/1.9781611975482.35.

14 Xianchao Zhang, Weifa Liang, and Guoliang Chen. Computing maximum flows in undirected planar networks with both edge and vertex capacities. In Xiaodong Hu and Jie Wang, editors, *Computing and Combinatorics, 14th Annual International Conference, COCOON 2008, Dalian, China, June 27-29, 2008, Proceedings*, volume 5092 of *Lecture Notes in Computer Science*, pages 577–586. Springer, 2008. doi:10.1007/978-3-540-69733-6_57.