Finding Paths with Minimum Shared Edges[⋆]

Masoud T. Omran[1], Jörg-Rüdiger Sack[1], and Hamid Zarrabi-Zadeh[2]

[1]School of Computer Science, Carleton University, Ottawa, Ontario K1S 5B6, Canada.
{mtomran,sack}@scs.carleton.ca
[2]Department of Computer Engineering, Sharif University of Technology, Tehran, Iran.
zarrabi@sharif.edu

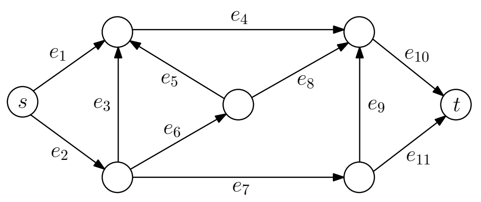
Abstract. Motivated by a security problem in geographic information systems, we study the following graph theoretical problem: given a graph G, two special nodes s and t in G, and a number k, find k paths from s to t in G so as to minimize the number of edges shared among the paths. This is a generalization of the well-known disjoint paths problem. While disjoint paths can be computed efficiently, we show that finding paths with minimum shared edges is NP-hard. Moreover, we show that it is even hard to approximate the minimum number of shared edges within a factor of $2^{\log^{1-\varepsilon} n}$, for any constant $\varepsilon > 0$. On the positive side, we show that there exists a $(k-1)$-approximation algorithm for the problem, using an adaption of a network flow algorithm. We design some heuristics to improve the quality of the output, and provide empirical results.

1 Introduction

In this paper, we address a problem motivated by a security assurance demand in a geographic information system (GIS) setting. The problem set arose in the following context. Suppose that a security organization is hired to do planning for a VIP who wishes to travel safely between two locations. Given the security concerns, k paths are determined in pre-trip planning and then, just prior to actual travel, randomly one path among the k paths is chosen. The fewer edges that are shared among the pre-trip paths, the higher the level of perceived security. However, if it becomes unavoidable to share edges among the paths, guards are employed on those shared edges. Once a guard has been employed for a particular edge, he/she protects all paths that use this edge. Since guards are expensive, we want to reduce their total number. We refer to this problem as *Minimum Shared Edges*, or *MSE* for short. The problem is formally defined as follows:

Problem 1 (Minimum Shared Edges (MSE)). Given a graph $G = (V, E)$, two special nodes $s, t \in V$, and an integer $k > 0$, find a set P of k paths from s to t in G so as to minimize $c(P) = \sum_{e \in E} \lambda(e)$, where $\lambda(e) = 0$ if e is used in at

[⋆] Research supported by NSERC, SUN Microsystems and HPCVL. A preliminary version of this work has been presented at COCOON 2011.



$\pi_1 = \langle e_1, e_4, e_{10} \rangle$
$\pi_2 = \langle e_2, e_7, e_{11} \rangle$
$\pi_3 = \langle e_2, e_6, e_8, e_{10} \rangle$
$\pi_4 = \langle e_2, e_3, e_4, e_{10} \rangle$
$\pi_5 = \langle e_2, e_7, e_9, e_{10} \rangle$
$\pi_6 = \langle e_2, e_6, e_5, e_4, e_{10} \rangle$

Fig. 1. A graph G with six possible (s, t)-paths, denoted by π_1 to π_6.

most one path of P, and $\lambda(e) = 1$ otherwise. An edge e with $\lambda(e) = 1$ is called a *shared edge.*

We assume, without loss of generality, that the input graph is directed. Figure 1 illustrates an instance of the MSE problem on a sample graph. For $k = 2$, the minimum possible number of shared edges is zero, attained by two paths π_1 and π_2. For $k = 3$, the minimum number of shared edges is two, realized by the set $\{\pi_1, \pi_2, \pi_3\}$. Any other set of three paths leads to a higher number of shared edges.

For the special case where the number of shared edges is required to be zero, the MSE problem is reduced to the "disjoint paths" problem which can be solved in polynomial time using standard maximum flow algorithms. In particular, one can use Goldberg and Rao's binary blocking flow algorithm [7] to find k disjoint paths in a graph $G = (V, E)$ in $O(m \min(n^{2/3}, m^{1/2}) \log(n^2/m) \log k)$ time, where $n = |V|$ and $m = |E|$. An improved algorithm is provided for the special case of $k = 2$ [15]. See also [9] for the related problem of finding "shortest" disjoint paths in a graph.

More complex variants of the disjoint paths problem have also been studied in the literature. The problem of finding a pair of disjoint (s, t)-paths such that the length of the shorter path is minimized is addressed in [16] and proved to be NP-hard. The authors show that finding a k-approximation of the optimal solution is NP-hard for any $k > 1$. Li et al. [13] studied a similar problem in which the objective is to find a pair of disjoint (s, t)-paths such that the length of the longer path is minimized and showed that it is NP-hard. Itai et al. [8] showed that the problem of finding a pair of disjoint (s, t)-paths such that the length of the shorter and longer path is bounded by Δ_1 and Δ_2, respectively, is also NP-hard. The general case of finding k min-sum (s, t)-disjoint paths where every edge is assigned k different costs and the jth edge cost is associated with the jth path, has been studied by Li et al. [12] and proved to be NP-hard.

A closely related problem studied in the context of communication networks is the so-called "k-best paths" problem [3, 14]. In this problem, the objective is to find a set P of k paths with minimum edge sharability, which is defined analogously to Problem 1, with the only difference that here, for each edge e, $\lambda(e) = 0$ if e is used in at most one path of P, otherwise $\lambda(e)$ is equal to the number of paths containing e minus 1. As shown in [11, 17], the k-best paths problem is polynomially solvable using a minimum-cost flow algorithm.

Despite its close similarity to the k-best paths problem, the minimum shared edges problem studied in this paper turns out to be substantially more challenging. In particular, we prove that the minimum shared edges problem is NP-hard. Moreover, we show that the problem admits no $2^{\log^{1-\varepsilon} n}$-factor approximation, for any constant $\varepsilon > 0$, unless NP $\subseteq$ DTIME($n^{\text{polylog}\, n}$). On the other hand, we show that there exists a $(k-1)$-approximation algorithm for the problem, using a simple adaption of a network flow algorithm. We propose some heuristics for improving the quality of the algorithm. Our empirical results show that the resulting algorithm works reasonably well in practice.

2 NP-Hardness Proof

In this section, we prove that the MSE problem is NP-hard. The proof is by a reduction from the Set Cover problem. The decision version of Set Cover is defined as follows: Given $\langle X, C, \ell \rangle$, where X is a finite set of elements, C is a collection of subsets of X, and ℓ is an integer, is there a subset $C' \subseteq C$ with $|C'| \leqslant \ell$ such that the member elements of C' cover X?

Theorem 2. *The MSE problem is NP-hard.*

Proof. We prove that the following decision version of MSE is NP-complete: Given $\langle G, k, h \rangle$, where G is a graph with two distinguished nodes s and t, and $k, h \in \mathbb{N}$ are two numbers, is there a set P of k paths from s to t such that the number of edges shared among paths in P is at most h? It is easy to see that MSE is in NP. A certificate for this problem composed of k paths from s to t, and a certifier can then, in polynomial time, verify whether the number of shared edges is at most h.

We reduce Set Cover to MSE, by transforming each instance $\langle X, C, \ell \rangle$ of Set Cover to an instance $\langle G, k, h \rangle$ of MSE. The transformation is as follows. We first add to G the set of nodes $V = V_X \cup V_C \cup \{t\}$, where $V_X = \{v_x \mid x \in X\}$ and $V_C = \{v_{C_i} \mid C_i \in C\}$. We connect every node $v_x \in V_X$ to a node $v_{C_i} \in V_C$ by a directed edge if $x \in C_i$. Moreover, we connect every node $v_{C_i} \in V_C$ by a directed edge to t. Additionally, we add a node s to G and connect it to every other node $v \in V_X \cup V_C$ using a path of size $\ell + 1$. We call each of these paths a *chain*. Figure 2 illustrates our construction on a sample instance of Set Cover. We complete the transformation by setting $k = |X| + |C|$ and $h = \ell$.

Suppose that there is a set P of k (s,t)-paths in G with at most h shared edges. We show that there exists a collection $C' \subseteq C$ with $|C'| \leq \ell$ that covers X. It is easy to observe that each chain appears in at most one (s,t)-path, because otherwise more than h $(= \ell)$ edges would be shared. Since the outdegree of s is equal to the number of paths, k, it follows that each chain is used exactly once, and thus, each vertex $v_x \in V_X$ appears in exactly one (s,t)-path. Therefore, only one outgoing edge from each $v_x \in V_X$ is used in P, and hence, shared edges are only among those incident to t. Now, let $V' = \{v \in V_C \mid (v,t) \text{ is a shared edge}\}$. Consider a (s,t)-path that goes through a node $v_x \in V_X$ and a node $v \in V_C$. We claim that $v \in V'$. Otherwise, node v is incident to two paths, one coming from

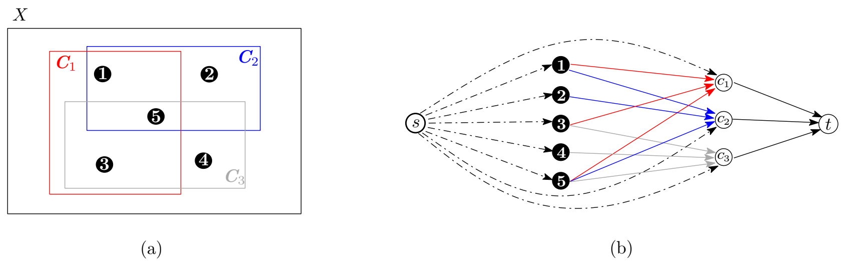


Fig. 2. (a) An instance of the Set Cover problem, with a covering set $\{C_2, C_3\}$. (b) Reduction from Set Cover to MSE. Dashed lines represent chains of size $\ell + 1$.

v_x and the other coming from s via a chain, causing the edge (v, t) to be used in at least two paths; a contradiction. Therefore, in the induced subgraph $G[P]$, each node $v_x \in V_X$ is connected to a node $v \in V'$. The set $C' = \{C_i \,|\, v_{C_i} \in V'\}$ is thus a covering of X with $|C'| = \ell$.

Conversely, let $C' \subseteq C$ be a covering of X with $|C'| \leq \ell$. We show that in the corresponding graph G, there is a set P of k paths with at most h shared edges. Let $V' = \{v_{C_i} \in V_C \,|\, C_i \in C'\}$. For each $x \in X$, we define a (s,t)-path P_x as follows. We start from s and follow the chain to v_x. Since x is covered by a collection $C_i \in C'$, there is an edge (v_x, v_{C_i}) such that $v_{C_i} \in V'$. So, we use the edge (v_x, v_{C_i}) to reach from v_x to v_{C_i}, and then proceed to t. The set $P_X = \{P_x \,|\, x \in X\}$ consists of $|X|$ (s,t)-paths. Now, we define a set P_C of $|C|$ (s,t)-paths by concatenating, for each $C_i \in C$, the chain from s to v_{C_i} and the edge (v_{C_i}, t). Let $P = P_X \cup P_C$. It is easy to observe that only edges between V_C and t can be used in more than one path of P. Since nodes in $V_C \setminus V'$ are not touched by the paths in P_X, each edge (v, t) for $v \in V_C \setminus V'$ is used exactly once in P, and hence, the number of shared edges in P is at most $|V'| = h$. $\square$

3 Approximation Algorithm

In this section, we provide an approximation algorithm for the minimum shared edges problem by transforming it to a network flow problem, called "Minimum Edge-Cost Flow". The problem definition is as follows.

Problem 3 (Minimum Edge-Cost Flow (MECF)). Given a graph $G = (V, E)$ with a capacity $u(e) \in \mathbb{Z}^+$ and a cost $c(e) \in \mathbb{Z}_0^+$ associated to each edge $e \in E$, find an integral flow f of value F from a source node s to a destination node t such that the total cost of edges sending non-zero flow, i.e., $\sum_{e \in E, f(e)>0} c(e)$, is minimized.

It is known that the MECF problem is NP-hard [6]. Krumke *et al.* [10] have provided an F-approximation algorithm for the MECF problem. We use their technique to obtain a $(k-1)$-approximation algorithm for MSE. The following lemma provides the ingredient.

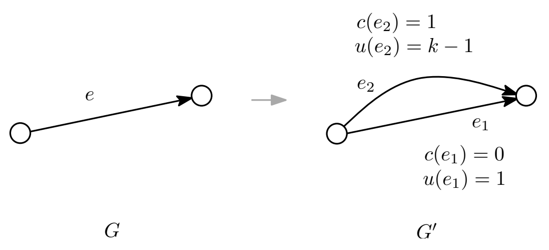


Fig. 3. Transforming an edge in MSE to two edges in MECF.

Lemma 4. *MSE can be reduced to MECF.*

Proof. We transform each instance of MSE on a graph $G = (V, E)$ to an instance of MECF on a graph $G' = (V', E')$. The transformation is as follows. We set $V' = V$, and for every edge $e \in E$, we add two edges e_1 and e_2 to E' with $u(e_1) = 1$, $c(e_1) = 0$, $u(e_2) = k - 1$ and $c(e_1) = 1$ (see Figure 3). Any solution of cost ℓ for MECF on G' corresponds to $k = F$ paths in G with ℓ shared edges. To see this, consider the set of edges that have positive flow and cost 1 in a solution for MECF on G'. The corresponding edges in G are exactly those who are shared in a solution for MSE. Conversely, any solution of size ℓ for MSE on G corresponds to a solution of cost ℓ for MECF on G'. $\square$

By Lemma 4, any α-approximation algorithm for MECF immediately gives an α-approximation for MSE. In the following, we provide an approximation algorithm for instances of MECF with maximum edge capacities $k - 1$. Our algorithm is based on the solution for a well-known related problem, called Minimum-Cost Flow, defined as follows:

Problem 5 (Minimum-Cost Flow (MCF)). Given a graph $G = (V, E)$ with a capacity $u(e) \in \mathbb{Z}^+$ and a cost $c(e) \in \mathbb{Z}_0^+$ associated to each edge $e \in E$, find an integral flow f of value F from a source node s to a destination node t such that $\sum_{e \in E} c(e)f(e)$ is minimized.

Let $G' = (V', E')$ be a graph obtained from G using Lemma 4. We construct a graph G'' from G' by replacing the cost of each edge e in G'' by $c(e)/u(e)$ (the capacities of the edges remain the same). Let $\mathtt{OPT}'$ be the cost of an optimal solution to the MECF problem on G', and $\mathtt{OPT}''$ be the cost of an optimal solution to the MCF problem on G''. Let f be an integral flow that realizes an optimal cost $\mathtt{OPT}''$ on G''. Since capacities of the edges are the same in G' and G'', f is a valid flow in G' as well. Let $E^+ = \{e \in E' : f(e) > 0\}$. Then, using an argument similar to what is used in [10] we get

$$\begin{aligned}\mathtt{OPT}' = \sum_{e \in E^+} c(e) &= \sum_{e \in E^+} u(e)\frac{c(e)}{u(e)} \\ &\leqslant (k-1)\sum_{e \in E^+} \frac{c(e)}{u(e)}\end{aligned}$$

$$\leqslant (k-1)\sum_{e\in E'}\frac{c(e)}{u(e)}f(e) \;=\; (k-1)\texttt{OPT}'',$$

where the first inequality holds because the capacity of the edges are at most $k-1$ by our construction of G' in Lemma 4, and the second inequality holds because $f(e) \geqslant 1$ for all edges in E^+. Now, if $\texttt{OPT}$ is the cost of an optimal solution to MSE on G, combined with Lemma 4 we get

$$\texttt{OPT} = \texttt{OPT}' \leqslant (k-1)\texttt{OPT}''.$$

Therefore, any optimal algorithm for MCF yields a $(k-1)$-approximation algorithm for MSE. There are a number of efficient algorithms for the MCF problem. The best one for our setting is an algorithm due to Ahuja *et al.* [1] that runs in $O(nm\log(nC)\log\log U)$ time, where n, m, C, and U are the number of nodes, number of edges, maximum edge cost, and maximum edge capacity, respectively. Since in our transformation $C=1$ and $U=k-1$, we get the following result.

Theorem 6. *There is a $(k-1)$-approximation algorithm for the MSE problem that runs in $O(nm\log n\log\log k)$ time.*

On series-parallel graphs, a fully polynomial time approximation scheme is given for the MECF problem in [10]. It leads to a $(1+\varepsilon)$-approximation algorithm for the MSE problem on series-parallel graphs, with a running time of $O(m^3(1+1/\varepsilon)\log k)$.

Remark The MECF problem is listed in Garey and Johnson's book ([6], Problem [ND32]) as an NP-complete problem, leaving the proof to an unpublished work by Even and Johnson. As a by-product, Theorem 2 and Lemma 4 together provide a simple proof for the NP-completeness of MECF.

4 Inapproximability Result

In the previous section, we provided a $(k-1)$-approximation algorithm for the MSE problem. It is natural to ask if this is the best approximation factor one can achieve. In this section, we prove a lower bound on the approximability of the problem. The proof is based on the following theorem from [5] (here, n refers to the number of nodes in the input graph).

Theorem 7 (Even *et al.* [5]). *The MECF problem with uniform edge-costs does not admit a $2^{\log^{1-\varepsilon} n}$-ratio approximation, for any constant $\varepsilon > 0$, unless $\mathrm{NP} \subseteq \mathrm{DTIME}(n^{\mathrm{polylog}\, n})$. This hardness holds even if only two edge capacity values are allowed, namely, $u(e) \in \{1, \mathrm{poly}(n)\}$, for every e.*

We establish an analogous hardness result for our problem, using an approximation preserving reduction from MECF with uniform edge-costs to MSE. The reduction is provided below.

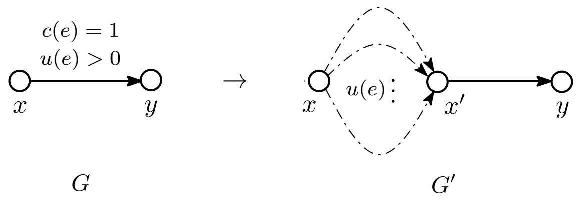


Fig. 4. Conversion of an edge in MECF with uniform edge-costs to an edge component in MSE. Dashed lines represent chains of length $|E| + 1$.

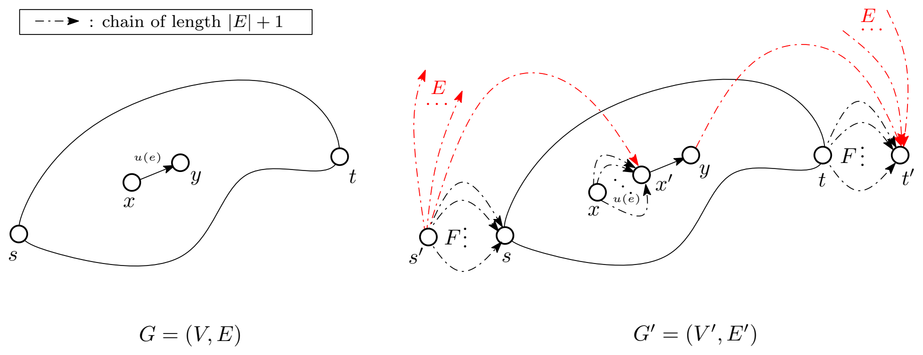


Fig. 5. Reduction from MECF with uniform edge-costs to MSE.

Theorem 8. *The MSE problem admits no* $2^{\log^{1-\varepsilon} n}$*-ratio approximation, for any* $\varepsilon > 0$*, unless* $\mathrm{NP} \subseteq \mathrm{DTIME}(n^{\mathrm{polylog}\, n})$.

Proof. Let $\mathcal{P}$ be the problem of finding a (s, t)-flow f of value F and cost C on a graph $G = (V, E)$ (see Problem 3). Each edge $e \in E$ is associated with an integer capacity $u(e) \in \{1, \mathrm{poly}(n)\}$ and a uniform cost $c(e) = 1$. We construct a graph $G' = (V', E')$ from $G = (V, E)$ as follows. For each node $x \in V$ we insert a corresponding node x in V'. For each edge $e = (x, y) \in E$, we add an "edge component" between x and y in G', as depicted in Figure 4. Each edge component is composed of $u(e)$ parallel chains from x to a newly-added node x', and a directed edge from x' to y. Each chain is composed of $|E|+1$ directed edges. We denote the edge component corresponding to an edge (x, y) by (x, x', y), and refer to chains connecting x to x' as type-1 chains. Additionally, we add two nodes s' and t', and connect s' to s and t to t' with F chains. We call these chains type-2 chains. Finally, we add for each edge component (x, x', y), a chain from s' to x' and a chain from y to t'. We refer to these chains as type-3 chains. The resulting graph is illustrated in Figure 5.

Let $\mathcal{P}'$ be the problem of finding $k = |E| + F$ (s', t')-paths in $G' = (V', E')$ with $S \leqslant |E|$ shared edges. We show that solutions to $\mathcal{P}$ and $\mathcal{P}'$ are in one-to-one correspondence. First, we show that every solution to $\mathcal{P}'$ is a solution to $\mathcal{P}$. A solution to $\mathcal{P}'$ is a set P of k (s', t')-paths in G' with $S \leqslant |E|$ shared edges. Observe that none of the chains in G' can be on more than one path of P, otherwise, the number of shared edges exceeds $|E|$. Since the out-degree of s' in G' is $k = |E| + F$, each chain incident to s' must be on exactly one path of P. Similarly, each chain incident to t' is on exactly one path of P. Therefore, each edge (x', y) of an edge component (x, x', y) of G' is used in at least one path of P. Moreover, the only edges that can be shared among paths of P are these (x', y) edges.

Now, view P as a flow f' of value $|E|+F$ from s' to t'. We convert f' to a flow of value F in G as follows. First, for each path p of the form $s' \dashrightarrow x' \to y \dashrightarrow t'$, where $\dashrightarrow$ represents a chain, we remove a flow of value 1 along p from f'. After this step, f' has value F, and each type-3 chain has flow zero. Thus, we can remove type-3 chains from the graph. For each edge component (x, x', y) corresponding to an edge e of E, the remaining flow on (x', y) is at most $u(e)$. Since the whole flow of x' now comes from x and continues to y, we can contract type-1 chains in between, and replace the edge component (x, x', y) by a single edge (x, y) of capacity $u(e)$, carrying the same amount of flow previously carried by (x', y). Similarly, we can contract type-2 chains and merge s' to s and t' to t. The resulting graph is isomorphic to G, and the new flow f' corresponds to a feasible (s, t)-flow of value F in G. Observe that edges having positive flow in the new f' are exactly those edges having flow greater than 1 in the original f', and thus correspond to edges shared in P. As the cost of each corresponding edge in G is one, the total cost of flow f' is equal to the number of shared edges, namely $C = S$.

By reversing the above process, we can show that every solution to $\mathcal{P}$ is also a solution to $\mathcal{P}'$. Therefore, the one-to-one correspondence follows. The constructed graph G' has size $O(|V| + |E|(F + \sum_{e \in E} u(e)))$. Recall that $u(e) \in \{1, \text{poly}(n)\}$. Moreover, $F \leqslant |V|^2$ in the construction used in [5]. The reduction is thus polynomial, and the theorem statement follows. □

The lower bound proved in Theorem 8 is stated in terms of n. Since k is unbounded in the original definition of MSE, we cannot directly use the above theorem to get a lower bound in terms of k. The following lemma, however, enables us to bound the value of k, and get an analogous lower bound.

Lemma 9. *If $k > |E|$, then the minimum number of shared edges is equal to the size of the shortest (s, t)-path.*

Proof. It is easy to see that in any set of k paths, for $k > |E|$, there is a path whose all edges are shared. Because, otherwise, each path needs to have at least one edge different from other paths, requiring more than $|E|$ edges, which is impossible. □

Lemma 9 implies that the MSE problem is polynomially solvable on instances with $k > |E|$. Therefore, we can simply assume that $k \leqslant |E| = O(n^2)$. Theorem 8 thus implies a lower bound of $2^{\log^{1-\varepsilon} k}$ on the approximability of MSE.

5 Heuristic Improvements

In this section, we discuss some heuristics for improving the quality of the $(k-1)$-approximation algorithm described in Section 3. Experimental results from implementing the heuristics are also presented and compared.

5.1 Successive Cost Update

The approximation algorithm described in Section 3 is based on running a minimum-cost flow (MCF) algorithm, and returning the obtained flow as a $(k-1)$-approximation to MECF, which in turn, gives a $(k-1)$-approximation to MSE.

The MCF algorithm receives a transformed graph in which each edge has a cost in the set $\{0, 1/(k-1)\}$. When it comes to using an edge of cost $1/(k-1)$, the additive cost of selecting an edge which is not sending any flow is the same as that of an edge that is currently sending a positive flow. Given that a positive flow on an edge of cost $1/(k-1)$ corresponds to a shared edge in the original graph G, it follows that in the $(k-1)$-approximation algorithm, there is no preference in reusing a previously shared edge rather than sharing a fresh edge. Our first heuristic attempts to force the approximation algorithm to reuse edges previously used in the solution.

We implement this heuristic by an iterative cost update method. To encourage the MCF algorithm to reuse a previously shared edge, we select at each iteration an edge with maximum flow among edges that have positive cost, update the cost of that edge to zero, and re-run the MCF algorithm. It is easy to observe that this cost update operation does not affect the approximation factor of the algorithm. Details of the heuristic are provided in Algorithm 1.

Algorithm 1 MSE-APPROX(G, k, s, t)

```
 1: construct $G'$ from $G$ using Lemma 4
 2: obtain $G_0$ from $G'$ by updating the cost of each edge $e$ to $c(e)/u(e)$
 3: compute a minimum-cost $(s,t)$-flow $f$ of value $k$ in $G_0$
 4: set $i = 0$
 5: while cost of $f \neq 0$ do
 6:     find an edge $e$ with a maximum flow among positive-cost edges in $G_i$
 7:     obtain $G_{i+1}$ from $G_i$ by updating the cost of $e$ to zero
 8:     compute a minimum-cost $(s,t)$-flow $f$ of value $k$ in $G_{i+1}$
 9:     $i \leftarrow i + 1$
10: return $i$
```

5.2 Shortest Path Bound

The second heuristic is based on the fact that the minimum number of shared edges in the MSE problem is bounded from above by the number of edges in a shortest path from s to t. Let p be the size of a shortest (s,t)-path. If a feasible solution to MSE consisting of k (s,t)-paths uses more than p edges, we can reroute all k paths through a shortest path, and reduce the number of shared edges to p. We use this tweak on top of Algorithm 1 to obtain the second heuristic as shown in Algorithm 2.

Algorithm 2 MSE-APPROX2(G,k,s,t)

```
1: let $r =$ MSE-APPROX$(G,k,s,t)$
2: let $p =$ size of a shortest $(s,t)$-path in $G$
3: return min$(r,p)$
```

5.3 Random Heuristics

Uniform Random In the first heuristic algorithm (Algorithm 1) we select an edge with a maximum flow at each iteration, and update its cost to encourage the MCF algorithm to reuse that edge in the next round. One can easily see that this greedy choice might not be necessarily the best. In our first random heuristic, we turn Algorithm 1 to a random one by changing line 6 as follow:

```
6:    pick an edge $e$ uniformly at random form positive-cost edges in $G_i$
```

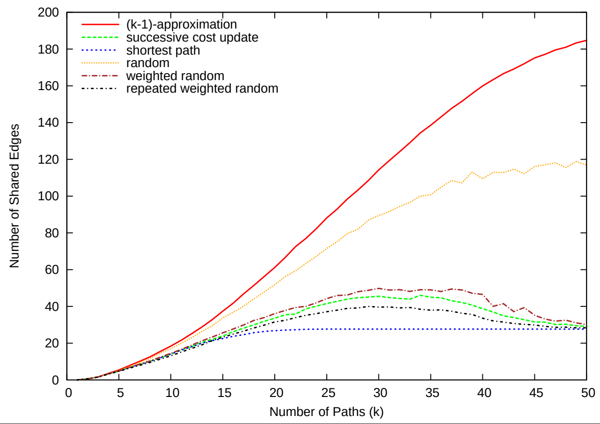
Weighted Random In this heuristic, we use a weighted random approach for the edge selection criteria. Here, to each edge e, we assign a weight $w(e)$ equal to the number of times e is used in (s,t)-paths (i.e., the current value of $f(e)$ in Algorithm 1). Let W be the sum of the weights of all edges. As opposed to the first random heuristic, at each iteration of the algorithm, we pick an edge e at random with probability $w(e)/W$. Therefore, edges used in more (s,t)-paths are more likely to be selected.

Repeated Weighted Random The last random heuristic is the Repeated Weighted Random heuristic, in which, we run the Weighted Random heuristic multiple times and report the minimum as the number of shared edges.

5.4 Experimental Results

We implemented the $(k-1)$-approximation algorithm described in Section 3 as well as the five heuristics described in this section. We evaluated our code on two families of graphs: road networks for large cities, and networks produced by benchmark graph generators. Figures 6 and 7 summarize the results of running our code on the two sample graphs: a road network for the city of Rome[1], and

[1] The graph is available at: `http://www.dis.uniroma1.it/~challenge9/data/rome/rome99.gr`



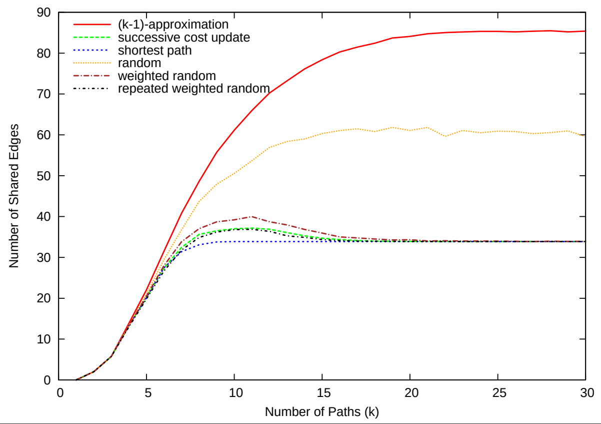
k	$(k-1)$-approx	successive cost update	shortest path	random	weighted random	repeated weighted random
1	0.0	0.0	0.0	0.0	0.0	0.0
2	0.6	0.6	0.6	0.6	0.6	0.6
3	1.6	1.6	1.6	1.6	1.6	1.6
4	3.57	3.31	3.31	3.28	3.31	3.19
5	5.44	4.81	4.81	5.1	4.88	4.73
6	7.77	6.72	6.72	7.24	6.77	6.35
7	10.11	8.43	8.43	9.44	8.59	7.95
8	12.61	10.16	10.16	11.7	10.34	9.57
9	15.5	12.05	12.05	14.66	12.37	11.32
10	18.43	14.09	14.09	16.97	14.5	13.14
11	21.58	16.21	16.21	19.94	16.62	15.19
12	25.1	18.17	18.04	22.95	18.92	17.11
13	28.79	20.47	20.01	26.37	21.3	19.1
14	32.88	22.12	21.39	29.48	23.4	21.45
15	37.39	23.93	22.55	33.75	25.61	23.22

Fig. 6. Empirical results for the road network of Rome.

a random directed graph from DARPA HPCS SSCA#2 graph theory benchmark [2][2]. A SSCA#2 graph is a representative of computations in the field of national security, scientific computing, and computational biology. Both test graphs have 3350 nodes and 8870 edges.

The algorithms are run for $k = 1$ to 50, and the average number of shared edges are reported for 100 randomly-picked pairs of source and destination nodes. To force random pairs to be far enough, we discarded pairs of source and destination nodes that were less than $\sqrt{n}/4$ edges apart, for n being the number of nodes. We used High Performance Computing Virtual Laboratory (HPCVL)'s

[2] The generator is available at: `https://sdm.lbl.gov/∼kamesh/software/GTgraph/`



k	$(k-1)$-approx	successive cost update	shortest path	random	weighted random	repeated weighted random
1	0.0	0.0	0.0	0.0	0.0	0.0
2	2.01	2.01	2.01	2.01	2.01	2.01
3	5.75	5.75	5.75	5.75	5.75	5.75
4	13.83	13.3	13.3	13.34	13.25	12.97
5	22.08	20.18	20.18	21.07	20.54	19.85
6	31.55	27.45	27.44	29.5	28.0	26.71
7	40.82	32.43	31.42	36.77	33.83	31.79
8	48.61	35.63	33.08	43.69	37.03	34.88
9	55.75	36.51	33.78	47.94	38.7	36.2
10	61.14	37.01	33.88	50.56	39.2	36.81

Fig. 7. Empirical results for a SSCA benchmark graph.

Beowulf Cluster[3] that has 64 nodes of 4×2.2 GHz Opteron Cores with 8 GB RAM for running the experiments in parallel.

As can be seen in Figures 6 and 7, our heuristics perform significantly better compared to the original $(k-1)$-approximation algorithm. (For the sake of clear comparison, numerical data is provided for smaller values of k.) For large enough values of k, we get an improvement of 50% to 85% in the number of shared edges in these two graphs. The second heuristic performs better than the first one for some range of k. However, the two heuristics eventually converge for k sufficiently large. The reason for this convergence is that when the number of paths, k, is large, it is more likely for the edges on shortest paths to be shared in more paths, and thus, be selected by Algorithm 1 for cost update. For smaller values of k, the Repeated Weighted Random heuristic gives better results compared to other heuristics. The results shown here are for 10 repetitions. Better results can be obtained by increasing the number of repetitions.

[3] The information is available at: `http://www.hpcvl.org/hpc-env-beowulf-cluster.html`

6 Conclusions

In this paper, we studied the complexity of the minimum shared edges problem, and showed that the problem admits no $2^{\log^{1-\varepsilon} k}$-factor approximation, for any constant $\varepsilon > 0$. Moreover, we presented a $(k-1)$-approximation algorithm for the problem, and proposed some heuristics to improve it in practice. Heuristics presented in Section 5 (except the second one), can be indeed used as a practical algorithm for the MECF problem.

Whether or not the minimum shared edges problem is NP-complete, for small values of k (e.g., $3, 4, \ldots$), remains a topic for future investigation. An interesting open problem is to see if an algorithm with an approximation ratio better than $k-1$ exists for the minimum shared edges problem. Although our lower bound in Section 4 eliminates the possibility of having a poly-logarithmic approximation factor, we have not ruled out the possibility of having an approximation factor of $O(n^c)$, for a constant $c < 1$. (For example, see [4] for two variants of the Label Cover problem for which the same hardness of $2^{\log^{1-\varepsilon} n}$ holds, yet they admit a $O(n^{1/3})$-factor approximation.) Improving the lower bound on the approximability is another open problem.

Acknowledgments The authors would like to thank Anil Maheshwari and Peter Widmayer for helpful discussions.

References

1. R. K. Ahuja, A. V. Goldberg, J. B. Orlin, and R. E. Tarjan. Finding minimum-cost flows by double scaling. *Mathematical Programming*, 53(1):243–266, 1992.
2. D. Bader and K. Madduri. Design and implementation of the HPCS graph analysis benchmark on symmetric multiprocessors. In *Proc. 12th Internat. Conf. High Perform. Comput.*, volume 3769 of *Lecture Notes Comput. Sci.*, pages 465–476. 2005.
3. D. A. Castanon. Efficient algorithms for finding the k best paths through a trellis. *IEEE Trans. Aerospace and Electronic Systems*, 26(2):405–410, 1990.
4. M. Charikar, M. Hajiaghayi, and H. Karloff. Improved approximation algorithms for label cover problems. In *Proc. 17th Annu. European Sympos. Algorithms*, volume 5757 of *Lecture Notes Comput. Sci.*, pages 23–34. 2009.
5. G. Even, G. Kortsarz, and W. Slany. On network design problems: fixed cost flows and the covering steiner problem. *ACM Trans. Algorithms*, 1(1):74–101, 2005.
6. M. Garey and D. S. Johnson. *Computers and intractability : A guide to the theory of NP-completeness.* W. H. Freeman, 1979.
7. A. V. Goldberg and S. Rao. Beyond the flow decomposition barrier. *J. ACM*, 45(5):783–797, 1998.
8. A. Itai, Y. Perl, and Y. Shiloach. The complexity of finding maximum disjoint paths with length constraints. *Networks*, 12(3):277–286, 1982.
9. Y. Kobayashi and C. Sommer. On shortest disjoint paths in planar graphs. *Discrete Optimization*, 7(4):234–245, 2010.
10. S. O. Krumke, H. Noltemeier, S. Schwarz, H.-C. Wirth, and R. Ravi. Flow improvement and network flows with fixed costs. In *Proc. Internat. Conf. Oper. Res.: OR-98*, pages 158–167, 1998.

11. S.-W. Lee and C.-S. Wu. A k-best paths algorithm for highly reliable communication networks. *IEICE Trans. Commun.*, E82-B(4):586–590, 1999.
12. C. Li, S. T. McCormick, and D. Simchi-Levi. Finding disjoint paths with different path-costs: Complexity and algorithms. *Networks*, 22(7):653–667, 1992.
13. C. Li, T. S. McCormick, and D. Simich-Levi. The complexity of finding two disjoint paths with min-max objective function. *Discrete Appl. Math.*, 26(1):105–115, 1989.
14. S. D. Nikolopoulos, A. Pitsillides, and D. Tipper. Addressing network survivability issues by finding the k-best paths through a trellis graph. In *Proc. 16th IEEE Internat. Conf. Comput. Commun.*, pages 370–377, 1997.
15. J. W. Suurballe and R. E. Tarjan. A quick method for finding shortest pairs of disjoint paths. *Networks*, 14(2):325–336, 1984.
16. D. Xu, Y. Chen, Y. Xiong, C. Qiao, and X. He. On the complexity of and algorithms for finding the shortest path with a disjoint counterpart. *IEEE/ACM Trans. Networking*, 14(1):147–158, 2006.
17. S. Q. Zheng, B. Yang, M. Yang, and J. Wang. Finding minimum-cost paths with minimum sharability. In *Proc. 26th IEEE Internat. Conf. Comput. Commun.*, pages 1532–1540, 2007.