A New Minimum Cost Flow Algorithm with Applications to Graph Drawing*

Ashim Garg and Roberto Tamassia

Department of Computer Science
Brown University
Providence, RI 02912–1910, USA
{ag,rt}@cs.brown.edu

Abstract. Let N be a single-source single-sink flow network with n nodes, m arcs, and positive arc costs. We present a pseudo-polynomial algorithm that computes a maximum flow of minimum cost for N in time $O(\chi^{3/4} m\sqrt{\log n})$, where χ is the cost of the flow. This improves upon previously known methods for networks where the minimum cost of the flow is small. We also show an application of our flow algorithm to a well-known graph drawing problem. Namely, we show how to compute a planar orthogonal drawing with the minimum number of bends for an n-vertex embedded planar graph in time $O(n^{7/4}\sqrt{\log n})$. This is the first subquadratic algorithm for bend minimization. The previous best bound for this problem was $O(n^2 \log n)$ [19].

1 Introduction

Minimum cost flow is a fundamental problem in network optimization, and a large body of literature exists on theoretical and practical methods for solving it [1].

While sophisticated polynomial and strongly-polynomial algorithms for minimum cost flow have been recently devised [1], their complexity is $\Omega(nm \log n)$, where n and m denote the number of nodes and arcs, respectively, of the flow network, and they may perform worse than some of the simpler pseudo-polynomial algorithms when the magnitude and/or cost of the flow are small. For example, let the magnitude of the optimal flow be ϕ. One can achieve running time $O(\phi\, m \log n)$ with a simple minimum cost flow algorithm based on successive flow augmentations along a minimum cost path determined with Dijkstra's method.

This paper is organized as follows: In Section 2, we provide background material on network flow algorithms and present some preliminary results. In Section 3, we present our new pseudo-polynomial algorithm that computes a minimum cost flow for a flow network with positive arc costs in time $O(\chi^{3/4} m\sqrt{\log n})$,

* Research supported in part by the National Science Foundation under grant CCR–9423847.

where χ is the cost of the flow. This improves upon all previously known methods for networks with positive arc costs such that $\chi = o(n^{4/3}\sqrt{\log n})$ and $\chi = o(\phi^{4/3}\sqrt{\log n})$.

In Section 4, we give an application of our minimum cost flow algorithm to an important graph drawing problem. Namely, we show how to compute a planar orthogonal drawing with the minimum number of bends for an n-vertex embedded planar graph in time $O(n^{7/4}\sqrt{\log n})$. This is the first subquadratic algorithm for bend minimization. The previous best bound for this problem was $O(n^2 \log n)$ [19]. Improving the time complexity of bend minimization was mentioned as one of the major open problems of graph drawing in a standard bibliographic survey of the field [3].

2 Preliminaries

In this section, we review some basic concepts and definitions related to network flows. We also give a brief description of Algorithms *AugPath*, *BlockFlow* and *PrimDua*, which embody three well-known techniques for computing flows in flow networks. The notation followed is generally the one of [15].

2.1 Definitions

A *flow network* N is a directed graph such that N has two disjoint non-empty sets of distinguished nodes, called its *sources* and *sinks*, and each arc e of N has a cost $c(e)$ and a capacity $u(e)$ associated with it, where $c(e)$ is an integer and $u(e)$ is a positive integer. Let m and n be the number of arcs and nodes, respectively, of N. We assume that N is connected so that $m \geq n - 1$. N is a flow network with *positive arc costs* if the cost of each arc is at least 1. N is a *single-source single-sink* flow network if it has only one source and only one sink.

Let *inarc(v)* and *outarc(v)* be the set of incoming and outgoing arcs of a node v of N. A *flow* f in N is an assignment of a non-negative integer label $f(e)$ to each arc e of N such that:

- for each arc e, $f(e) \leq u(e)$;
- for each node v, where v is not a source or a sink,

$$\sum_{e \in inarc(v)} f(e) = \sum_{e \in outarc(v)} f(e) \ (\textit{flow conservation});$$

- for each source s,

$$\sum_{e \in inarc(s)} f(e) \leq \sum_{e \in outarc(s)} f(e); \text{ and}$$

- for each sink t,

$$\sum_{e \in inarc(t)} f(e) \geq \sum_{e \in outarc(t)} f(e).$$

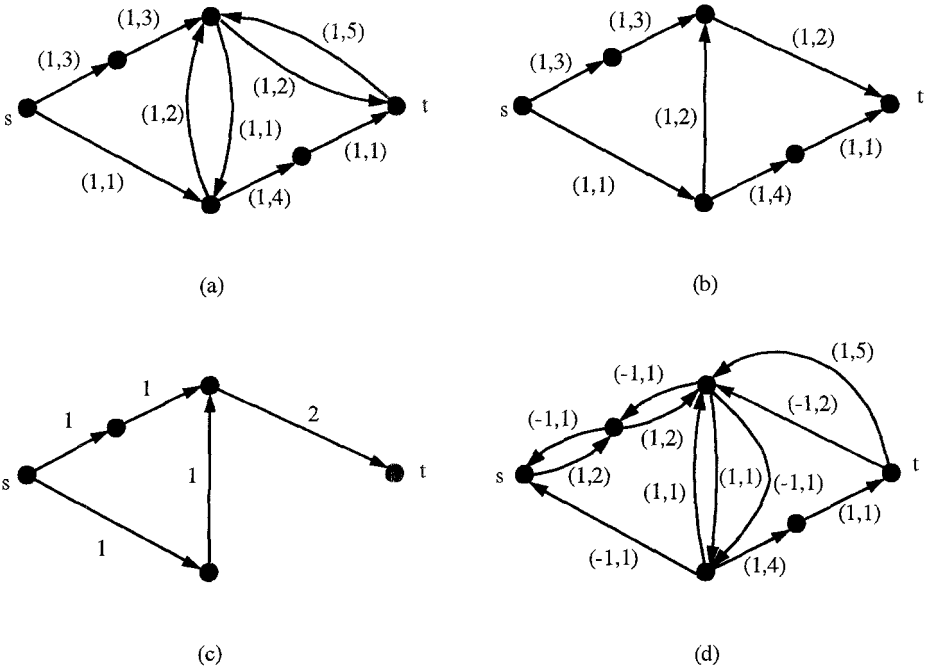

Fig. 1. **(a)** A flow network N with source s and sink t. **(b)** The shallowest layered network $L(N, \mathbf{0})$ of N with respect to zero flow; L is also the admissible flow network of N with respect to zero flow because each arc of N has cost 1. **(c)** A Blocking flow f of L. **(d)** Residual network $R(N, f)$ of N with respect to f. In parts (a), (b), and (d) we have labeled each arc e by the pair $(c(e), u(e))$. In part (c) we have labeled each arc by the amount of flow in it, and have shown only the arcs with non-zero flow.

We say that $f(e)$ is the flow in arc e *due to* flow f. For notational convenience, we do not distinguish between a flow in an arc and its magnitude. An arc e is *saturated by* flow f if $f(e) = u(e)$. Let S be the set of the sources of N. The *magnitude* of flow f, denoted by $|f|$, is defined as $|f| = \sum_{s \in S}(\sum_{e \in outarc(s)} f(e) - \sum_{e \in inarc(s)} f(e))$, i.e., the "net" flow going out of the sources of N.

The *cost* of a flow f, denoted by $c(f)$, is defined as the sum of the costs of the flows due to f in the arcs of N. A *maximum* flow of N is a flow with maximum magnitude among all the flows of N. A *minimum cost* flow of N is a maximum flow with minimum cost among all the maximum flows of N. A *zero* flow, denoted by $\mathbf{0}$, is a flow with zero magnitude. A *non-zero* flow is a flow with non-zero magnitude. For simplicity of discussion, we assume in the rest of this section that N is a flow network with a single source s and a single sink t.

The *cost* of a directed path of N is equal to the sum of the costs of its arcs. A *shortest* directed path from a node u to a node v is a directed path with the least number of arcs among all the directed paths from u to v.

The *residual* flow network $R(N, f)$ of N with respect to a flow f (see Figure 1(c–d)) is the flow network such that for every arc $e(u, v)$ of N, it con-

sists of arcs $e' = e(u, v)$ and $e'' = (v, u)$ such that $c(e') = -c(e'') = c(e)$, $u(e') = u(e) - f(e)$, and $u(e'') = f(e)$ (notice that from the definition of a flow network, $R(N, f)$ can only have arcs with non-zero capacities, hence if e' or e'' has zero capacity, then we delete it from $R(N, f)$). A directed path of $R(N, f)$ from s to t is an *augmenting path* of N with respect to flow f. A flow g in $R(N, f)$ corresponds to a flow f^* in N as follows: for every arc e of N, $f^*(e) = f(e) + g(e') - g(e'')$; we say that flow g is the new flow *pushed* in N (and also in $R(N, f)$). Notice that because $u(e'') = f(e)$, we have that $g(e'') \leq u(e'') \leq f(e)$, and hence, $f^*(e) \geq 0$. It can also be shown easily that f^* satisfies the capacity and demand-supply constraints, and the flow conservation property. Therefore, from a high-level perspective, the concepts of residual flow networks and augmenting paths give us a convenient way of introducing more flow in the original flow network by reducing flow in certain arcs and increasing it in others.

A *layered* flow network $L(N, f, d)$ of N with respect to a flow f is the maximal subgraph of the residual flow network $R(N, f)$ such that $L(N, f, d)$ contains both s and t, all the directed paths of $L(N, f, d)$ from s to t have the same length d, and each arc of $L(N, f, d)$ is in a directed path of $L(N, f, d)$ from s to t. The *depth* of $L(N, f, d)$ is equal to d. Figure 1(b) shows a layered network with depth 3. $L(N, f, d)$ is the *shallowest* layered network of N with respect to flow f if all the layered networks of N with respect to f have depth at least d (see Figure 1(b)). A *blocking flow* g of $L(N, f, d)$ is one that saturates at least one arc of every directed path from s to t, i.e., every directed path from s to t has an arc e such that $g(e) = u(e)$(see Figure 1(c)). Notice that a blocking flow of a flow network may not be a maximum flow of the network. For example, the layered flow network of Figure 1(b) admits a maximum flow of magnitude 3 and also admits a blocking flow of magnitude 2 (which is shown in Figure 1(c)). The concept of layered flow networks allows us to introduce more flow in a flow network by pushing it through the augmenting paths with smallest lengths.

Let p be a directed path of $R(N, f)$ from s to t with least cost among all the directed paths of $R(N, f)$ from s to t. Let $c(N, f)$ be the cost of p. The *admissible* flow network $A(N, f)$ of N with respect to a flow f is the maximal subgraph of the residual flow network $R(N, f)$ such that $A(N, f)$ contains both s and t, all the directed paths of $A(N, f)$ from s to t have the same cost, equal to $c(N, f)$, and each arc of $A(N, f)$ is in a directed path of $A(N, f)$ from s to t(see Figure 1(b)). Hence the concept of an admissible flow network is similar to that of a layered flow network except that we consider the costs of directed paths instead of their lengths. For the flow network of Figure 1(a), the admissible and layered flow networks are the same (shown in Figure 1(b)), but for a general flow network, they may be different. The concept of admissible flow networks allows us to introduce more flow in a flow network by pushing it through the augmenting paths with least costs.

Lemma 1. *Let $R(A(N, f), g)$ be the residual flow network of $A(N, f)$ with respect to a flow g. All the directed paths of $R(A(N, f), g)$ from s to t have the same cost, namely, $c(N, f)$.*

Proof. Our proof is based on the proof of Lemma 8.4 of [21](chapter 8, pg. 110).

Let p be a directed path of $R(A(N,f),g)$ from s to t, and v be a node of p. Let $c(v,p,g)$ denote the cost of the subpath from s to v of p. It follows from the definition of $A(N,f)$ that in $A(N,f)$, all the paths from s to a node w have the same cost, which we denote by $c(w)$, and that $c(t) = c(N,f)$. It is also easy to see that if (u,w) is an arc of $A(N,f)$, then $c(w) - c(u) = c(u,w)$ (recall that $c(u,w)$ denotes the cost of the arc (u,w)).

We claim that if p is a directed path of $R(A(N,f),g)$ from s to t and v is a node of p, then $c(v,p,g) = c(v)$, and if (u,v) is an arc of p, then $c(u,v) = c(v) - c(u)$. This will give immediately that all the directed paths of $R(A(N,f),g)$ from s to t have the same cost, which is equal to $c(t) = c(N,f)$. We prove our claim using induction over the magnitude of flow g.

If $|g| = 0$, our claim is trivially true. Let h be a flow in N with magnitude $|g| - 1$ such that pushing a flow with unit magnitude through a directed path p' of $R(A(N,f),h)$ from s to t gives us flow g in $A(N,f)$. From the inductive hypothesis, it follows that if (u,v) is an arc of p', then $c(u,v) = c(v) - c(u)$. The only arcs of $R(A(N,f),g)$ that are not arcs of $R(A(N,f),h)$ also are all of the form (y,x) where (x,y) is an arc of p', and $c(y,x) = -c(x,y) = c(x) - c(y)$.

Let p be a directed path of $R(A(N,f),g)$ from s to t. Let v be a node of p. Let (u,v) be an arc of p. We first show that $c(u,v) = c(v) - c(u)$, and then using this show that $c(v,p,g) = c(v)$.

If (u,v) was an arc of $R(A(N,f),h)$ also then from the inductive hypothesis, $c(u,v) = c(v) - c(u)$. However, if (u,v) was not an arc of $R(A(N,f),h)$ then also from the above discussion it follows that $c(u,v) = c(v) - c(u)$.

A simple argument based on induction over the length of the subpath from s to v of p shows that $c(v,p,g) = c(v)$. □

2.2 Three Basic Flow Algorithms

Research in the area of network flows has a rich tradition (see [1] for an extensive survey). A number of algorithms have been proposed for finding maximum flows and minimum cost flows in flow networks. In this section, we review three well-known basic flow algorithms, which we call *AugPath*, *BlockFlow* and *PrimDua*, respectively.

Algorithm *AugPath* (see Figure 2) is the classic method developed by Ford and Fulkerson [10] that finds a maximum flow in a single-source single-sink flow network by successively pushing flow through augmenting paths.

Lemma 2 [10]. *Let N be a single-source single-sink flow network with n nodes and m arcs. Algorithm* AugPath *computes a maximum flow for N in time $O(\phi \cdot m)$, where ϕ is the maximum flow magnitude.*

Figure 3 shows Algorithm *BlockFlow*, which computes a maximum flow in a single-source single-sink flow network N. Each phase of Algorithm *BlockFlow* computes a blocking flow in the shallowest layered network of N with respect to the flow already computed. See [15] for further details about this algorithm.

Algorithm *Augpath(N)*: /*N is a flow network with a single source s and a single sink t*/
begin
    $R_1 \leftarrow N$; $F_1 \leftarrow \mathbf{0}$; $i \leftarrow 1$;
    while N has an augmenting path with respect to flow F_i
    begin (Phase i)
        Using R_i, find an augmenting path p_i of N with respect to flow F_i;
        Push a non-zero flow f_i from s to t through p_i
        Let F_{i+1} be the total flow in N after pushing flow f_i in R_i;
        Let R_{i+1} be the residual flow network of N with respect to flow F_{i+1};
        $i \leftarrow i + 1$;
    end
end

Fig. 2. Algorithm *AugPath*.

Algorithm *BlockFlow(N)*: /*N is a single-source single-sink flow network */
begin
    $L_1 \leftarrow N$; $i \leftarrow 1$;
    while L_i admits a non-zero blocking flow
    begin (Phase i)
        Find a blocking flow f_i of L_i;
        Let F_{i+1} be the total flow in N after pushing flow f_i in L_i;
        Let L_{i+1} be the shallowest layered network of N with respect to flow F_{i+1};
        $i \leftarrow i + 1$;
    end
end

Fig. 3. Algorithm *BlockFlow*.

Lemma 3 [15] (chapter IV, Section 9). *The following properties hold for each phase i of Algorithm* BlockFlow*:*

1. *The magnitude of flow in N is increased by at least one, i.e., $|F_i| \geq |F_{i-1}|+1$.*
2. *The depth of L_i is strictly greater than the depth of L_{i-1}.*
3. *Phase i can be executed in time $O(m \log n)$, where n and m denote the number of nodes and arcs, respectively, of N.*

Algorithm *PrimDua*, shown in Figure 4, reduces the problem of computing a minimum cost flow in a single-source single-sink flow network N to the problem of computing maximum flows in a sequence of intermediate flow-networks. Each of these maximum flows can be computed using any algorithm for computing maximum flows. Algorithm *PrimDua* (short for primal-dual) was developed first by Ford and Fulkerson [9, 10]. This algorithm computes f in a sequence of stages, where each stage computes a maximum flow in the admissible network of N with respect to the flow already pushed in it. See [1, 9, 10] for details.

Algorithm *PrimDua(N)*: /**N* is a single-source single-sink flow network*/
begin
Let A_1 be the admissible network of N with respect to a zero-flow;
$i \leftarrow 1$;
while A_i admits a non-zero maximum flow
begin (Stage i)
Find a maximum flow f_i of A_i;
Let F_{i+1} be the total flow in N after pushing flow f_i in A_i;
Let A_{i+1} be the admissible network of N with respect to flow F_{i+1};
$i \leftarrow i + 1$;
end
end

Fig. 4. Algorithm *PrimDua*.

Notice that from the definition of an admissible flow network, all the directed paths of A_i from s to t have the same cost, which we denote by c_i (if A_i has no directed path from s to t, then c_i is ∞). Lemma 4 follows directly from the discussion on the primal-dual algorithm in [1].

Lemma 4. *For each stage i of Algorithm* PrimDua, $c_{i+1} \geq c_i + 1$.

The following corollary is immediate.

Corollary 5. *If N is a flow network with positive arc costs, then for each stage i of Algorithm* PrimDua, $c_i \geq i$.

3 A New Minimum Cost Flow Algorithm

Let N be a single-source single-sink flow network with n nodes, m arcs, and positive arc costs. Let s and t be the source and sink, respectively, of N. Theorem 10, which is the main result of this section, shows that a minimum cost flow in N can be computed in $O(\chi^{3/4} m\sqrt{\log n})$ time, where χ is the cost of the flow. This computation is done using Algorithm *AugBlock* shown in Figure 5. Algorithm *AugBlock* is a variation of Algorithm *PrimDua* described in Section 2. The main feature of Algorithm *AugBlock* is that, in each stage i, it computes f_i by running algorithms *AugPath* and *BlockFlow* in parallel. Flow f_i is the flow computed by the algorithm terminating first.

Let χ be the cost of the minimum cost flow f computed by Algorithm *AugBlock*. Let ϕ_i and χ_i be the magnitude and cost, respectively, of flow f_i. Let $S = \{f_i | 1 \leq \phi_i \leq \sqrt{\chi}\log n\}$. Let $B = \{f_i | \phi_i > \sqrt{\chi}\log n\}$. We call the stages of S as the *light* stages and the stages of B as the *heavy* stages. Let $\phi_S = \sum_{f_i \in S} \phi_i$, and $\phi_B = \sum_{f_i \in B} \phi_i$. Let $\chi_S = \sum_{f_i \in S} \chi_i$, and $\chi_B = \sum_{f_i \in B} \chi_i$. Therefore, ϕ_S and χ_S (ϕ_B and χ_B) denote the total magnitude and total cost, respectively, of the flow pushed in the light (heavy) stages.

Algorithm *AugBlock(N)*: /*N is a single-source single-sink flow network with positive arc costs*/
begin
Let A_1 be the admissible network of N with respect to a zero-flow;
$i \leftarrow 1$;
while A_i admits a non-zero maximum flow
begin (Stage i)
To find a maximum flow of A_i, run *AugPath*(N_i) and *BlockFlow*(N_i) in parallel. Go to the next step when one of them terminates after computing a maximum flow f_i;
Let F_{i+1} be the total flow in N after pushing flow f_i in A_i;
Let A_{i+1} be the admissible network of N with respect to flow F_{i+1};
$i \leftarrow i+1$;
end
end

Fig. 5. Algorithm *AugBlock*.

Lemma 6. *The total flow pushed in the light stages of Algorithm* AugBlock *is* $O(\chi^{3/4}\sqrt{\log n})$, *i.e.,* $\phi_S = O(\chi^{3/4}\sqrt{\log n})$.

Proof. Suppose set S consists of flows $f_{i_1}, f_{i_2}, \ldots, f_{i_l}$, where $1 \leq i_1 < i_2 \ldots < i_l$. From the definition of ϕ_S, $\phi_S = \sum_{1 \leq j \leq l} \phi_{i_j}$, and each ϕ_{i_j} is at most $\sqrt{\chi}\log n$. From the definition of flow network A_{i_j}, all the directed paths of A_{i_j} from s to t have the same cost c_{i_j}. Therefore, $\chi_{i_j} = c_{i_j} \cdot \phi_{i_j}$. Since N is a flow network with positive arc costs, from Corollary 5, it follows that $\chi_{i_j} = c_{i_j} \cdot \phi_{i_j} \geq i_j\phi_{i_j} \geq j\phi_{i_j}$. Therefore, $\chi \geq \chi_S = \sum_{1 \leq j \leq l} \chi_{i_j} \geq \sum_{1 \leq j \leq l} j\phi_{i_j}$. In other words, $\chi \geq \phi_{i_1} + 2\phi_{i_2} + \cdots + l\phi_{i_l}$. Since ϕ_{i_j} is at most $\sqrt{\chi}\log n$, it follows that for a given χ, ϕ_S is maximum when the values of l and the ϕ_{i_j}'s are such that $\phi_{i_1} = \phi_{i_2} = \cdots = \phi_{i_l} = \sqrt{\chi}\log n$ and $\chi = \sum_{1 \leq j \leq l} j\sqrt{\chi}\log n = (l(l+1)/2)\sqrt{\chi}\log n$. From some mathematical manipulation, it follows that when ϕ_S is maximum, l is such that $l^2 + l = 2\sqrt{\chi}/\log n$, and hence, if we denote by l^*, the value of l for which ϕ_S is maximum, we have that $l^* = O(\chi^{1/4}/\sqrt{\log n})$. Therefore, $\phi_S \leq \sum_{1 \leq j \leq l^*} \sqrt{\chi}\log n = l^*\sqrt{\chi}\log n = O(\chi^{1/4}/\sqrt{\log n})\sqrt{\chi}\log n = O(\chi^{3/4}\sqrt{\log n})$. □

Lemma 7. *Let* $k(\alpha, \beta)$, *where* $0 < \alpha < 1$ *and* $\beta > 0$, *be the number of stages of Algorithm* AugBlock *for which* $\phi_i > \chi^\alpha\beta$. *Then we have*

$$k(\alpha, \beta) < \sqrt{2/\beta}\chi^{(1-\alpha)/2}$$

Proof. Let $f_{i_1}, f_{i_2}, \ldots, f_{i_l}$, where $1 \leq i_1 < i_2 \ldots < i_l$ and $l = k(\alpha, \beta)$, be the set of all the flows for which $\phi_{i_j} > \chi^\alpha\beta$. From the definition of flow network A_{i_j}, all the directed paths of A_{i_j} from s to t have the same cost c_{i_j}. Therefore, $\chi_{i_j} = c_{i_j} \cdot \phi_{i_j}$. Since N is a flow network with positive arc costs, from Corollary 5, it follows that $\chi_{i_j} = c_{i_j} \cdot \phi_{i_j} \geq i_j\phi_{i_j} \geq j\phi_{i_j}$. We have that,

$\chi \geq \sum_{1 \leq j \leq l} \chi_{i_j} = \sum_{1 \leq j \leq l} j\phi_{i_j} > \sum_{1 \leq j \leq l} j\chi^{\alpha}\beta = (l(l+1)/2)\chi^{\alpha}\beta$. From some mathematical manipulation, we get that $(2/\beta)\chi^{1-\alpha} > l^2 + l \geq l^2$. Hence, $k(\alpha, \beta) = l < \sqrt{2/\beta}\chi^{(1-\alpha)/2}$. □

Since $|B| = k(1/2, \log n)$, by Lemma 7 we have,

Corollary 8. *The total number of heavy stages of Algorithm* AugBlock *is less than* $\sqrt{2/\log n}\chi^{1/4}$, *i.e.,* $|B| < \sqrt{2/\log n}\chi^{1/4}$.

Lemma 9 yields an upper bound on the time used by Algorithm *BlockFlow* for computing the maximum flow f_i of A_i.

Lemma 9. *In Stage i of Algorithm* AugBlock, *Algorithm* BlockFlow *computes the maximum flow* f_i *of* A_i *in time* $O(\sqrt{\chi}m_i \log n_i)$, *where* n_i *and* m_i *are the number of nodes and arcs, respectively, of* A_i.

Proof. Let r be the number of phases used by Algorithm *BlockFlow* to compute f_i. We now show that r is at most $3\sqrt{\chi}$. From Lemma 3 (Property 3) it will then follow that Algorithm *BlockFlow* computes f_i in time $O(\sqrt{\chi}m_i \log n_i)$.

We consider two cases:

Case 1. $\phi_i < \sqrt{\chi}$: From Lemma 3 (Property 1), each phase of Algorithm *BlockFlow* increases flow in A_i by at least 1. Therefore, $r \leq \phi_i < \sqrt{\chi}$.

Case 2. $\phi_i \geq \sqrt{\chi}$: Let l be the phase that increases the flow in A_i to at least $\phi_i - \sqrt{\chi}$. From Lemma 3 (Property 1), each phase of Algorithm *BlockFlow* increases the flow in A_i by at least 1. Therefore, the number of phases after phase l is at most $\sqrt{\chi}$. Hence, if we show that $l \leq 2\sqrt{\chi}$, we are done. From Lemma 3 (Property 2), each phase of Algorithm *BlockFlow* increases the depth of the shallowest layered network used for pushing more flow by at least 1. Therefore, if we can show that the depth d of the shallowest layered network L_l of A_i used for phase l is at most $2\sqrt{\chi}$, it will follow that $l \leq 2\sqrt{\chi}$. Now we show that $d < 2\sqrt{\chi}$.

Our proof follows this approach: We first compute the cost of the new flow pushed in A_i in the phases $l, l+1, \ldots, r$. Using the costs of this new flow, we compute the cost $c(F_{i+1})$ of the total flow in N at the end of stage i. Then we show that because $c(F_{i+1})$ is at most χ, d is less than $2\sqrt{\chi}$. We now give the details of the proof.

Let g be the total flow pushed in A_i in phases $1, 2, \ldots, l-1$. Let $R(A_i, g)$ be the residual flow network of A_i with respect to flow g. Let h be the total flow pushed in $R(A_i, g)$ in phases $l, l+1, \ldots, r$. Let μ be the magnitude of flow h. From the definition of phase l, we have that $\mu \geq \sqrt{\chi}$. Flow h can be decomposed into μ flows $h_1, h_2, \ldots, h_\mu$ where each h_j has unit magnitude. From Lemma 1 and Corollary 5, it follows that the cost of each h_j is greater than 0. Also, since the depth of L_l is d, we have that the length of each directed path of $R(A_i, g)$ from s to t is at least d, and therefore, each h_j uses at least d arcs of $R(A_i, g)$.

Let W_j and Z_j be the sets consisting of the arcs of h_j with positive and negative costs, respectively. Let $c(W_j) = \sum_{e \in W_j} c(e)$, i.e., the total cost of the arcs of h_j with positive costs. Similarly, let $c(Z_j) = \sum_{e \in Z_j} c(e)$. We now show that $c(W_j) > d/2$. We have shown earlier that h_j uses at least d arcs of $R(A_i, g)$. Therefore, $|W_j| + |Z_j| \geq d$. Since each arc of N and therefore, each arc of $R(A_i, g)$ has non-zero cost, we have that $c(W_j) \geq |W_j|$ and $|c(Z_j)| = -c(Z_j) \geq |Z_j|$. Hence, $c(W_j) + |c(Z_j)| \geq |W_j| + |Z_j| \geq d$. We have shown earlier that the cost of h_j is greater than 0. Since the cost of h_j is equal to $c(W_j) + c(Z_j)$, we have that $c(W_j) + c(Z_j) > 0$. It follows that $c(W_j) > -c(Z_j) = |c(Z_j)|$. Therefore, $2c(W_j) > c(W_j) + |c(Z_j)| \geq d$. Hence, $c(W_j) > d/2$.

Recall that F_i denotes the total flow pushed in N by Algorithm *AugBlock* (since the beginning of its execution) at the end of Stage $i-1$ (See Figure 5). Let K be the set of the arcs of $R(A_i, g)$ with negative costs. Also recall that h is the total flow pushed in $R(A_i, g)$ in phases $l, l+1, \ldots, r$. We have that

$$\begin{aligned}
c(F_{i+1}) &= c(F_i) + c(g) + \sum_{1 \leq j \leq \mu} c(h_j) \\
&\geq c(F_i) + c(g) + \sum_{1 \leq j \leq \mu} (c(W_j) + c(Z_j)) \\
&\geq \sum_{1 \leq j \leq \mu} c(W_j) + c(F_i) + c(g) + \sum_{1 \leq j \leq \mu} c(Z_j) \\
&\geq \sum_{1 \leq j \leq \mu} c(W_j) + c(F_i) + c(g) + \sum_{1 \leq j \leq \mu} \sum_{e \in Z_j} c(e) \\
&= \sum_{1 \leq j \leq \mu} c(W_j) + c(F_i) + c(g) + \sum_{e \in K} h(e)c(e) \qquad (1)
\end{aligned}$$

Let $e = (y, x)$ be an arc of $R(A_i, g)$ with negative cost. Arc e corresponds to an arc $e^* = (x, y)$ of N such that $u(e) = F_i(e^*) + g(e^*)$ and $c(e) = -c(e^*)$. The cost of the flow in e^* due to flows F_i and g is $F_i(e^*)c(e^*) + g(e^*)c(e^*) = (F_i(e^*) + g(e^*))c(e^*) = -u(e)c(e) \geq -h(e)c(e)$. Summing both the sides of the inequality over all the arcs of $R(A_i, g)$ with negative costs, it follows that $c(F_i) + c(g) \geq -\sum_{e \in K} h(e)c(e)$, and hence, $c(F_i) + c(g) + \sum_{e \in K} h(e)c(e) \geq 0$. Informally speaking, this means that the (negative) costs of the flows due to flow h in the arcs of $R(A_i, g)$ with negative costs is "accounted for" by the (positive) cost of the flows due to F_i and g in the arcs of N.

We have shown earlier that $c(W_j) > d/2$. Hence, $\sum_{1 \leq j \leq \mu} c(W_j) > \mu d/2$. We have also shown that $\mu \geq \sqrt{\chi}$. Therefore, $\sum_{1 \leq j \leq \mu} c(W_j) > \sqrt{\chi} d/2$.

From Eq. 1 it follows now that $c(F_{i+1}) > \sqrt{\chi} d/2$. Since $\chi \geq c(F_{i+1})$, we have that $\chi \geq c(F_{i+1}) > \sqrt{\chi} d/2$, and therefore, $d < 2\sqrt{\chi}$. □

The following theorem summarizes the main result of this section.

Theorem 10. *Let N be a single-source single-sink flow network with n nodes, m arcs, and positive arc costs. Algorithm* AugBlock *computes a minimum cost flow f of N in time $O(\chi^{3/4} m \sqrt{\log n})$, where χ is the cost of f.*

Proof. The proof is based on the following idea: We can bound the running-time of a stage of Algorithm *AugBlock* by the running-time (for that stage) of either of Algorithm *AugPath* and Algorithm *BlockFlow*. We can do this because the flow computed in a stage is the one given by the algorithm that terminates first. Suppose we bound the running time for a light stage by the running-time of Algorithm *AugPath*, and for a heavy stage by the running-time of Algorithm *BlockFlow*. From Lemma 6, the total flow computed in the light stages is $O(\chi^{3/4}\sqrt{\log n})$. Therefore, from Lemma 2, it follows that the total running-time of light stages is bounded by $O((\chi^{3/4}\sqrt{\log n})m)$. As for the heavy stages, from Corollary 8, we have that the number of heavy stages is at most $\sqrt{2/\log n}\chi^{1/4}$. Since from Lemma 9, Algorithm *BlockFlow* takes time $O(\sqrt{\chi}m\log n)$ for computing a flow in a stage of Algorithm *AugBlock*, the total running time of the heavy stages is bounded by $\sqrt{2/\log n}\chi^{1/4} \cdot O(\sqrt{\chi}m\log n) = O(\chi^{3/4}m\sqrt{\log n})$. This gives the total time-complexity of $O(\chi^{3/4}m\sqrt{\log n})$ for Algorithm *AugBlock*.

We now give the details of the proof.

Let r be the total number of stages used by *Algorithm AugBlock* to compute the optimal flow f. Consider stage i, where $1 \leq i \leq r$. Stage i increases the amount of flow in N by at least 1. Therefore, $\phi_i \geq 1$. Let n_i and m_i be the number of nodes and arcs, respectively, of A_i. Clearly, $n_i \leq n$ and $m_i \leq 2m$ (the factor of 2 appears because there may be some backward arcs in A_i). Let T_i, T_{S_i}, and T_{B_i} be the time taken by algorithms *AugBlock*, *AugPath* and *BlockFlow*, respectively, for computing f_i in A_i. Flow f_i is the one computed by the algorithm that terminates first. Therefore, $T_i = \min(T_{S_i}, T_{B_i})$. Hence, $T_i = T_{S_i}$ if $f_i \in S$, and $T_i = T_{B_i}$ if $f_i \in B$ are valid upper bounds for T_i.

Let T be the total time taken by *Algorithm AugBlock* to compute f. We have:

$$\begin{aligned} T &= \sum_{1 \leq i \leq r} T_i \\ &= \sum_{f_i \in S} T_{S_i} + \sum_{f_i \in B} T_{B_i} \end{aligned}$$

From Lemma 2, $T_{S_i} = O(\phi_i \cdot m_i)$, and from Lemma 9, $T_{B_i} = O(\sqrt{\chi}m_i \log n_i)$. Therefore,

$$\begin{aligned} T &= \sum_{f_i \in S} O(\phi_i \cdot m_i) + \sum_{f_i \in B} O(\sqrt{\chi}m_i \log n_i) \\ &= O(m) \sum_{f_i \in S} \phi_i + (\sum_{f_i \in B} 1) \cdot O(\sqrt{\chi}m \log n) \\ &= \phi_S \cdot O(m) + |B| \cdot O(\sqrt{\chi}m \log n) \end{aligned}$$

From Lemma 6, we have that $\phi_S = O(\chi^{3/4}\sqrt{\log n})$, and from Corollary 8, we have that $|B| < \sqrt{2/\log n}\chi^{1/4}$. Therefore, $T = O(\chi^{3/4}\sqrt{\log n} \cdot m) + O(\sqrt{2/\log n}\chi^{1/4} \cdot \sqrt{\chi}m\log n) = O(\chi^{3/4}m\sqrt{\log n})$. □

4 Faster Bend Minimization

Orthogonal drawings of graphs, where the edges are drawn as polygonal chains with alternating horizontal and vertical segments, are widely used in visualization applications, and they have been extensively studied (see, e.g., [2, 4, 6, 8, 11, 12, 13, 14, 16, 18, 20, 22]).

An important quality measure for orthogonal drawings is the total number of bends along the edges. Bend minimization is the core of a practical drawing technique [5] for general graphs, called Giotto, which performs a preliminary planarization followed by bend-minimization. Giotto has has been widely used in software and data visualization systems [5, 7]. A sample drawing is shown in Figure 6(d)). The extensive experiments conducted by Di Battista et al. [4] on general-purpose orthogonal drawing algorithms, which use 11,582 graphs derived from "real-life" software engineering and database applications, show that Giotto outperforms all other known orthogonal drawing algorithms in quality measures such as area, number of bends, and aspect-ratio. However, the bottleneck in the running time of Giotto is the execution of the bend minimization step.

Let G be an embedded planar graph with maximum degree 4. As shown in [19], a drawing of G with the minimum number of bends can be computed by an algorithm consisting of the following two main phases:

1. computation of an *orthogonal shape* for G, where only the bends and the angles of the orthogonal drawing are defined;
2. assignment of integer lengths to the segments of the orthogonal shape.

Phase 1 uses a transformation into a network flow problem (Figure 6(a–c)), where each unit of flow is associated with a right angle in the orthogonal drawing. Hence, angles are viewed as a commodity that is produced by the vertices, transported across faces by the edges through their bends, and eventually consumed by the faces. It is easier to to describe this flow problem on a network with lower bounds on flows in arcs (in addition to arc-capacities) and supplies/demands on the sources and sinks. From the embedded graph G we construct such a flow network N as follows. The nodes of network N are the vertices and faces of G. Let $\deg(f)$ denote the number of edges of the circuit bounding face f. Each vertex v supplies $\sigma(v) = 4$ units of flow, and each face f consumes $\tau(f)$ units of flow, where

$$\tau(f) = \begin{cases} 2\deg(f) - 4 \text{ if } f \text{ is an internal face} \\ 2\deg(f) + 4 \text{ if } f \text{ is the external face} \end{cases}$$

By Euler's formula, $\sum_v \sigma(v) = \sum_f \tau(f)$, i.e., the total supply is equal to the total consumption.

Network N has two types of arcs:

- arcs of the type (v, f), where f is a face incident on vertex v; the flow in (v, f) represents the angle at vertex v in face f, and has lower bound 1, capacity 4, and cost 0;

- arcs of the type (f, g), where face f shares an edge e with face g; the flow in (f, g) represents the number of bends along edge e with the right angle inside face f, and has lower bound 0, capacity $+\infty$, and cost 1.

The conservation of flow at the vertices expresses the fact that the sum of the angles around a vertex is equal to 2π. The conservation of flow at the faces expresses that fact that the sum of the angles at the vertices and bends of an internal face is equal to $\pi(p-2)$, where p is the number of such angles. For the external face, the above sum is equal to $\pi(p+2)$.

It can be shown that every flow ϕ in network N corresponds to an admissible orthogonal shape for graph G, whose number of bends is equal to the cost of flow ϕ. Hence, an orthogonal shape for G with the minimum number of bends can be computed from a minimum cost flow in G.

Phase 2 uses a simple compaction strategy derived from VLSI layout, where the lengths of the horizontal and vertical segments are computed independently after a preliminary refinement of the orthogonal shape that decomposes each face into into rectangles.

The best previous time bound for bend minimization is $O(n^2 \log n)$ [19], which is achieved with standard flow-augmentation techniques. Our new minimum cost flow method yields a faster bend minimization algorithm:

Theorem 11. *Let G be an embedded planar graph with n vertices and maximum vertex degree 4. An orthogonal drawing of G with the minimum number of bends can be computed in $O(n^{7/4}\sqrt{\log n})$ time.*

Sketch of Proof: Modify the flow network N associated with G to a get a new single-source single-sink flow network N' with positive arc costs by

- assigning unit cost to all the arcs of the type (v, f),
- adding two new nodes s and t, designating them the source and sink, respectively,
- for every vertex v, adding an arc (s, v) with capacity $\sigma(v)$ and cost 1, and
- for every face f, adding an arc (f, t) with capacity $\tau(f)$ and cost 1.

It is easy to see a flow f is of minimum cost in N if and only if it is of minimum cost in N'. Network N' has $O(n)$ nodes and arcs. Also, since the minimum number of bends for G is $O(n)$ (see, e.g., [20]), the minimum cost of the flow in N' is $O(n)$. Hence, we can use Algorithm *AugBlock* to compute a minimum cost flow for N' in time $O(n^{7/4}\sqrt{\log n})$ (see Theorem 10).

□

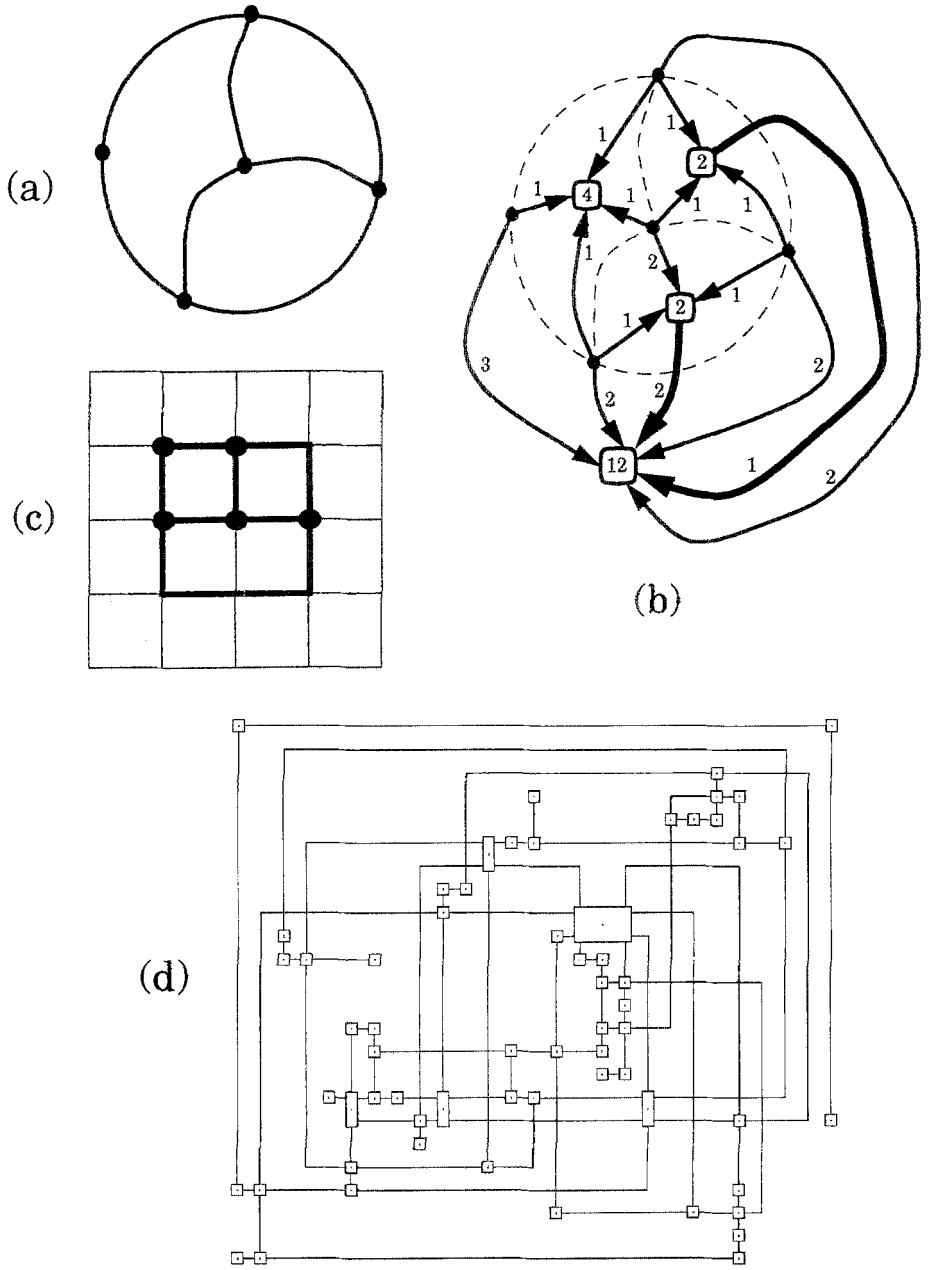

Fig. 6. **(a)** Embedded graph G. **(b)** Minimum cost flow in network N associated with G: the flow is shown next to each arc; arcs with zero flow are omitted; arcs with unit cost are drawn with thick lines; a face f is represented by a box labeled with $\tau(f)$. **(c)** Planar orthogonal grid drawing of G with minimum number of bends. **(d)** Orthogonal grid drawing of a nonplanar graph produced by Giotto.

References

1. R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall, Englewood Cliffs, NJ, 1993.
2. T. Biedl and G. Kant. A better heuristic for orthogonal graph drawings. In *Proc. 2nd Annu. European Sympos. Algorithms (ESA '94)*, volume 855 of *Lecture Notes in Computer Science*, pages 24–35. Springer-Verlag, 1994.
3. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. Algorithms for drawing graphs: an annotated bibliography. *Comput. Geom. Theory Appl.*, 4:235–282, 1994.
4. G. Di Battista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari, and F. Vargiu. An experimental comparison of three graph drawing algorithms. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 306–315, 1995.
5. G. Di Battista, A. Giammarco, G. Santucci, and R. Tamassia. The architecture of Diagram Server. In *Proc. IEEE Workshop on Visual Languages (VL'90)*, pages 60–65, 1990.
6. G. Di Battista, G. Liotta, and F. Vargiu. Spirality of orthogonal representations and optimal drawings of series-parallel graphs and 3-planar graphs. In *Proc. Workshop Algorithms Data Struct.*, volume 709 of *Lecture Notes in Computer Science*, pages 151–162. Springer-Verlag, 1993.
7. G. Di Battista, G. Liotta, and F. Vargiu. Diagram Server. *J. Visual Lang. Comput.*, 6(3):275–298, 1995. (special issue on Graph Visualization, edited by I. F. Cruz and P. Eades).
8. S. Even and G. Granot. Grid layouts of block diagrams — bounding the number of bends in each connection. In R. Tamassia and I. G. Tollis, editors, *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes in Computer Science*, pages 64–75. Springer-Verlag, 1995.
9. L.R. Ford and D.R. Fulkerson. A primal-dual algorithm for the capacitated hitchcock problem. *Naval Research Logistics Quarterly*, 4:47–54, 1957.
10. L.R. Ford and D.R. Fulkerson. *Flows in Networks.* Princeton University Press, Princeton, NJ, 1962.
11. U. Fößmeier and M. Kaufmann. On bend-minimum orthogonal upward drawing of directed planar graphs. In R. Tamassia and I. G. Tollis, editors, *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes in Computer Science*, pages 52–63. Springer-Verlag, 1995.
12. A. Garg and R. Tamassia. On the computational complexity of upward and rectilinear planarity testing. In R. Tamassia and I. G. Tollis, editors, *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes in Computer Science*, pages 286–297. Springer-Verlag, 1995.
13. G. Kant. Drawing planar graphs using the canonical ordering. *Algorithmica*, 16:4–32, 1996. (special issue on Graph Drawing, edited by G. Di Battista and R. Tamassia).
14. Y. Liu, P. Marchioro, R. Petreschi, and B. Simeone. Theoretical results on at most 1-bend embeddability of graphs. Technical report, Dipartimento di Statistica, Univ. di Roma "La Sapienza", 1990.
15. K. Mehlhorn. *Graph Algorithms and NP-Completeness*, volume 2 of *Data Structures and Algorithms.* Springer-Verlag, Heidelberg, West Germany, 1984.
16. A. Papakostas and I. G. Tollis. Improved algorithms and bounds for orthogonal drawings. In R. Tamassia and I. G. Tollis, editors, *Graph Drawing (Proc. GD '94)*, volume 894 of *Lecture Notes in Computer Science*, pages 40–51. Springer-Verlag, 1995.

17. D.D. Sleator. *An $O(nm \log n)$ Algorithm for Maximum Network Flow.* PhD thesis, Dept. Comput. Sci., Stanford Univ., Palo Alto, California, 1980.
18. J. A. Storer. On minimal node-cost planar embeddings. *Networks*, 14:181–212, 1984.
19. R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM J. Comput.*, 16(3):421–444, 1987.
20. R. Tamassia and I. G. Tollis. Planar grid embedding in linear time. *IEEE Trans. Circuits Syst.*, CAS-36(9):1230–1234, 1989.
21. R. E. Tarjan. *Data Structures and Network Algorithms*, volume 44 of *CBMS-NSF Regional Conference Series in Applied Mathematics.* Society for Industrial Applied Mathematics, 1983.
22. L. Valiant. Universality considerations in VLSI circuits. *IEEE Trans. Comput.*, C-30(2):135–140, 1981.