Algorithms for Placing Monitors in a Flow Network

(Preliminary Version)

Francis Chin[1,⋆], Marek Chrobak[2,⋆⋆], and Li Yan[2,⋆⋆]

[1] Department of Computer Science, The University of Hong Kong, Pokfulam, Hong Kong

[2] Department of Computer Science, University of California, Riverside, CA 92521

Abstract. In the Flow Edge-Monitor Problem, we are given an undirected graph $G = (V, E)$, an integer $k > 0$ and some unknown circulation ψ on G. We want to find a set of k edges in G, so that if we place k monitors on those edges to measure the flow along them, the total number of edges for which the flow can be uniquely determined is maximized. In this paper, we first show that the Flow Edge-Monitor Problem is NP-hard, and then we give two approximation algorithms: a 3-approximation algorithm with running time $O((m+n)^2)$ and a 2-approximation algorithm with running time $O((m+n)^3)$, where $n = |V|$ and $m = |E|$.

1 Introduction

We study the *Flow Edge-Monitor Problem* (FlowMntrs, for short), where the objective is to find k edges in an undirected graph $G = (V, E)$ with an unknown circulation ψ, so that if we place k flow monitors on these edges to measure the flow along them, we will maximize the total number of edges for which the value and direction of ψ is uniquely determined by the flow conservation property. Intuitively, the objective is to maximize the number of bridge edges in the subgraph induced by edges not covered by monitors. (For a more rigorous definition of the problem, see Section 2.)

Consider, for example, the graph and the monitors shown in Figure 1. In this example we have $k = 4$ monitors represented by rectangles attached to edges, with measured flow values and directions shown inside. Thus we have $\psi(2,3) = 4$, $\psi(3,8) = 2$, $\psi(6,4) = 7$ and $\psi(1,2) = 1$. From the flow conservation property, we can then determine that $\psi(3,5) = 2$, $\psi(8,6) = 2$, $\psi(7,5) = 3$ and $\psi(5,6) = 5$. Thus with 4 monitors we can determine flow values on 8 edges.

Our results. We first show that the FlowMntrs problem is NP-hard. Next, we study polynomial-time approximation algorithms. We introduce an algorithm called σ-Greedy that, in each step, places up to σ monitors in such

⋆ Research supported in parts by grant (HKU 7113/07E).

⋆⋆ Research supported by NSF Grant CCF-0729071.

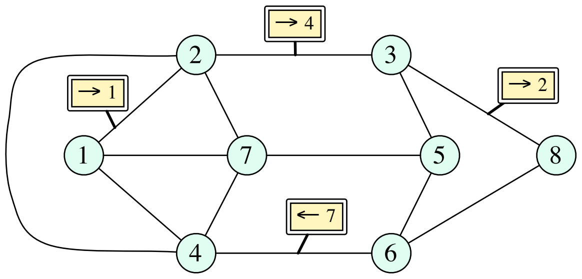

Fig. 1. A graph with 4 monitors

a way that the number of edges with known flow is maximized. We then prove that 1-Greedy is a 3-approximation algorithm and that 2-Greedy is a 2-approximation algorithm. The running times of these two algorithms are $O((m + n)^2)$ and $O((m + n)^3)$, respectively, where $n = |V|$ and $m = |E|$. In both cases, our analysis is tight. In fact, our approximation results are stronger, as they apply to the weighted case, where the input graph has weights on edges, and the objective is to maximize the total weight of the edges with known flow.

Related work. A closely related problem was studied by Gu and Jia [4] who considered a traffic flow network with directed edges. They observed that $m - n + 1$ monitors are necessary to determine the flow on all edges of a strongly connected graph, and that this bound can be achieved by placing flow monitors on edges in the complement of a spanning tree. (The same bound applies to connected undirected graphs.) Khuller et al. [5] studied an optimization problem where pressure meters may be placed on nodes of a flow network. An edge whose both endpoints have a pressure meter will have the flow determined using the pressure difference, and other edges may have the flow determined via flow conservation property. The goal is to compute the minimum number of meters needed to determine the flow on every edge in the network. They showed that this problem is NP-hard and MAX-SNP-hard, and that a local-search based algorithm achieves 2-approximation. For planar graphs, they have a polynomial-time approximation scheme. The model in [5] differs from ours in that it assumes that the flow satisfies Kirchhoff's current and voltage laws, while we only assume the current law (that is, the flow preservation property). This distinction is reflected in different choices of "meters": vertex meters in [5] and edge monitors in our paper. Recall that, as explained above, minimizing the number of *edge monitors* needed to determine the flow on all edges is trivial, providing a further justification for our choice of the objective function.

The FlowMntrs problem is also related to the classical k-cut and multi-way cut problems [6,8,1], where the goal is to find a minimum-weight set of edges that partitions the graph into k connected components. One can view our monitor problem as asking to maximize the number of connected components obtained from removing the monitor edges and the resulting bridge edges.

2 Preliminaries

We now give formal definitions. Let $G = (V, E)$ be an undirected graph. We assume that G is simple, that is, it does not have multiple edges or loops, although our algorithms work for multi-graphs with loops as well. Throughout the paper, we use $n = |V|$ to denote the number of vertices in G and $m = |E|$ to be the number of edges. We will typically use letters $u, v, x, y, ...$, possibly with indices, to denote vertices, and $a, b, e, f, ...$ to denote edges. If an edge e has endpoints x, y, we write $e = \{x, y\}$.

A *circulation* on G is a function ψ that assigns a flow value and a direction to any edge in E. (We use the terms "circulation" and "flow" interchangeably, slightly abusing the terminology.) Denoting by $\psi(u, v)$ the flow on $e = \{u, v\}$ from u to v, we require that ψ satisfies the following two conditions (i) ψ is anti-symmetric, that is $\psi(u, v) = -\psi(v, u)$ for each edge $\{u, v\}$, and (ii) ψ satisfies the flow conservation property, that is $\sum_{\{u,v\}\in E} \psi(u, v) = 0$ for each vertex v.

A *bridge* in G is an edge whose removal increases the number of connected components of G. Let $Br(G)$ be the set of bridges in G. The flow value on any bridge of G must be 0, so, without loss of generality, throughout the paper we will be assuming that the input graph does not have any bridges. In other words, each connected component of G is 2-edge-connected. (Recall that, for an integer $c \geq 1$, a graph H is called c-edge-connected, if H is connected and it remains connected after removing any $c - 1$ edges from H.)

Suppose that some circulation ψ is given for all edges in some set $M \subseteq E$, and not for other edges. We have the following observation:

Observation 1. *For $\{u, v\} \in E - M$, $\psi(u, v)$ is uniquely determined if and only if $\{u, v\} \in Br(G - M)$.*

We can now define the *gain* of M to be $gain(G, M) = |M \cup Br(G - M)|$, that is, the total number of edges for which the flow can be determined if we place monitors on the edges in M. We will refer to the edges in M as *monitor* edges, while the bridge edges in $Br(G - M)$ will be called *extra* edges. If G is understood from context, we will write simply $gain(M)$ instead of $gain(G, M)$.

The *Flow Edge-Monitor Problem* (FlowMntrs) can now be defined formally as follows: given a graph $G = (V, E)$ and an integer $k > 0$, find a set $M \subseteq E$ with $|M| \leq k$ that maximizes $gain(G, M)$.

The weighted case. We consider the extension of FlowMntrs to weighted graphs, where each edge e has a non-negative weight $w(e)$ assigned to it, and the task is to maximize the *weighted gain*. More precisely, if M are the monitor edges, then the formula for the (weighted) gain is $gain(M) = \sum_{e\in M\cup B} w(e)$, for $B = Br(G - M)$. We will denote this problem by WFlowMntrs.

Throughout the paper, we denote by M^* some arbitrary, but fixed, optimal monitor edge set. Let $B^* = Br(G - M^*)$ be the set of extra edges corresponding to M^*. Then the optimal gain is $gain^*(G, k) = w(M^* \cup B^*)$.

We now claim that in the weighted case we can restrict our attention to graphs whose all connected components are 3-edge-connected. More specifically, we show that any weighted 2-edge-connected graph $G = (V, E)$ can be converted in linear time into a 3-edge-connected weighted graph $G' = (V', E')$ such that:

(i) $gain^*(G, k) = gain^*(G', k)$, and
(ii) If $M' \subseteq E'$ is a set of k monitor edges in G', then in linear time one can find a set $M \subseteq E$ of k monitor edges in G with $gain(G, M) = gain(G', M')$.

We now show the construction of G'. A 2-*cut* is a pair of edges $\{e, e'\}$ whose removal disconnects G. Write $e \simeq e'$ if $\{e, e'\}$ is a 2-cut. It is known, and quite easy to show, that relation "$\simeq$" is an equivalence relation on E. The equivalence classes of $\simeq$ are called *edge groups.*

Suppose that G has an edge group F with $|F| = q$, for $q \geq 2$, and let $H_1, ..., H_q$ be the connected components of $G - F$. Then $F = \{e_1, ..., e_q\}$, where, for each i, $e_i = \{u_i, v_i\}$, $u_i \in H_i$ and $v_i \in H_{i+1}$ (for $i = q$ we assume $q + 1 \equiv 1$). For $i = 1, ..., q - 1$, contract edge e_i so that vertices u_i and v_i become one vertex, and then assign to edge $e_q = \{u_q, v_q\}$ weight $\sum_{i=1}^{q} w(e_i)$. We will refer to e_q as the *deputy edge* for F. Figure 2 illustrates the construction.

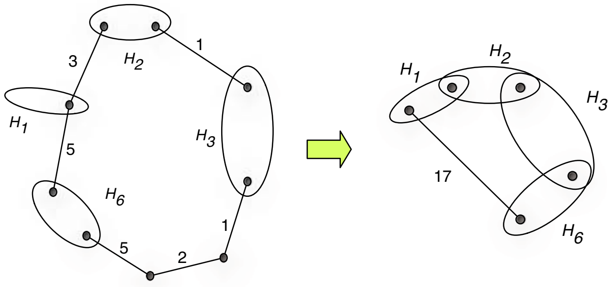

Fig. 2. Contracting edge groups

Let $G' = (V', E')$ be the resulting weighted graph. By the construction, G' is 3-edge-connected. All edge groups can be computed in linear time (see, [7], for example), so the whole transformation can be done in linear time as well.

It remains to show that G' satisfies conditions (i) and (ii). If M is any monitor set, and if M has two or more monitors in the same edge group, we can remove one of these monitors without decreasing the gain of M. Further, for any monitor edge e of M, we can replace e by the deputy edge of the edge group containing e, without changing the gain. This implies that, without loss of generality, we can assume that the optimal monitor set M^* in G consists only of deputy edges. These edges remain in G' and the gain of M^* in G' will be exactly the same as its gain in G. This shows the "$\geq$" inequality in (i). The "$\leq$" inequality follows from the fact that any monitor set in G' consists only of deputy edges from G. The same argument implies (ii) as well.

The kernel graph. Consider a graph $G = (V, E)$ and a monitor edge set M, and let $B = Br(G - M)$. The *kernel graph associated with* G *and* M is defined as a weighted graph $G_M = (V_M, E_M)$, where V_M is the set of connected components of $G - M - B$, and E_M is determined as follows: For any edge $\{u, v\} \in M \cup B$, where $u, v \in V$, let x and y be the connected components of $G - M - B$ that contain, respectively u and v. Then we add edge $\{x, y\}$ to E_M. The weights are preserved, that is $w(\{x, y\}) = w(\{u, v\})$. We will say that this edge $\{x, y\}$ *represents* $\{u, v\}$ or *corresponds to* $\{u, v\}$. In fact, we will often identify $\{u, v\}$ with $\{x, y\}$, treating them as the same object. Note that G_M is a multigraph, as it may have multiple edges and loops (even though G is a simple graph).

Figure 3 shows the kernel graph corresponding to the graph and the monitor set in the example from Figure 1 (all edge weights are 1):

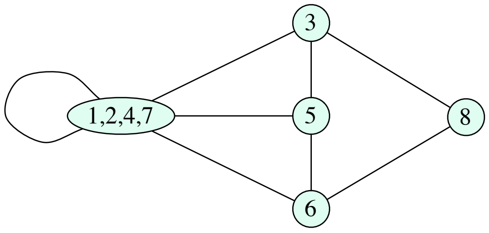

Fig. 3. The kernel graph for the example in Figure 1. The loop in vertex $\{1, 2, 4, 7\}$ represents edge $\{2, 1\}$.

Note that we have $|E_M| \leq k + |V_M| - cc(G_M)$, where $cc(H)$ denotes the number of connected components of a graph H. This can be derived directly from the definitions: The edges in G_M that represent extra edges are the bridges in G_M and therefore they form a forest in G_M. This implies that the number of extra edges is at most $|V_M| - cc(G_M)$, and the inequality follows.

In the paper, we will use the concept of kernel graphs only with respect to some optimal monitor set. Let M^* be some arbitrary, but fixed, optimal monitor edge set. To simplify notation, we will write $G^* = (V^*, E^*)$ for the kernel graph associated with M^*, that is $G^* = G_{M^*}$, $V^* = V_{M^*}$ and $E^* = E_{M^*}$. In this notation, we have $gain^*(G, k) = w(E^*)$. In the analysis of our algorithms, we will be comparing the weights of edges collected by the algorithm against the edges in the kernel graph G^*.

3 Proof of NP-hardness of FlowMntrs

We show that the FlowMntrs is NP-hard (even in the unweighted case), via a reduction from the Clique problem. We start with a simple lemma whose proof is omitted. (In the lemma, we assume that $\binom{1}{2} = 1(1-0)/2 = 0$.)

Lemma 1. *Let $a_1, a_2, \ldots, a_s$ be s positive integers such that $\sum_{i=1}^{s} a_i = n$, for a fixed integer n. Then $\sum_{i=1}^{s} \binom{a_i}{2}$ is maximized if and only if $a_j = n - s + 1$ for some j and $a_i = 1$ for all $i \neq j$.*

Theorem 2. FlowMntrs *is* NP*-hard.*

Proof. In the Clique problem, given an undirected graph $G = (V, E)$ and an integer $q > 0$, we wish to determine if G has a clique of size at least q. Clique is well-known to be NP-complete (see [2]). We show how to reduce Clique, in polynomial-time, to DecFlowMntrs, the decision version of FlowMntrs, defined as follows: Given a graph $G = (V, E)$ and two integers, $k, l > 0$, is there a set M of k edges in G for which $|Br(G - M)| \geq l$?

The reduction is simple. Suppose we have an instance $G = (V, E), q$ of Clique. We can assume that G is connected and $q \geq 3$. Let $n = |V|$ and $m = |E|$. We map this instance into an instance G, k, l of DecFlowMntrs, where $k = m - \binom{q}{2} - l$ and $l = n - q$. This clearly takes polynomial time. Thus, to complete the proof, it is sufficient to prove the following claim:

$(*)$ G has a clique of size q iff G has a set M of k edges for which $|Br(G - M)| \geq l$.

We now prove $(*)$. The main idea is that, by the choice of parameters k and l, the monitors and extra edges in the solution of the instance of DecFlowMntrs must be exactly the edges outside the size-q clique of G.

$(\Rightarrow)$ Suppose that G has a clique C of size q. Let G' be the graph obtained by contracting C into a single vertex and let T be a spanning tree of G'. We then take M to be the set of edges of G' outside T. Thus the edges in T will be the bridges of $G - M$. Since G' has $n - q + 1$ vertices, T has $l = n - q$ edges, and M has $m - \binom{q}{2} - l = k$ edges.

$(\Leftarrow)$ Suppose there is a set M of k monitor edges that yields a set B of l' extra edges, where $l \leq l' \leq n - 1$. We show that G has a clique of size q.

Let s be the number of connected components of $G - M - B$, and denote by $a_1, a_2, ..., a_s$ the cardinalities of these components (numbers of vertices). Since $|B| = l'$, we have $s \geq l' + 1$. Also, $\sum_{i=1}^{s} a_i = n$ and $\sum_{i=1}^{s} \binom{a_i}{2} + k + l' \geq m$. Therefore, using Lemma 1, and the choice of k and l, we have

$$\binom{n - l'}{2} + l' \;\geq\; \binom{n - s + 1}{2} + l' \;\geq\; \sum_{i=1}^{s} \binom{a_i}{2} + l' \;\geq\; m - k \;=\; \binom{n - l}{2} + l.$$

By routine calculus, the function $f(x) = \frac{1}{2}(n - x)(n - x - 1) + x$ is decreasing in interval $[0, n - 1]$, and therefore the above derivation implies that $l' \leq l$, so we can conclude that $l' = l$. This, in turn, implies that all inequalities in this derivation are in fact equalities. Since the first inequality is an equality, we have $s - 1 = l' = l = n - q$. Then, since the second inequality is an equality, Lemma 1 implies that $a_j = q$ for some j and $a_i = 1$ for all $i \neq j$. Finally, the last inequality can be an equality only if all the connected components are cliques. In particular, we obtain that the jth component is a clique of size q.

4 Algorithm σ-Greedy

Fix some integer constant $\sigma \geq 1$. Let $G = (V, E)$ be the input graph with weights on edges. For simplicity, we will assume that G is connected. (A full argument will appear in the final version.) As explained in Section 2, we can then also assume that G is 3-edge-connected.

Algorithm σ-Greedy that we study in this section works in $\lceil k/\sigma \rceil$ steps and returns a set of k monitor edges. In each step, it assigns σ monitors to a set P of σ remaining edges that maximizes the gain in this step, that is, the total weight of the monitor edges and the resulting bridges. A more rigorous description is given in Figure 4, which also deals with special cases when the number of monitors or edges left is less than σ.

```
Algorithm $\sigma$-Greedy
    $G_0 = (V, E_0) \leftarrow G = (V, E)$
    $M_0 \leftarrow \emptyset$
    $X_0 \leftarrow \emptyset$
    for $t \leftarrow 1, 2, ..., \lceil k/\sigma \rceil$
        if $E_{t-1} = \emptyset$
            then return $M = M_{t-1}$ and halt
        $\sigma' \leftarrow \sigma$
        if $t = \lfloor k/\sigma \rfloor + 1$
            then $\sigma' = k \bmod \sigma$
        if $|E_{t-1}| \leq \sigma'$
            then $P \leftarrow E_{t-1}$
            else
                find $P \subseteq E_{t-1}$ with $|P| = \sigma'$
                    that maximizes $w(P \cup Br(G_{t-1} - P))$
                $Y_t \leftarrow P \cup Br(G_{t-1} - P)$
                $X_t \leftarrow X_{t-1} \cup Y_t$
                $E_t \leftarrow E_{t-1} - Y_t$
                $G_t \leftarrow (V, E_t)$
        $M_t \leftarrow M_{t-1} \cup P$
    return $M = M_{\lceil k/\sigma \rceil}$
```

Fig. 4. Pseudo-code for Algorithm σ-Greedy. Y_t represents the edges collected by the algorithm in step t, with $P \subseteq Y_t$ being the set of monitor edges and $Y_t - P$ the set of extra edges. M_t represents all monitor edges collected up to step t and X_t represents all edges collected up to step t.

Note that each step of the algorithm runs in time $O(m^\sigma(n+m))$, by trying all possible combinations of σ edges in the remaining graph G_{t-1} to find P. Hence, for each fixed σ, Algorithm σ-Greedy runs in polynomial time.

4.1 Analysis of 1-Greedy

For $\sigma = 1$, Algorithm 1-Greedy is: At each step, choose an edge whose removal creates a maximum number of bridges, and place a monitor on this edge. Then remove this edge and the resulting bridges. We show that this algorithm has approximation ratio 3.

Analysis. For simplicity, assume that the input graph G is connected. As explained in previous sections, we can assume that G is in fact 3-edge-connected.

Fix the value of k, and some optimal solution M^*, and let $G^* = (V^*, E^*)$ be the corresponding kernel graph. To avoid cluttered notation, we will identify each edge in E^* with its corresponding edge in E, thus thinking of E^* as a subset of E. For example, when we say that the algorithm collected some $e \in E^*$, we mean that it collected the edge in E represented by e.

Recall that $gain^*(G, k) = w(E^*)$, where $w(E^*)$ is the sum of weights of the edges in E^*. Thus we need to show that our algorithm's gain is at least $\frac{1}{3}w(E^*)$.

Intuitively, since G is 3-edge-connected, each vertex in G^* has degree at least 3, so $|E^*| \geq \frac{3}{2}|V^*|$. Thus $k = |E^*| - |V^*| + 1 > \frac{1}{3}|E^*|$. 1-Greedy collects at least k edges. Since 1-Greedy maximizes the gain at each step, its total gain will be at least the total weight of the $\frac{1}{3}|E^*|$ heaviest edges in E^*. (This does not mean, however, that 1-Greedy will collect the $\frac{1}{3}|E^*|$ heaviest edges.)

We now give a more rigorous argument. The proof is by amortized analysis. We will analyze consecutive steps of the algorithm, while maintaining a dynamic set L_t of edges. Initially, we set $L_0 = E^*$. As the algorithm collects edges in each step t, we will also remove edges from L_{t-1}, so that $L_t \subseteq L_{t-1}$ for $t \geq 1$. In addition, this set L_t will satisfy the following conditions for each step $t = 1, 2, ...$:

(L1.1) $L_t \cap X_t = \emptyset$; that is, all edges in L_t are available to the algorithm after step t;
(L1.2) $w(Y_t) \geq \frac{1}{3}w(L_{t-1} - L_t)$; that is, our gain at each step is at least one third of the total weight of all the edges removed from L_{t-1}; and
(L1.3) $|L_{t-1}| - |L_t| \geq \min\{3, |L_{t-1}|\}$.

We claim that the conditions above imply that 1-Greedy's approximation ratio is 3. Since $k \geq \frac{1}{3}|E^*|$, from (L1.3) and amortization, we have $L_k = \emptyset$. Then, again by amortization, (L1.2) implies that $w(X_k) \geq \frac{1}{3}w(E^*)$, as claimed.

It thus remains to show how to update L_t to maintain (L1.1), (L1.2) and (L1.3). Suppose that these conditions hold up to step $t - 1$. Let $\gamma = |Y_t \cap L_{t-1}|$ be the number of edges collected in step t that are in L_{t-1}. We first set $L_t \leftarrow L_{t-1} - Y_t$. Next, if $\gamma < 3$, we further remove arbitrary $3 - \gamma$ edges from L_t (If it so happens that L_t has fewer than $3 - \gamma$ edged, then we remove all edges from L_t.) Since we have removed all Y_t from L_{t-1}, (L1.1) is preserved. Moreover, since we either remove at least 3 edges or $L_t = \emptyset$, (L1.3) is preserved as well. Finally, by the algorithm and (L1.1), $w(Y_t)$ is at least as large as the weight of each of the $3 - \gamma$ additional edges removed from L_{t-1}, which implies (L1.2).

Summarizing the argument above, we obtain:

Theorem 3. *Algorithm* 1-Greedy *is a polynomial-time* 3*-approximation algorithm for the* WFlowMntrs *problem.*

With a somewhat more careful analysis, one can show that the approximation ratio of 1-Greedy is actually $3(1 - 1/k)$, which matches our lower bound example below. Also, we remark that the proof above is perhaps unnecessarily technical, but we introduce it here on purpose, as a stepping stone to a much more involved analysis of Algorithm 2-Greedy in the next section.

A tight-bound example. We now present an example showing that our analysis of 1-Greedy is tight. Graph G consists of one connected component with $2k-2$ vertices, in which each vertex has degree 3 and each edge has weight 1, and the other connected component that has only two vertices connected by $k+2$ edges each of weight $1+\epsilon$. Fig. 5 shows the construction for $k=5$.

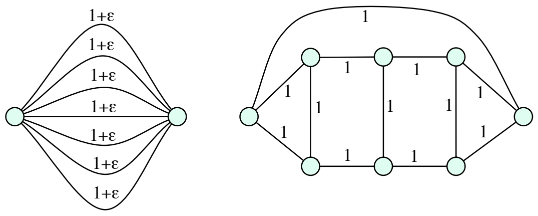

Fig. 5. Lower bound example for 1-Greedy, with $k=5$

1-Greedy will be collecting edges from the 2-vertex component on the left, ending up with k edges and total gain $(1+\epsilon)k$. The optimum solution is to put k monitors in the cubic component on the right, thus gaining all $3k-3$ edges from this component. For $\epsilon \to 0$, the approximation ratio tends to $3(1-1/k)$.

4.2 Analysis of 2-Greedy

Let $G=(V,E)$ be the input graph with weight on edges. As in the previous section we will assume that G is 3-edge-connected. For $\sigma=2$, Algorithm 2-Greedy, at each step, collects two edges whose total weight combined with the weight of all resulting bridges, is maximized among all possible choices of two edges. Ties are broken arbitrarily. We place monitors on these two edges, and then remove them from G, as well as the resulting bridges. The exceptional situations, when k is odd, or we run out of edges, etc., are handled as in Figure 4.

Analysis. We can assume that the algorithm never runs out of edges (that is, $E_{t-1} \neq \emptyset$ for each step t), for otherwise it computes the optimum solution. For simplicity, we will assume that k is even. If k is odd, the proof below can be shown to work by taking into account the gain of the algorithm in the last step when it has only one monitor. We also fix some optimal solution M^*, and let $G^* = (V^*, E^*)$ be the corresponding kernel graph. Recall that $gain^*(G,k) = w(E^*)$; thus we need to show that our algorithm's gain is at least $\frac{1}{2}w(E^*)$.

Before we delve into formal proof, we give a high level description of our approach. We start with a set L which contains two copies of every edge in E^*, so that $w(L) = 2w(E^*) = 2gain^*(G,k)$. It thus suffices to show that the algorithm's gain in $k/2$ steps is at least $w(L)/4$. In the analysis of one step we remove some copies of edges from L, while guaranteeing that their total weight is at most 4 times 2-Greedy's gain in this step. Then we show that after $k/2$

steps L becomes empty. Hence the algorithm must have collected a set of edges with total weight at least $w(L)/4$, as needed. Below we give the complete proof.

As in the previous section, the proof is by amortized analysis. If $e = \{x, y\}$ is an edge, then by e^x and e^y we will denote two copies of e, and we will call them *arcs.* We also say that arcs e^x and e^y are *associated* with e. One can think of these arcs as directed edges, with $e^x = (x, y)$ directed from x to y, and e^y directed in the opposite direction. Each arc e^x has weight $w(e^x) = w(e)$. In our analysis, we maintain a dynamic set L_t of arcs, for each step t, that will satisfy invariants similar to those in the previous section. Initially, L_0 contains two copies of each edge, that is $L_0 = \{e^x, e^y : e = \{x, y\} \in E^*\}$. Note that $|L_0| = 2|E^*|$ and $w(L_0) = 2w(E^*)$. As we analyze each step t of the algorithm, we will be removing some arcs from L_{t-1}, so that we will have $L_t \subseteq L_{t-1}$ for each $t = 1, 2, ..., k/2$. L_t will not contain any arcs associated with edges already collected by the algorithm in the first t steps. For an edge in E^*, L_t may contain both its associated arcs, one of the two arcs, or neither of them.

Let x be a vertex of degree-3 in G^* and e, f and g be the three edges incident to x. We will say that x is a *tripod at step* t if none of edges e, f, g have been collected by 2-Greedy when step $t-1$ completes. Arcs e^x, f^x, g^x are then called *tripod arcs* or *arcs of* x. We say that x is a *bipod at step* t if exactly one of these edges has been collected by 2-Greedy. (Note that the case when exactly two of these edges was collected is not possible since any two collected by the algorithm implies the algorithm also gets the third one.) If, say, g is the one edge collected by 2-Greedy, then e^x and f^x are called *bipod arcs* or *arcs of* x.

Let x be a tripod at step t. If $A \subseteq L_0$ is a set of arcs, we say that x *is in* A if its three arcs e^x, f^x, g^x are in A. Similarly, if x is a bipod at time t, we say that x *is in* A if both its arcs e^x, f^x are in A. Let $\xi(A)$ be the total number of tripods and bipods in A. It will be convenient to have some name for arcs in A that are neither tripod arcs nor bipod arcs. We refer to such arcs as *loose arcs in* A. Note that for an edge $e = \{x, y\}$, arc $e^x \in A$ could be a loose arc while the opposite arc e^y is a tripod arc or a bipod arc.

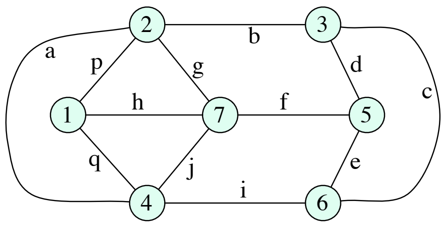

Fig. 6. Example of a graph G^*

Consider, for example, the graph G^* in Figure 6, where all weights are 1. For this graph, we would have $L_0 = \{a^2, a^4, b^2, b^3, c^3, c^6, \ldots, p^1, p^2, q^1, q^4\}$. The tripods of L_0 are vertices 1, 3, 5 and 6, so $\xi(L_0) = 4$. By definition of bipod, the initial set L_0 does not contain any bipods. All arcs in L_0 that do not belong to a tripod are loose arcs, for example q^4, a^4, j^7, etc. If the algorithm picks edges b, d in step 1, c will become a bridge, so the gain is 3. In the analysis, all arcs associated with these edges will be removed from L_0. Assuming that in the analysis we will not remove any arcs associated with edges f, e and i, L_1 will contain two bipods, 5 and 6.

In our analysis, we will assume for now that $|L_0| - \xi(L_0)$ is even; later we will explain how to modify the proof to cover the case when this quantity is odd. As mentioned earlier, the overall idea of the proof is to construct a decremental sequence $L_0 \supseteq L_1 \supseteq L_2 \supseteq ...$ that satisfies appropriate conditions, from which we can derive a bound on the gain of 2-Greedy. In fact, we will actually construct the sets $\Delta_t \subseteq L_{t-1}$ of arcs to be removed at each step, so that $L_t = L_{t-1} - \Delta_t$. We claim that for any given step t, $1 \leq t \leq k/2$. there exists a set of arcs $\Delta_t \subseteq L_{t-1}$ that satisfies the following conditions:

(D2.1) If $e^x \in L_{t-1}$ and $e \in Y_t$ then $e^x \in \Delta_t$.
(D2.2.) $w(Y_t) \geq \frac{1}{4} w(\Delta_t)$.
(D2.3) Either (a) $|\Delta_t| - \xi(\Delta_t) \geq 8$ and $|\Delta_t| - \xi(\Delta_t)$ is even, or (b) $|\Delta_t| - \xi(\Delta_t) < 8$ and $\Delta_t \cup \Delta_{t+1} = L_{t-1}$ (in other words, we will remove all arcs in this and next step).
(D2.4) For any bipod x in L_{t-1}, if one arc of x is in Δ_t, then both of them are in Δ_t. For any tripod x in L_{t-1}, if two of the arcs of x are in Δ_t, then all three arcs are in Δ_t.

Recall that Y_t is the set of edges collected by the algorithm at step t, including the two monitor edges and the resulting bridges. Condition (D2.1) ensures that L_t contains only arcs whose associated edges are available to the algorithm right after step t. Condition (D2.2) states that 2-Greedy's gain in this step is at least $\frac{1}{4}$th of the total weight of arcs removed from L_{t-1}. Condition (D2.3) says that we remove a sufficient number of arcs from L_{t-1} at step t, guaranteeing that we will empty L_t at or before step $k/2$. The parity condition in (D2.3) and condition (D2.4) are of more technical nature, and their significance will become apparent in the construction of Δ_t below.

Before explaining how to construct such a set Δ_t, we first show that the existence of Δ_t implies the 2-approximation of 2-Greedy. As explained earlier, we define $L_t = L_{t-1} - \Delta_t$, for any $t = 1, 2, ...$ Denoting by η_d the number of vertices in G^* of degree d, we have

$$\begin{aligned}|L_0| - \xi(L_0) &= \textstyle\sum_{d \geq 3} d\eta_d - \eta_3 \;=\; 2\eta_3 + \sum_{d \geq 4} d\eta_d \;\leq\; 2\sum_{d \geq 3}(d-2)\eta_d \\ &= 4 \cdot (\tfrac{1}{2}\textstyle\sum_{d \geq 3} d\eta_d - \sum_{d \geq 3}\eta_d) \;=\; 4(|E^*| - |V^*|) \;\leq\; 4k - 4. \quad (1)\end{aligned}$$

From invariant (D2.3), by amortization over all steps, we have $|L_{k/2}| - \xi(L_{k/2}) \leq \max\{0\,,\, |L_0| - \xi(L_0) - 4k\} \leq 0$. On the other hand, for all t, whenever $L_t \neq \emptyset$, we have $|L_t| > \xi(L_t)$. We thus conclude that $L_{k/2} = \emptyset$. This, together with condition (D2.2) and amortization, implies that $w(X_{k/2}) \geq \frac{1}{4} w(L_0) = \frac{1}{2} w(E^*) = \frac{1}{2} gain^*(G, k)$. Here $X_t = \bigcup_{j=1}^{t} Y_t, t = 1, 2, \ldots, k/2$ is the set of edges collected by the algorithm up to and include step t. Thus 2-Greedy approximates the optimum solution within a factor of 2.

To complete the analysis, it remains to show how to construct a set Δ_t that satisfies conditions (D2.1) to (D2.4). Suppose we have already constructed sets Δ_s, for $s = 1, 2, ..., t-1$. Given the definition of L_0, the assumption that $|L_0| - \xi(L_0)$ is even, as well as conditions (D2.1), (D2.3) together imply that L_{t-1} satisfies the following three conditions:

(L2.1) If $e^x \in L_{t-1}$ then $e \notin X_{t-1}$.
(L2.2) $|L_{t-1}| - \xi(L_{t-1})$ is even.

We start with a high level idea. We first include in Δ_t all arcs in L_{t-1} associated with edges in Y_t, the set of edges collected at step t. This will satisfy condition (D2.1). Note that removing additional arcs from the dynamic set L will not violate (D2.1). The total weight of these arcs will be at most $2w(Y_t)$, since every edge is associated with at most two arcs and every arc removed is associated with some edge in Y_t. Thus, at least so far, condition (D2.2) holds as well. To satisfy condition (D2.3), we may need to include more arcs to Δ_t. This requires that we keep a delicate balance between the number of additional arcs to be included and their total weight: If we include too many arcs, we may violate (D2.2) because the total weight of arcs in Δ_t is too large. On the other hand, if we include too few arcs, we may not be able to satisfy (D2.3) which requires $|\Delta_t| - \xi(\Delta_t)$ to be at least 8.

Thus we will have $\Delta_t = \Delta'_t \cup \Delta''_t$, where Δ'_t and Δ''_t are the arcs from L_{t-1} removed in Stage 1 and Stage 2, respectively. We now describe these two stages.
Stage 1: removing affected arcs. We set Δ'_t to be the set of all arcs $e^x \in L_{t-1}$ such that $e \in Y_t$. These arcs can be grouped into the following four categories:

Case (I): Loose arcs. Any loose arc $e^x \in L_{t-1}$ associated with $e \in Y_t$, contributes 1 to $|\Delta'_t| - \xi(\Delta'_t)$.
Case (II): Triples of tripod arcs. Suppose that x is a tripod in L_{t-1} and e^x, f^x, g^x are its arcs. If $e, f, g \in Y_t$, then the arcs of x will contribute 2 to $|\Delta'_t| - \xi(\Delta'_t)$.
Case (III): Single tripod arcs. Suppose that x is a tripod in L_{t-1} and e^x, f^x, g^x be its arcs. If $e \in Y_t$ but $f, g \notin Y_t$, then e^x contributes 1 to $|\Delta'_t| - \xi(\Delta'_t)$.
Case (IV): Pairs of bipod arcs. Suppose that x is a bipod in L_{t-1} and e^x, f^x are its arcs. If $e, f \in Y_t$, then the two bipod arcs of x will together contribute 1 to $|\Delta'_t| - \xi(\Delta'_t)$.

Note that the above four cases exhaust all possibilities. Clearly (D2.1) holds. (D2.2) is true because each edge in Y_t is associated with at most two arcs. (D2.4) follows from the algorithm since, if it collects one edge of a bipod, then it also collects the other; if it collects two of three edges of a tripod, then the third one is collected as well. So only (D2.3) needs further attention. In Stage 2 we shall include additional arcs in Δ_t to satisfy (D2.3).

Stage 2: removing additional arcs. Now we choose a set $\Delta''_t \subseteq L_{t-1} - \Delta'_t$ of additional arcs that will be removed from L_{t-1}. Let $L' = L_{t-1} - \Delta'_t$, and define $\delta' = (|L_{t-1}| - \xi(L_{t-1})) - (|L'| - \xi(L')) = |\Delta'_t| - \xi(\Delta'_t)$. We have two cases, depending on the value of δ'.
Case 1: δ' is even. If $\delta' \geq 8$, then we are done. Now consider subcases $\delta' = 2, 4, 6$.

Case 1.1: $\delta' = 2$. In this sub-case, Δ'_t may contain either one tripod, two independent bipods, or one bipod and one arc that are independent, or two independent arcs. Here tripods and bipods are with respect to L_{t-1}, arc may be a tripod arc or a loose arc, and independent simply means they do not contain two arcs associated with the same edge. In every case we have $w(\Delta'_t) \le w(Y_t)$ due to the algorithm. Now we need to bring down $|L| - \xi(L)$ further by 6. Recall L is the dynamic set of arcs that we are maintaining in the analysis of the algorithm. It is easy to see that every tripod in $L' = L_{t-1} - \Delta'_t$ has weight at most $w(Y_t)$ and each counts 2 toward $|L|-\xi(L)$. So if there are enough tripods in L', then we may include 3 tripods in Δ''_t and we are done. If we run out of tripods, then we may use bipods and loose arcs. Any two independent loose arcs or bipods have total weight no more than $w(Y_t)$ and bring down $|L|-\xi(L)$ by 2. One potential issue is that a bipod(call it *new* bipod) might result from removing a tripod arc in constructing Δ'_t in Stage 1. Since $\delta' = 2$, this new bipod together with the tripod arc will have total weight no more than $w(Y_t)$ and bring down $|L| - \xi(L)$ by 2, while the other arc or bipod in Δ'_t together with the other arc or bipod(could be a new bipod too) chosen in stage 2 so far will have weight no more than $w(Y_t)$(Either they both belong to the same tripod in L_{t-1} or they are they are independent, as all arcs or bipods in stage 1 are independent of bipods or arcs in stage 2) and also bring down $|L| - \xi(L)$ by 2. Overall we are removing arcs with total weight at most $2w(Y_t)$ and bring down $|L|-\xi(L)$ by 4, and this is the same as if the new bipod were a bipod in L_{t-1}. We may further look for two tripods or two pairs of independent bipods(cannot be new bipods) or loose arcs until we have Δ''_t such that $\delta'' = |\Delta''_t| - \xi(\Delta''_t) = 6$ and $w(\Delta''_t) \le 3w(Y_t)$. If the above completes successfully, then we have $w(\Delta_t) = w(\Delta'_t) + w(\Delta''_t) \le w(Y_t) + 3w(Y_t) = 4w(Y_t)$ and $\delta = \delta' + \delta'' = 8$ as desired.

It is also possible that we do not have a pair of independent bipods or loose arcs while δ'' is still less than 6. And this happens after we removed arcs in Δ'_t and Δ''_t from the dynamic set L. We first note that this can happen only when $|L| - \xi(L) \le 3$. The reason is that, as long as $|L| - \xi(L) \ge 4$, and L contains no tripods, we know the count of bipods and the count of loose arcs combined must be at least 4. But a bipod can depend on at most two bipods or loose arcs, hence it must be independent of one of the three bipods or loose arcs. So to be in this situation we have $|L|-\xi(L) \le 3$ and L has no tripods. The set L contains either 3 interlocked bipods forming a 3-cycle, or two dependent bipods or loose arcs. And it is not hard to see that $w(L) \le 3w(Y_t)$. If we have one more step to go, then in the next step $t+1$ the algorithm gets a gain of at least the weight of one bipod or loose arc that is in L, and we shall have $w(Y_{t+1}) \ge w(L)/4$ since there are at most 3 bipods or loose arcs, and the set L becomes empty after those dependent bipods and arcs are removed. Otherwise we are in the last step, step $k/2$. Recall inequality (1) is actually $|L_0| - \xi(L_0) \le 4(k-1)$. Given that we have kept $|L| - \xi(L)$ down by at least 8 in all previous steps except the last step, we must have $|L_{k/2-1}| - \xi(L_{k/2-1}) \le 4$. It follows that after stage 1 of step $k/2$, we shall have $|L| - \xi(L)$ no more than 2. In this case we can simply include all the dependent bipods or loose arcs in Δ''_t. Their total weight is no

more than $3w(Y_t)$ and we have emptied the set L, while maintained (D2.2) since $w(\Delta_{k/2}) = w(\Delta'_{k/2}) + w(\Delta''_{k/2}) \leq w(Y_t) + 3w(Y_t) \leq 4w(Y_t)$ as desired.
Case 1.2: $\delta' = 4$ and $w(\Delta'_t) \leq 2w(Y_t)$. If we can find 2 tripods in $L' = L_{t-1} - \Delta'_t$, then we are done since each tripod contributes 2 to δ'' and its weight is no more than $w(Y_t)$. Otherwise we may use bipods and loose arcs. Because all bipods and arcs in $L' = L_{t-1} - \Delta'_t$ are independent of bipods or arcs in Δ'_t, we can pair up bipods or arcs in Δ'_t and bipods or arcs in L'. Each pair is either from the same tripod in L_{t-1} or the two in that pair are independent with no new bipods, hence each pair contributes 2 to δ'' and its weight is no more than $w(Y_t)$. So we are able to construct Δ''_t such that $w(\Delta'_t) + w(\Delta''_t) \leq 4w(Y_t)$ and $\delta' + \delta'' = 8$. Additional pairs of indepdent bipods or loose arcs may be needed to construct Δ''_t just described. In case we do not have independent pairs, we argue as in $\delta' = 2$ subcase that we are able to empty the set L in the next step.

$\delta' = 6$ and we have $w(\Delta'_t) \leq 2w(Y_t)$. We simply include one tripod, or two bipods or loose arcs in Δ''_t. The total weight is no more than $2w(Y_t)$, and we have $\delta'' = 2$. Overall we have $w(\Delta_t) = w(\Delta'_t) + w(\Delta''_t) \leq 2w(Y_t) + 2w(Y_t) = 4w(Y_t)$ and $\delta = \delta' + \delta'' = 6 + 2 = 8$ as desired. If we are not able to find any tripods, bipods and loose arcs, then effectively we have already emptied the dynamic set L while maintained (D2.2), that is, total weight of arcs removed from the set L at each step is no more than 4 times the gain of the algorithm at that step.

The analysis of the case when δ' is odd, as well as other the remaining cases, will appear in the full paper. Summarizing, we obtain our main result.

Theorem 4. *Algorithm* 2-Greedy *is a polynomial-time* 2*-approximation algorithm for the Weighted Flow Edge-Monitor Problem.*

A tight-bound example. The example is essentially the same as the one for 1-Greedy (see Figure 5), except that the edges in the 2-vertex component on the left side have now weights $1.5 + \epsilon$. 2-Greedy will collect edges from this 2-vertex component, so its total gain will be $(1.5 + \epsilon)k$, while the optimum gain is $3k - 3$. For $\epsilon \to 0$ and $k \to \infty$, the ratio tends to 2.

5 Final Comments

The most intriguing open question is what is the approximation ratio of σ-Greedy in the limit for $\sigma \to \infty$. We can show that this limit is not lower than 1.5, and we conjecture that 1.5 is indeed the correct answer.

A natural question to ask is whether our results can be extended to directed graphs. It is not difficult to show that this is indeed true; both the NP-hardness proof and 2-approximation can be adapted to that case.

Another direction to pursue would be to study the extension of our problem to arbitrary linear systems of equations. Here we can put k "monitors" on k variables of the system to measure their values. The objective is to maximize the number of variables whose values can be uniquely deduced.

References

1. Dahlhaus, E., Johnson, D.S., Papadimitriou, C.H., Seymour, P.D., Yannakakis, M.: The Complexity of Multiway Cuts (Extended Abstract). In: 24th ACM Symposium on Theory of Computing, pp. 241–251 (1992)
2. Garey, M.R., Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W.H. Freeman, New York (1979)
3. Goldschmidt, O., Hochbaum, D.S.: Polynomial Algorithm for the k-Cut Problem. In: 29th Annual IEEE Symposium on Foundations of Computer Science, pp. 444–451 (1988)
4. Gu, W., Jia, X.: On a Traffic Control Problem. In: 8th International Symposium on Parallel Architectures, Algorithms and Networks, pp. 510–515 (2005)
5. Khuller, S., Bhatia, R., Pless, R.: On Local Search and Placement of Meters in Networks. SIAM J. on Comput. 32, 470–487 (2003)
6. Saran, H., Vazirani, V.V.: Finding k-cuts within Twice the Optimal. In: 32nd Annual Symposium on Foundations of Computer Science, pp. 743–751 (1991)
7. Tsin, Y.H.: A Simple 3-Edge-Connected Component Algorithm. Theor. Comp. Sys. 40, 125–142 (2007)
8. Vazirani, V.V.: Approximation Algorithms. Springer, Heidelberg (2001)