On Dynamic Algorithms for Minimum Cut and Other Source-Sink Problems

Timothy G. Abbott, You Zhou
tabbott@mit.edu, yoz@mit.edu
MIT Computer Science and Artificial Intelligence Laboratory

July 5, 2007

Abstract

This paper shows that dynamic algorithms for many problems with a source-sink structure give rise to static algorithms for the all-pairs version of the problem. In the case of the s-t minimum cut problem, this technique gives a strong hardness relation between dynamically maintaining the s-t minimum cut in a weighted graph and computing the Gomory-Hu tree.

1 Introduction

Given an efficient, partially dynamic (incremental or decremental) algorithm for the weighted single-source shortest paths problem, Roditty and Zwick [RZ04a] showed that one can construct an efficient algorithm for the static weighted all-pairs shortest paths problem. Combined with an $\Omega(mn)$ lower bound for the static all-pairs shortest paths problem for path comparison algorithms [KKP93], this provides a lower bound on the runtime of any partially dynamic single-source shortest paths structure based on classic path comparison algorithms such as Dijkstra's algorithm.

In this paper, we show that Roditty and Zwick's result is in fact part of a more general phenomenon. We present an analogue for the dynamic s-t minimum cut problem, relating the hardness of the weighted dynamic s-t minimum cut problem to the hardness of computing the Gomory-Hu tree, a structure that compactly encodes the all-pairs minimum cut.

The Gomory-Hu tree has a node for each vertex in the graph G and each edge in the tree corresponds to a minimum cut separating the corresponding pair of vertices of G. The Gomory-Hu tree has the property that the minimum cut between vertices s and t in G is the cut corresponding to the minimum weight edge along the path from s to t in the Gomory-Hu tree.

The algorithm of Gomory and Hu computes the Gomory-Hu tree using $n-1$ calls to an algorithm for the s-t minimum cut problem; the computations were done on graphs similar to the graph G, but with some vertices combined into supernodes [GH61]. Gusfield showed that the recursive mincuts can in fact be computed using $n-1$ s-t minimum cuts on the original graph and $O(n^2)$ additional computation [Gus90].

For unweighted graphs, an $\tilde{O}(mn)$ algorithm for constructing Gomory-Hu trees has been recently discovered, far faster than using n s-t minimum cuts [BHKP07]. However, all known algorithms for computing the Gomory-Hu tree in a weighted graph make $O(n)$ calls to an algorithm for the s-t minimum cut problem (which uses an s-t max flow algorithm). Thus, advances in the running time for computing a Gomory-Hu tree in a weighted graph have been corollaries of more efficient algorithms for the maximum flow problem. An important open question is whether one can compute the Gomory-Hu tree more efficiently than making $O(n)$ calls to an s-t maximum flow algorithm in the weighted case.

In this paper, we prove that modulo a minor technical condition, one of two interesting things must happen. Either there are more efficient algorithms solving the Gomory-Hu problem than computing n s-t minimum cuts (resolving a question that has been open for nearly 50 years), or there are no algorithms for dynamically maintaining the s-t minimum cut in a weighted graph that are more efficient than the obvious algorithm of recomputing the cut from scratch on each query. We achieve this with an explicit hardness relation showing that the problem of dynamically maintaining the s-t minimum cut is at least as hard as the problem of computing the Gomory-Hu tree. We prove a similar hardness relation for partially dynamic algorithms maintaining the s-t minimum cut on a directed graph.

Given that the problem of computing the Gomory-Hu tree more efficiently has remained open for so long, we believe the more likely possibility is that dynamically maintaining the s-t minimum cut (and consequently, the s-t max flow) is hard. Our work leaves open the possibility of efficient algorithms for dynamically maintaining the all-pairs s-t minimum cut of a weighted graph, in analogy to the shortest paths case.

This paper also describes several hardness results for dynamic algorithms solving other problems with a source-sink structure, including resistance in a graph, the s-t shortest edge-disjoint round trip problem, and the minimum cost maxflow problem.

Throughout this paper, we will assume that all graphs are weighted graphs (unless explicitly described as unweighted) and all weights are non-negative. On a graph with n vertices and m edges, we will for simplicity assume that $n = O(m)$, as is the case in a connected graph.

2 Hardness for Dynamic Single-Source Shortest Paths

In this section, we present Roditty and Zwick's result about the hardness of a partially dynamic single-source shortest path algorithm [RZ04a], to provide appropriate context for our related results.

Let A be an incremental algorithm for the single-source shortest paths problem. Let $\text{init}_A(m,n)$ be the initialization time, $\text{query}_A(m,n)$ be the amortized query time, and $\text{update}_A(m,n)$ be the amortized update time for A on a graph with m vertices and n edges. We assume all these time functions are monotonic and that they are bounded by a polynomial in the two inputs.

Theorem 1 *Given an algorithm A for the incremental (or decremental) directed (or undirected) single-source shortest path problem, we can solve the static all-pairs shortest paths problem on a graph of n vertices and m edges, using $O(n^2 \cdot \text{query}_A(m,n) + n \cdot \text{update}_A(m,n) + \text{init}_A(m,n))$ time.*

Proof: We consider an incremental algorithm first. Pick $W > n \cdot \text{maxweight}(E)$. Suppose $V = \{1, 2, \ldots, n\}$. Construct a graph G' with vertices $V \cup \{0\}$, and with edge set E, and initialize the incremental algorithm on it with source 0. Now, for $j = n, \ldots, 2, 1$, do the following. First, create a new edge $(0, j)$ of weight jW. Then, query the distances from 0 to i for all i. I claim that this distance is exactly $jW + D_G(j,i)$. A path which uses the edge $(0,j)$ will have this distance. A path which does not use this edge must use some $(0,k)$ for $k > j$, but the distance for edge $(0,k)$ is at least $(j+1)W > jW + D_G(j,i)$, by the definition of W. If i is unreachable from j, this will be detected, since then the distance returned will be at least $(j+1)W$. Over the n vertices, we obtain the all-pairs shortest paths in G using $O(n^2 \cdot \mathrm{query}_A(m+n, n+1) + n \cdot \mathrm{update}_A(m+n, n+1) + \mathrm{init}_A(m+n, n+1))$. We obtain the stated bounds by using that A is bounded by a polynomial and $n = O(m)$.

The decremental version can be found by time-reversing the construction. Obviously, this result also extends to a fully dynamic algorithm.

Notice that the argument is correct for both directed and undirected graphs, but that it does not immediately extend to the unweighted case. Roditty and Zwick also present reductions of the unweighted case to boolean matrix multiplication [RZ04a], but we will not discuss those results here. The best known algorithms for all-pairs shortest paths in weighted directed or undirected graphs have runtimes that are $\Omega(mn)$. In fact, Karger et al. showed that any path-comparison algorithm, using edge weights only to compare path weights, for the static all-pairs shortest paths problem has a runtime that is $\Omega(mn)$ [KKP93]. Thus, without finding a faster algorithm for the static all-pairs shortest paths problem, we cannot hope to do better than an $O(m)$ amortized total update and query time for a partially dynamic single-source shortest paths algorithm, without spending an unreasonable amount of time initializing the structure. Running Dijkstra's algorithm from scratch takes $O(m + n \log n)$ time if a Fibonacci heap is used, which is almost tight. It is thus unsuprising that recent progress in dynamic shortest paths algorithms has focused on the all-pairs shortest paths problem.

While Roditty and Zwick's result was about the Single-Source Shortest Paths problem, this problem is really a source-sink problem disguised by the fact that efficient algorithms for the s-t shortest path problem, like Dijkstra's algorithm, can be modified to solve the single-source shortest paths without increasing the asymptotic runtime.

3 Minimum Cuts

We will now discuss our analogous result for the s-t minimum cut problem. Given a fully dynamic algorithm A for the s-t minimum cut problem on directed or undirected graphs, with amortized query time $\mathrm{query}_A(m, n)$, amortized update time $\mathrm{update}_A(m, n)$, and initialization time $\mathrm{init}_A(m, n)$, we give an efficient algorithm for constructing the Gomory-Hu tree on an undirected graph G. In Section 3.1, we show the same result for partially dynamic algorithms, but only in the case of an s-t min cut algorithm that works on directed graphs (a property that known algorithms, based on max flows, possess). We assume all these time functions are monotonic and that they are bounded by a polynomial in the two inputs.

Our data structure should support updates of adding or deleting an edge, along with adding or deleting a vertex other than s and t which has no edges attached to it. We will define the s-t minimum cut query so that it returns the set defining one side of the cut, rather than simply the value of the cut or the edges cut. Our choice of convention here makes it clear that Gusfield's algorithm requires only $O(n^2)$ additional processing (a claim Gusfield does not make, since the $O(n^2)$ time is almost certainly dominated by the $O(n)$ minimum cut calls, but which is clear from the pseudocode from page 10 of [Gus90]).

Theorem 2 *If A is a fully dynamic algorithm for the s-t mincut problem on an undirected (or directed) graph, then with $O(n \cdot \mathrm{query}_A(m, n) + n \cdot \mathrm{update}_A(m, n) + \mathrm{init}_A(m, n) + n^2)$ time, we can compute the Gomory-Hu tree for an undirected graph G with n vertices and m edges.*

Proof: Let W be a number larger than the sum of all the edge weights in G. We construct a graph G' with two extra vertices, S and T, which have no edges attached to them. We first show how to compute the static s-t minimum cut in the graph G. We do updates adding edges (S, s) and (t, T) each of weight W to G', and query the mincut. Since the edges we added have weight greater than that of any cut in G, the min cut in G' does not cut either of the two new edges. Thus, it must be an s-t minimum cut in the original graph G. We then restore our graph to the original graph G' by doing updates to remove the two edges we added. This process allows us to query the s-t minimum cut for any vertices s and t in G using only $O(1)$ queries and $O(1)$ updates to our initialized data structure.

We can now use this procedure as the s-t minimum cut subroutine required for Gusfield's algorithm [Gus90]. This allows us to compute the Gomory-Hu tree using $O(n)$ calls to the s-t minimum cut query subroutine we just described and $O(n^2)$ additional time. Thus, we can compute the Gomory-Hu tree using only $O(n^2 + n \cdot \text{update}_A(m+2n, n+2) + n \cdot \text{query}_A(m+2n, n+2) + \text{init}_A(m+2n, n+2)) = O(n^2 + n \cdot \text{update}_A(m, n) + n \cdot \text{query}_A(m, n) + \text{init}_A(m, n))$ time, using that A is bounded by a polynomial and that $n = O(m)$.

Corollary 3 *If A is a fully dynamic algorithm for the s-t mincut problem, then with $O(n \cdot \text{query}_A(m, n) + n \cdot \text{update}_A(m, n) + \text{init}_A(m, n) + n^2)$ time, we can compute the all-pairs minimum cut of an undirected graph G with n vertices and m edges.*

Proof: It suffices to show we can compute the all-pairs minimum cut from the Gomory-Hu tree in $O(n^2)$ time. Using the Gomory-Hu tree's property, we can find all the minimum cuts from a source s in an $O(n)$ time traversal of the tree. Repeating this for each vertex as the source, one can compute the all-pairs minimum cut using only $O(n^2)$ time beyond the time required to compute the Gomory-Hu tree.

We have just shown that in a strong sense, it is as hard to efficiently maintain the dynamic s-t minimum cut as it is to compute the Gomory-Hu tree. We can now prove the following surprising theorem, which relates the hardness of dynamically maintaining s-t minimum cuts to the hardness of computing them in the static case, under the assumption that current techniques for constructing Gomory-Hu trees are optimal.

Theorem 4 *For any fully dynamic algorithm A maintaining the s-t minimum cut, one of the following must hold:*

1. *The initialization time for A must be $\omega(n \cdot T(m, n))$, where $T(m, n) = \text{query}_A(m, n) + \text{update}_A(m, n)$ is the larger of the update time for A and the query time for A.*

2. *There are algorithms for computing the Gomory-Hu tree that are more efficient than $O(n)$ s-t minimum cuts.*

3. *A is no more efficient than computing the s-t minimum cut from scratch on each query.*

Since most efficient data structures do not require more than n updates worth of time to initialize, case 1 is a minor technical condition. Aside from this condition, this theorem implies that any dynamic algorithm for the s-t minimum cut problem more efficient than recomputing from scratch on each query gives rise to a new, more efficient kind of algorithm for the Gomory-Hu tree problem.

Proof: Suppose that we are not in case 1, so that the initialization time does not dominate n updates and n queries. Then the runtime of our Gomory-Hu tree algorithm becomes $O(n^2 + n \cdot T(m,n))$. Since a query on the data structure must return the actual nodes in the cut, we know that $T(m,n) = \Omega(n)$. It follows that the running time of our Gomory-Hu algorithm is in fact $O(n \cdot T(m,n))$.

Suppose further that we are not in case 2. Let $g(m,n)$ be the best running time for an algorithm B computing the s-t minimum cut in a graph with n vertices and m edges. The complement of case 2 implies that constructing a Gomory-Hu tree on a weighted graph with $\Omega(n)$ queries to B, using $\Omega(n \cdot g(m,n))$ time, is optimal. It follows that $T(m,n) = \Omega(g(m,n))$.

Now, since $T(m,n) = \text{query}_A(m,n) + \text{update}_A(m,n) = \Omega(g(m,n))$, we conclude that either the update time or the query time for A takes as long as computing the s-t minimum cut from scratch using algorithm B! Thus, we could obtain a dynamic algorithm of the same efficiency as algorithm A by using one of the following two strategies:

1. Run B on each update, caching the result. On queries, output the cached result.

2. Do nothing on updates, and run B upon each query, caching the result to handle several consecutive queries.

The second of these strategies is strictly superior to the first, since it runs algorithm B once for each time an update is followed by a query, while the first runs algorithm B on every update. Thus algorithm A is no more efficient than strategy 2, and we must be in case 3.

We can remove the technical condition on the initialization time for sparse graphs:

Corollary 5 *For any fully dynamic algorithm A maintaining the s-t minimum cut on sparse graphs (where $m = \Theta(n)$), one of the following must hold:*

1. *There are algorithms for computing the Gomory-Hu tree on sparse graphs that are more efficient than computing $O(n)$ s-t minimum cuts.*

2. *On sparse graphs, A is no more efficient than computing the s-t minimum cut from scratch on each query.*

Proof: The argument is the same as for Theorem 4, replacing the initialization process with a series of $O(m+n) = O(n)$ updates to construct the input graph.

Arikati et al. showed that one can efficiently compute the all-pairs minimum cut in sparse graphs when a certain topological property is small, but in general there are no known algorithms for computing the Gomory-Hu tree in sparse graphs more efficient than n times the best algorithms for max flow in sparse graphs [ACZ95].

We can similarly remove the technical condition on initialization times if we require the data structure to support *bulk updates*, in which a single operation supports adding or deleting a vertex along with all incident edges.

Corollary 6 *For any fully dynamic algorithm A maintaining the s-t minimum cut and supporting bulk updates, one of the following must hold:*

1. *There are algorithms for computing the Gomory-Hu tree that are more efficient than computing $O(n)$ s-t minimum cuts.*

2. *A is no more efficient than computing the s-t minimum cut from scratch on each query.*

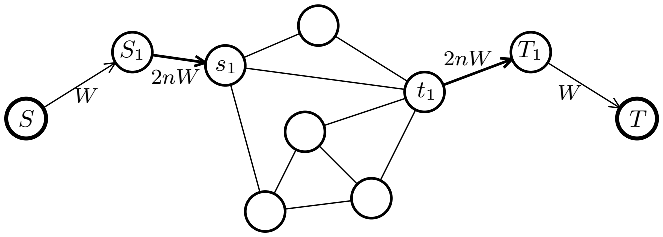

Figure 1: Query 1

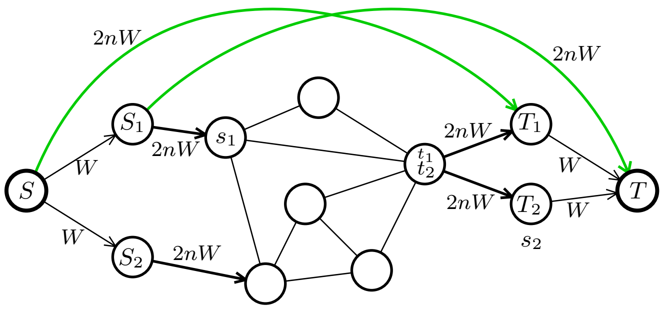

Figure 2: Query 2, wherein the first query is bypassed by edges from S to T_1 and from S_1 to T

Proof: Since one can create an arbitrary n-vertex graph is only $O(n)$ bulk updates, we can replace the initialization time with $O(n)$ bulk updates. The result then follows from the proof of Theorem 4.

3.1 Hardness of Partially Dynamic Minimum Cut Algorithms

We now present a similar reduction that works for partially dynamic (incremental or decremental) algorithms. We will suppose the algorithm is incremental; the decremental algorithm can once again be found by time-reversing the incremental algorithm.

Theorem 7 *Given an incremental (or decremental) algorithm for the s-t minimum cut problem on directed graphs, one can compute the Gomory-Hu tree (and thus solve the static all-pairs minimum cut problem) on undirected graphs using* $O(n \cdot \text{update}_A(m,n) + n \cdot \text{query}_A(m,n) + \text{init}_A(m,n) + n^2)$ *time.*

Proof: Using the same strategy as in the fully dynamic case, we need to show how to effectively query the s_i-t_i minimum cut n times. To do this, we work with an expanded graph with auxiliary vertices S_i and T_i for each s_i-t_i mincut query i, a supersource S, and a supersink T. For query i which is for the mincut from s_i to t_i, we do the following:

1. Add edges (S_i, s_i) and (t_i, T_i) of weight $2nW$, and edges (S, S_i) and (T_i, T) of weight W, where $W > n \cdot \text{maxweight}(E)$.

2. Query the minimum S-T cut.

3. Add bypass edges (S_i, T) and (S, T_i) of weight $2nW$.

We claim that the minimum S-T cut thus computed gives the minimum s_i-t_i cut for query i. First, consider the cut of this graph that cuts the edges (S, S_j) and (T_j, T), for $j < i$, and performs a minimum cut between s_i, t_i in graph G. This cut costs $2(i-1)W + \mathrm{cut}(s_i, t_i) < (2i-1)W$. Clearly, this is optimal among cuts that cut all the (S, S_j) and (T_j, T) edges for $j < i$.

The minimum cut must cut every S-T path, including the S–S_j–T and S–T_j–T paths for $j < i$ created in step 3. The bypass edges (S_j, T) and (S, T_j) for $j < i$, shown in figure 2, also have weight at least $2nW > (2i-1)W$, so they cannot be in a minimum cut. Thus, all the edges (S, S_j) and (T_j, T) for $j < i$ are forced to be cut in any minimum cut. Hence, a minimum cut of our graph equals $2(i-1)W$ plus a cut between S_i and T_i. Since the edges (S_i, s_i) and (t_i, T_i) each have weight greater than $(2n-1)W$, they can never be in a minimum cut. Consequently, any S-T minimum cut must have weight $2(i-1)W$ plus the weight of a s_i-t_i minimum cut in G, as desired. The corresponding minimum cut set must be the edges (S, S_j) and (T_j, T) for $j < i$, along with an s_i-t_i minimum cut set for G.

Our construction is effective even if the s_i and t_i are reused.

Thus, in finding the n s-t minimum cut queries we add $4n$ edges and use $3n+2$ vertices, with $O(n^2)$ additional computation as before. Thus we compute the Gomory-Hu tree using $O(n \cdot \mathrm{update}_A(m+4n, 3n+2) + n \cdot \mathrm{query}_A(m+4n, 3n+2) + \mathrm{init}_A(m+4n, 3n+2) + n^2) = O(n \cdot \mathrm{update}_A(m,n) + n \cdot \mathrm{query}_A(m,n) + \mathrm{init}_A(m,n) + n^2)$ time, as desired.

We had to assume that the algorithm works for directed graphs, since our edge-deletion mechanism requires that it be impossible to traverse the (S_i, s_i) in the reverse direction. However, the Gomory-Hu tree algorithm that we got out of it only works for an undirected graph G. This is in fact necessary because analogues for Gomory-Hu trees do not exist for directed graphs [Benc95].

This result has implications for the partially dynamic s-t max flow problem. All known algorithms for the weighted s-t minimum cut problem are based on algorithms for the max flow problem, which are effective on both directed and undirected graphs. It is easy to compute the minimum cut set from the maximum flow. Thus, our hardness result for the partially dynamic directed s-t min cut problems gives rise to a hardness relation between the partially dynamic s-t max flow problem and the Gomory-Hu problem. We can relate this hardness result back to the all-pairs max flow problem. Though no general transformation is known for computing the maximum flow given the minimum cut set, by the classic Max Flow-Min Cut Theorem [EFS56], the values are the same. Our result then implies that a partially dynamic algorithm maintaining the s-t max flow can compute the Gomory-Hu tree, and thus the static all-pairs maximum flow *value* on an undirected graph, using n updates, n queries, and $O(n^2)$ additional computation.

4 Other Problems with a Source-Sink Structure

The shortest paths and minimum cut problems present examples of cases where an efficient dynamic algorithm for a source-sink problem gives rise to an efficient algorithm for computing the all-pairs version of the problem. In this section, we discuss a number of other source-sink problems, and show that for each of them a similar result holds (often through a slightly different mechanism).

4.1 Minimum Cost Maxflow

Suppose that edges in a graph G have capacities and costs per unit flow. Consider the minimum cost maxflow problem of finding the maximal s-t flow in G that has the minimum cost among all such maxflows. We will assume that all costs are non-negative.

Theorem 8 *Given an algorithm A for the partially dynamic s-t min cost max flow problem, we can solve the static all-pairs min cost max flow problem on a graph G of n vertices and m edges, using n^2 queries, n^2 updates, and one initialization on a graph with $O(n)$ vertices and $O(m+n)$ edges.*

Proof: We can use the strategy from Theorem 7, to query for any pair (s,t), the s-t min cost max flow in graph G, using $O(1)$ updates and $O(1)$ queries. We simply need to replace $2nW$ with $2n^2W$ everywhere, and give the bypass edges, (S,S_i) edges, and (T_i,T) edges cost 0, and the (S_i,s_i) and (t_i,T_i) edges cost 1. By the proof of Theorem 7, the cut induced by the min cost max flow must be an s_i-t_i cut in G along with the edges (S,S_j) and (T_j,T) for $j<i$. What remains to show is that the min cost way to saturate those edges gives a minimum cost for G. But this is clear, since the paths (S,S_j,T) and (S,T_j,T) have cost 0, while any other path saturating the (S,S_j) and (T_j,T) edges must pass through an edge of cost 1. Thus only the flow corresponding to the ith query passes through vertices of G, resulting in a minimum cost max flow from s_i to t_i in G, as desired.

This process requires $O(1)$ updates, $O(1)$ queries. Repeating it for each pair (s,t) in the graph, we compute the all-pairs min cost max flow in the graph.

This result might seem weaker than the minimum cut or shortest paths case, since it requires $O(n^2)$ updates and queries to A. However, it is not clear how to compute the all-pairs min cost max flow in a graph without making $O(n^2)$ calls to an algorithm for the s-t min cost max flow problem (or equivalently, the min cost circulation problem). Thus, it is possible that this result gives a hardness relation between the partially dynamic s-t problem and the static all-pairs problem that is just as tight as in the cases we have discussed in detail.

4.2 Resistance

Consider the problem of computing the effective resistance between two vertices s and t in a graph, where each edge in the graph has a resistance equal to its weight. The effective resistance metric has been proposed as an alternative to the shortest paths for applications related to the strength or reliability of a connection [DR93]. This problem can be solved in polynomial time [BGX03] using determinants.

Theorem 9 *Given an algorithm A for the fully dynamic s-t resistance problem, we can solve the static all-pairs resistance problem on a graph G of n vertices and m edges, using n^2 queries, n^2 updates, and one initialization on a graph with $O(n)$ vertices and $O(m+n)$ edges.*

Proof: Augment the graph G with a supersource S and a supersink T, and run our dynamic algorithm on it. To query the resistance between vertices s and t of G, add edges of weight W between S and s, and one between t and T, and query the algorithm. The resistance between s and t in G can be computed by subtracting $2W$ from the value returned by the query. We can then restore the graph to its state before the query by deleting the edges that we added. This process requires $O(1)$ updates, $O(1)$ queries. Repeating it for each pair (s, t) in the graph, we compute the all-pairs resistance in the graph.

Similar comments to those on the min cost max flow problem apply here. Because resistance is an undirected whole-graph property, it does not seem possible to generalize this argument to partially dynamic algorithms.

4.3 Shortest Edge-Disjoint Round Trip

Consider the problem of finding the shortest loop passing through vertices s and t in an undirected graph that does not use any edge twice. We will call this problem the shortest edge-disjoint round trip problem.

This problem has an efficient s-t algorithm in the static case, as follows. First, run an s-t shortest paths algorithm. Then, taking the edges to be pairs of directed edges, reverse all the edges used in the path, and compute an s-t shortest path in the resulting graph. The end result is two paths between s and t which share no edges in common, and with the minimal total cost. This algorithm is equivalent to using one shortest augmenting path to find a minimum cost flow of 2 units from s to t with unit capacity edges.

Theorem 10 *Given an algorithm A for the incremental (or decremental) dynamic s-t shortest edge-disjoint round trip problem, we can solve the static all-pairs shortest edge-disjoint round trip problem on a graph G with n vertices and m edges, using one initialization, n^2 queries, n^2 updates to A running on a graph with $O(n)$ vertices and $O(m+n)$ edges.*

Proof: We use a supersource S and supersink T as before, and the technique of [RZ04a] to supersede old queries with new ones. For $i = n(n-1), \ldots, 2, 1$, with the s_i and t_i iterating over all pairs of distinct vertices, we add double edges from S to s_i and from t_i to T_i, each with weight iW, where $W > n \cdot \text{maxweight}(E)$, and calculate the shortest edge-disjoint round trip between S and T. The shortest edge-disjoint round trip between s_i and t_i will be this result minus $4iW$.

Note that this algorithm generalizes to the case of finding $k > 2$ edge-disjoint paths from s to t in a directed or undirected graph of minimum total weight.

5 Conclusions and Open Problems

We have shown that dynamic algorithms for many source-sink problems to solving the static all pairs version of those problems. In some cases, even partially dynamic algorithms suffice. Each of these results provides a hardness relation on the dynamic version of the source-sink problem.

While there is a strong commonality in the character of these hardness results for dynamic algorithms, the proofs for individual problems are all distinct arguments using the same core idea. We have given several **GIVE A NUMBER** distinct constructions in terms of what updates can be done to the graph in order to make a new s-t query and later to undo that action. We have seen cases where the argument extends trivially to partially dynamic algorithms (shortest paths), cases where the result is weakened slightly when extended to the partially dynamic case (minimum cuts), and cases where it seems to not extend at all (resistance). Further advances in studying one of these problems could yield insights for the others.

In the case of minimum cut, we have proven that unless there are superior techniques for computing Gomory-Hu trees than those known, there can be no efficient algorithms for maintaining the s-t minimum cut in a graph. Our hardness result for s-t minimum cut explains why there is almost no work on the problem of dynamically maintaining the s-t minimum cut, in excellent analogy with the case of shortest paths. However, while there has been a great deal of work on dynamic algorithms for the all-pairs shortest paths problem, we are not aware of any work on dynamically maintaining the all-pairs minimum cut or the Gomory-Hu tree in a weighted graph. Developing efficient algorithms for dynamically maintaining the Gomory-Hu tree (or showing that they cannot exist) is a good direction for future investigation.

Another natural direction for further research is lower bounds for the Gomory-Hu problem for weighted graphs. A lower bound for this problem would immediately imply a lower bound for dynamically maintaining an s-t minimum cut. It would also imply a lower bound for partially dynamic algorithms maintaining an s-t max flow.

6 Acknowledgements

The authors thank Brian Dean and Erik Demaine for helpful discussions.

References

[ACZ95] S. R. Arikati, S. Chaudhuri, and C. D. Zaroliagis, All-pairs min-cut in sparse networks, in Proc. 15th Conf. on Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Comput. Sci. 1026, Springer, Berlin (1995) 363–376.

[BGX03] R. B. Bapat, I. Gutman, W. Xiao: A Simple Method for Computing Resistance Distance. Zeitschrift Fur Naturforschung A, 2003, VOL 58; PART 9/10, pages 494-498.

[BHKP07] A. Bhalgat, R. Hariharan, T. Kavitha, D. Panigrahi. An $O(mn)$ Gomory-Hu Tree Construction Algorithm for Unweighted Graphs. STOC 2007.

[Benc95] A. Benczr. Counterexamples for directed and node capacitated cut-trees. Siam Journal on Computing 24(3):505-510 (1995)

[CLRS] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, second edition, 2001.

[DI04] C. Demetrescu and G. Italiano. A new approach to dynamic all-pairs shortest paths. Journal of the ACM (JACM), Volume 51, Issue 6 (November 2004).

[EFS56] P. Elias, A. Feinstein and C.E. Shannon (1956), A note on the maximum flow through a network. IRE Transactions on Information Theory, Volume 2, Issue 2 (1956), pp. 117-119.

[ES81] S. Even and Y. Shiloach. An on-line edge-deletion problem. *Journal of the ACM, 28(1):1-4*, 1981.

[FSN96] D. Frigioni, A. Marchetti-Spaccamela, and Umberta Nanni. Fully dynamic output bounded single source shortest path problem. *Proceedings of the seventh annual ACM-SIAM Symposium on Discrete Algorithms*, 1996.

[DR93] D. J. Klein and M. Randić, Resistance Distance, Journal of Mathematical Chemistry 12 (1993), 81-95.

[GR98] Goldberg, A. V. and Rao, S. 1998. Beyond the flow decomposition barrier. J. ACM 45, 5 (Sep. 1998), 783-797.

[GH61] R. E. Gomory and T. C. Hu, Multi-Terminal Network Flows. *Journal of the Society of Industrial and Applied Mathematics*, 9(4):551-570, Dec. 1961.

[GK82] D.H. Greene and D. E. Knuth. *Mathematics for the Analysis of Algorithms*. Birkhauser, 1984.

[Gus90] D. Gusfield, Very Simple Methods For All Pairs Network Flow Analysis (SIAM J. Computing Feb. 1990).

[HK95] M.R. Henzinger and V. King. Fully Dynamic Biconnectivity and Transitive Closure. *Proceedings of the 36th annual IEEE symposium on Foundations of Computer Science*, 1995.

[KKP93] D. Karger, D. Koller and S. Philips, Finding the Hidden Path: Time Bounds for All-Pairs Shortest Paths, (SIAM J. Computing Dec. 1993).

[Kin99] V. King. Fully Dynamic Algorithms for Maintaining All-Pairs Shortest Paths and Transitive Closure in Digraphs. *Proceedings of the 40th annual IEEE symposium on Foundations of Computer Science*, 1999.

[RZ04a] L. Roditty and U. Zwick. On dynamic shortest paths problems. *Proceedings of the 12th Annual European Symposium on Algorithms*, 2004.

[RZ04b] L. Roditty and U. Zwick. Dynamic approximate all-pairs shortest paths in undirected graphs. *Proceedings of the 45th annual IEEE symposium on Foundations of Computer Science*, 2004.

[Tho04] M. Thorup. Fully-dynamic all-pairs shortest paths: faster and allowing negative cycles. *Proceedings of the 9th SWAT, 2004.*