# Simultaneous Source Location

Konstantin Andreev[1] Charles Garrod[2] Bruce Maggs[2]
Adam Meyerson[2]

July 11, 2003

CMU-CS-03-162

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

**Abstract**

We consider the problem of Simultaneous Source Location – selecting locations for sources in a capacitated graph such that a given set of demands can be satisfied. We give an exact algorithm for trees and show how this can be combined with a result of Räcke to give a solution that exceeds edge capacities by at most $O(\log n \log\log n)$, where $n$ is the number of nodes. On graphs of bounded treewidth, we show the problem is still NP-Hard, but we are able to give a PTAS with at most $O(1+\epsilon)$ violation of the capacities, or a $(k+1)$-approximation with exact capacities, where $k$ is the treewidth and $\epsilon$ can be made arbitrarily small.

[1]Mathematics Department, Carnegie Mellon University, konst@cmu.edu
[2]Computer Science Department, Carnegie Mellon University, {charlie,bmm,adam}@cs.cmu.edu

**Keywords:** Approximation algorithms, NP-Hardness, undirected capacitated flow graph, treewidth, source location, facility location

# 1 Introduction

Suppose we are given a capacitated network and we have various demands for service within this network. We would like to select locations for servers in order to satisfy this demand. Such problems arise naturally in a variety of scenarios. One example of this would be the placement of web caches in the Internet, or file servers in an intranet. Another example would be choosing the locations of warehouses in a distribution network.

What does it mean for a server to "serve" a demand? Most previous work has assumed that each server can service each demand for some cost (typically this cost is linear in some underlying distance between server and demand) [20, 16, 3]. In some cases the servers have been considered to be capacitated (each one can provide for only some number of demands) [17]. Still, the primary goal can be considered as minimizing the aggregate distance of demands to servers.

In many natural applications, there is no meaningful notion of distance. Consider serving a web page across the internet. The latency (travel time of a single small packet) under low-congestion conditions tends not to be noticeable to the end-users. The real difficulty here is the underlying capacity of the network. If links become congested, then latency will increase and throughput will suffer. In the case of a distribution network, there may be some relation (typically nonlinear) of costs to distance traveled. But we will definitely have to consider the available throughput! The transportation network has capacities as well as costs, and previous work (assuming costs only) tended to ignore this constraint except at the warehouses themselves.

We consider the problem of Simultaneous Source Location (SSL) – selecting locations for sources in a capacitated network in order to satisfy given demands. Our goal is to minimize the number of sources used. Arata et al. previously gave an exact algorithm for the *Source Location* problem, in the scenario where the sources must be able to satisfy *any single* demand [2]. They also show that the Source Location problem is NP-hard with arbitrary vertex costs. In our problem we must satisfy all demands simultaneously (thus the name Simultaneous Source Location). This is a better model of the natural problems described above. Simultaneous Source Location is easier than the Source Location problem with arbitrary vertex costs explored by Arata et al. – our version of the problem can be reduced to theirs using a high-cost super-sink that is satisfied if and only if all other demands are met. However, we show that our version of the problem is NP-Hard as well and describe various approximations for it.

Our results take several forms. We describe techniques for solving Simultaneous Source Location on a variety of simple graphs. We give an exact algorithm on trees and an approximation for graphs of bounded treewidth. We observe that, in contrast to many other NP-Hard problems (for example vertex cover), Simultaneous Source Location is still NP-Hard even on graphs of treewidth two. Using our algorithm for trees combined with a result of Räcke et al. [18, 6] and later Harrelson et al. [11], we show how to solve source location on a general undirected graph while overflowing the capacity by an $O(\log n \log \log n)$ factor. Combining this with a hardness result for directed graphs allows us to show that no tree decomposition similar to Räcke's decomposition can be found in the directed case. We show Simultaneous Source Location is at least hard on general undirected graphs, but there remains a significant gap between the best approximation algorithm and the known lower bound.

# 2 Problem Statement

An instance of Simultaneous Source Location (SSL) consists of a graph $G = (V, E)$ along with a capacity function $u : E \to R^+$ on the edges and a demand function $d : V \to R^+$ on the vertices. We must select some subset of the vertices $S \subseteq V$ to act as sources. A source set is considered to be feasible if there exists a flow originating from the nodes $S$ that simultaneously supplies demand $d_v$ to each node $v \in V$ without violating the capacity constraints. This is single-commodity flow – we can imagine adding a single "super-source" that is connected to each node of $S$ with an infinite capacity edge, and a single "super-sink" that is connected to each node of $V$ with an edge of capacity $d_v$, and asking whether the maximum flow equals the sum of the demands. Our goal is to select such a source set $S$ of the smallest size.

From the above description, it is clear that the problem is in NP. A source set $S$ can be checked for feasibility by simply solving a flow problem. Unfortunately, finding the set $S$ of minimum size is NP-Hard.

At times we consider various generalizations of this problem. For example, we might have certain nodes in $V$ that are not permitted to act as sources, or have costs associated with making various nodes sources and seek to find a set $S$ of minimum cost. We will also consider simplifications of the problem that place restrictions on the graph $G$ (for example by bounding the treewidth).

# 3 Solving Simultaneous Source Location on Trees

Suppose our graph $G = (V, E)$ is a tree. This special case allows us to solve SSL exactly by dynamic programming. For each vertex $v$ and number of sources $i$, we define $f(v, i)$ to be the amount of flow that must be sent to $v$ by its parent in order to satisfy all demands in the subtree of $v$, assuming this subtree contains $i$ sources. Our algorithm assumes that the tree is rooted and binary; in general we can create an equivalent binary tree by creating virtual nodes and connecting them by edges of infinite capacity, as shown in Figure 2. For convenience, for each node $v$ we define $u(v)$ to be the capacity of the edge from $v$ to its parent.

Our algorithm is described in Figure 1.

*Algorithm BinaryTree$(G, d)$*

1. Initialize $f(v, i) = \infty$ for all $v \in V$ and $0 \leq i \leq |V|$
2. For each leaf vertex $v \in V$:
   (a) Set $f(v, 0) = d_v$.
   (b) Set $f(v, i) = -u(v)$ for all $i \geq 1$.
3. Consider any vertex $v$ with children $v_1, v_2$ for whom $f$ has been computed:
   (a) Loop over all values of $i_1$ and $i_2$ with $0 \leq i_1, i_2 \leq |V|$.
   (b) If $f(v_1, i_1) \leq u(v_1)$ and $f(v_2, i_2) \leq u(v_2)$ then:
      i. Set $f(v, i_1 + i_2) = \min(f(v, i_1 + i_2), max(f(v_1, i_1) + f(v_2, i_2) + d_v, -u(v)))$.
      ii. Also set $f(v, i_1 + i_2 + 1) = -u(v)$.
4. Continue until $f$ is defined at all vertices.
5. Return the minimum $k$ such that $f(r, k) \leq 0$ where $r$ is the root.

Figure 1: Algorithm for SSL on a Binary Tree

Assuming that the above algorithm computes $f(v, i)$ correctly for each vertex, the correctness of the result is immediate. We need to show that $f(v, i)$ correctly represents the minimum amount of flow that must be sent from the parent of $v$ provided the number of sources in the subtree is $i$.

**Theorem 3.1.** *The algorithm described produces an exact solution to the source location problem on a binary tree.*

*Proof.* The proof will be by induction. Our base case is at the leaves. Either a leaf is a source or it is not. If the leaf $v$ is a source, then it requires no flow from its parent, and can send at most $u(v)$ flow upwards along its edge. This yields $f(v, 1) = -u(v)$. On the other hand, if the leaf is not a source it requires flow $d_v$ (its demand) from the parent, so $f(v, 0) = d_v$. Of course, it might not be feasible to send this amount of demand if $d_v > u(v)$.

We now consider any node $v$. Suppose we have correctly computed $f(v_1, i_1)$ and $f(v_2, i_2)$ for all values $i_1, i_2$ for the children of node $v$. Suppose we would like to compute $f(v, i)$. There are $i$ sources to be placed in this subtree. If $v$ is not a source itself, then all the sources are in the child trees. The total demand sent into $v$ will have to be enough to satisfy the demand $d_v$ and additionally to satisfy any residual demand on the children. This means $f(v, i) = \min(\max(d_v + f(v_1, i_1) + f(v_2, i - i_1), -u(v)))$ where the minimum is over choices of $i_1$ and the "max" term ensures that a child node cannot provide more flow to its parent than the capacity of their connecting edge. This is exactly what will be computed in the algorithm, in step 3b($i$). Notice that if satisfying either child tree in this way would require overflowing a capacity $u(v_1)$ or $u(v_2)$ then this allocation of sources to subtrees is not feasible and so should not be considered; this is resolved in step 3b. However, it is also possible that $v$ is itself a source. In this case, provided there is some choice of $i_1 \leq i - 1$ such that $f(v_1, i_1) \leq u(v_1)$ and $f(v_2, i - i_1 - 1) \leq u(v_2)$, we will be able to produce a solution. This solution can send $u(v)$ upwards since $v$ itself is a source. This is dealt with in step 3b(ii). It follows that the algorithm computes exactly the correct values $f(v, i)$ and the algorithm solves SSL. □

**Theorem 3.2.** *We can solve SSL in time $O(n^3)$ on a tree, even if some nodes are disallowed as sources.*

*Proof.* The running time of the algorithm described is $O(n^3)$. For each node we must do $O(n^2)$ work to compute $f(v, i)$ for all $i$. This involves considering all possible values $i_1, i_2$ for the two children.

If our tree is non-binary we can replace any node with more than two children by multiple nodes as shown in Figure 2. This increases the number of nodes by at most a constant factor.

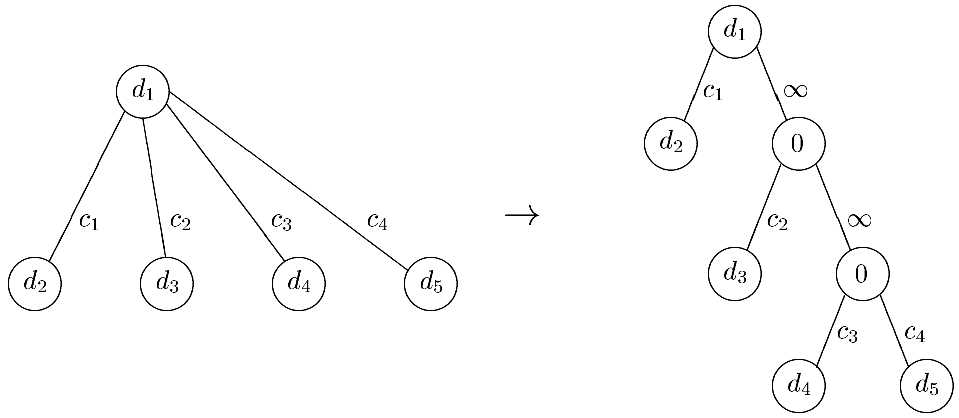


Figure 2: By adding additional nodes linked by infinite capacity, we can convert any non-binary tree into an equivalent binary tree by only doubling the number of nodes in the tree.

We can also modify our algorithm to disallow certain nodes as sources. If a leaf $v$ cannot be a source, then we ignore step 2b and instead set $f(v, i) = d_v$ for all $i$. If some higher node cannot be a source, then we remove step 3b(ii) for that node (this step considers the case where $v$ is a source). The correctness proof is the same as before. □

# 4 Using Räcke's Result

In recent work, Harold Räcke showed that for any undirected graph, it is possible to construct a tree which approximately captures the flow properties of the original graph[18]. Later work [6, 11] improved the result and described a polynomial-time algorithm to construct such a tree. The most recent result is stated more precisely in the following theorem:

**Theorem 4.1.** *Given any capacitated, undirected graph $G = (V, E, u : E \to R^+)$, there exists a capacitated tree $T = (V_T, E_T, u : E_T \to R^+)$ with the following properties:*

1. *The vertices of $V$ are the leaves of $T$.*

2. *For any multicommodity flow $F$ which is feasible on $G$, there exists a flow of equal value between the corresponding leaves on $T$.*

*3. For any flow $F_T$ feasible between the leaves of $T$, there is a feasible $\frac{1}{\rho}F_T$ flow on $G$ for some $\rho = O(\log^2 n \log\log n)$.*

This gives us an approximation for the source location problem. We first construct a tree $T$ as described above. We then solve SSL on the tree, permitting only the leaves to act as sources, using the algorithm of Section 3. We consider using the sources we have obtained on the original graph. We know there exists a flow $F_T$ on the tree from our selected source nodes which satisfies all the demands. It follows that $\frac{1}{\rho}F_T$ is feasible on the graph. We conclude that if we violate the capacities by a factor of $\rho = O(\log^2 n \log\log n)$, then we have a feasible solution to source location. On the other hand, any feasible solution on the graph must also be feasible on the tree. This allows us to produce an exact solution (in terms of the number of sources) while increasing the capacities by $O(\log^2 n \log\log n)$.

In fact, we can improve the result described. If we consider the proof technique used by Harrelson et al. [11], one of the logarithmic factors arises from the non-equality of the minimum cut and maximum flow for multicommodity flow [4]. Since SSL is a single-commodity flow, we can improve the approximation bound to $O(\log n \log\log n)$ on the capacities.

**Theorem 4.2.** *We can produce an optimum (in terms of number of sources) solution to SSL in polynomial time, if we permit $O(\log n \log\log n)$ stretch on the capacities.*

Our results also have interesting implications for directed graphs. Consider the possibility of a Räcke-like representation of a directed graph by a tree, where the capacity on an edge when routing "upwards" might differ from the "downwards" capacity. Suppose such a thing existed, and could be computed in polynomial time, for some factor $\rho$. This would enable us to solve SSL exactly on directed graphs, while exceeding capacities by a factor of $\rho$. But this is NP-Hard, as will be shown in Section 6.2. Thus we have the following:

**Theorem 4.3.** *No Räcke-like decomposition of a directed graph into an $\rho$-approximately flow-conserving tree can be computed in polynomial time, for any value of $\rho$ polynomial in $n$.*

Note that our hardness result is a computational hardness result and assumes a tree-like structure decomposition. Azar et al. [5] have found an existential hardness result which is true for any decomposition, but their bound is weaker: $O(\sqrt{n})$.

# 5 Simultaneous Source Location with Bounded Treewidth

## 5.1 Defining Treewidth

The notion of treewidth was introduced by Robertson and Seymour [19]. Many problems that are in general intractable become polynomial-time solvable when restricted to graphs of bounded treewidth. Furthermore, many graphs arising from natural applications have bounded treewidth. A good survey on the topic is given by Bodlaender [7]. Here is one of the many equivalent definitions of treewidth:

**Definition 5.1.** A graph $G = (V, E)$ has treewidth $k$ if there exists a tree $\tau = (V_\tau, E_\tau)$ along with a mapping $f : V_\tau \rightarrow 2^V$ with the following properties:

1. For all $\alpha \in V_\tau$, $|f(\alpha)| \leq k + 1$.
2. For any $(u, v) \in E$ there exists some $\alpha \in V_\tau$ such that $u, v \in f(\alpha)$.
3. For any $\alpha, \beta, \gamma \in V_\tau$ where $\beta$ lies along the path from $\alpha$ to $\gamma$, if for some $x \in V$ we have $x \in f(\alpha)$ and $x \in f(\gamma)$, then we must also have $x \in f(\beta)$.

These conditions essentially state that each tree vertex represents some subset of at most $k$ graph vertices, each edge in the graph has its endpoints represented together in at least one of the tree vertices, and the set of tree vertices which represent a single graph vertex must form a contiguous subtree.

We observe that it is possible to produce such a tree decomposition for a graph of treewidth $k$ in time linear in the number of nodes and vertices (but exponential in $k$). Assuming $k$ is constant, we are able to produce a tree decomposition – and thereby implicitly detect graphs of constant treewidth – in polynomial time. In general, however, computing the treewidth of an arbitrary graph is NP-Hard.

## 5.2 Nice Decompositions

Bodlaender [8] also introduced the notion of a nice decomposition and proved that any tree decomposition of a graph can be transformed into a nice decomposition still of polynomial size. In a nice decomposition, each node $\alpha \in V_\tau$ has one of the following types:

- A *leaf* node $\alpha$ has no children, and $|f(\alpha)| = 1$
- An *add* node $\alpha$ has one child $\beta$ with $f(\alpha) = f(\beta) \bigcup \{v\}$ for some node $v \in V$
- A *subtract* node $\alpha$ has one child $\beta$ with $f(\alpha) = f(\beta) - \{v\}$ for some node $v \in f(\beta)$
- A *merge* node $\alpha$ has two children $\beta, \gamma$ with $f(\alpha) = f(\beta) = f(\gamma)$

In addition, the nice decomposition has a root node $\rho$ (which is a *subtract* node) with $f(\rho)$ empty.

## 5.3 Approximation for Graphs of Bounded Treewidth

Suppose we are given a graph $G = (V, E)$ with treewidth $k$, for which we would like to approximate the SSL problem. Our algorithm takes in some set of current sources $S$ along with a graph $(V, E)$ and returns a set $S'$ of sources which are feasible for the given graph. Our algorithm for this problem appears in Figure 3.

*Algorithm SL(S, V, E)*

1. Check whether the source set $S$ is feasible for $(V, E)$; if so return $S$
2. If not, find sets $X$ and $B_X$ that have the following properties:
   (a) $|B_X| \leq k + 1$
   (b) For all $(x, y) \in E$ with $x \in X$ and $y \in V - X$, we have $x \in B_X$
   (c) $S \bigcup (V - X)$ is not a feasible source set
   (d) $S \bigcup (V - X) \bigcup B_X$ is a feasible source set
3. Recursively solve SL($S \bigcup B_X, (V - X) \bigcup B_X, E$)

Figure 3: Algorithm for SSL with treewidth $k$

We claim that the set $S^R$ of returned sources has $|S^R| \leq (k + 1)|S^*|$ where $|S^*|$ is the smallest feasible set of sources for $(V, E)$ which includes the given set $S$ as a subset.

**Lemma 5.2.** *Assuming we are always able to find sets $X$ and $B_X$ with the properties described, algorithm SL is a $(k + 1)$-approximation for the source location problem.*

*Proof.* We will prove the lemma by induction on the number of sources required by the optimum solution. If the optimum requires no sources (in other words, $S$ is feasible by itself) then the algorithm will not require any additional sources either (producing a trivial $k+1$-approximation). Otherwise, we suppose the optimum requires at least $t$ more sources (for some $t > 0$). We find a pair of sets $X, B_X$. The optimum must include at least one source from the set $X - S$, since $S \bigcup (V - X)$ is not feasible. Consider the flow structure of the optimum solution. Any flow that passes from a source in $X$ to some node in $V - X$ must travel through $B_X$. If $B_X$ are all sources, we can assume that no flow travels from sources in $X - B_X$ to nodes in $V - X$ (or vice versa), since such flow could equivalently originate from $B_X$. Consider adding all the sources of the optimum solution that are not in $X$ to the set of sources $B_X \bigcup S$. This must be a feasible solution. It follows that the recursive SSL problem on SL($S \bigcup B_X, (V - X) \bigcup B_X, E$) requires at most $t - 1$ additional optimum sources, so by induction we will produce a solution which uses at most $(k + 1)(t - 1) + |B_X| \leq (k + 1)t$ sources in addition to $S$. $\square$

Of course, we still need to prove that we can always find the sets $X, B_X$ with the required properties in polynomial time. In a general graph, such a pair of sets might not even exist, but we will use the assumption of treewidth $k$ to prove existence and the ability to find the sets in polynomial time.

**Lemma 5.3.** *If the current set of sources $S$ is not feasible for the graph $G = (V, E)$ with treewidth $k$, then there exists a pair of sets $X, B_X$ with the required properties; furthermore, such a pair of sets can be found in polynomial time.*

*Proof.* We produce a tree decomposition $(\tau, f)$ of $G$. For each tree node $\alpha$, we define $\tau_\alpha$ to be the subtree of $\tau$ rooted at $\alpha$. We define $f(\tau_\alpha) = \bigcup_{\beta\in\tau_\alpha} f(\beta)$. For each node $\alpha$ we will test whether $S \bigcup (V - f(\tau_\alpha))$ is a feasible set of sources. We find node $\alpha$ such that $S \bigcup (V - f(\tau_\alpha))$ is infeasible, but such that $S \bigcup (V - f(\tau_\beta))$ is feasible for each child $\beta$ of $\alpha$. Note that such an $\alpha$ must exist; we simply travel upwards from the each leaf of the tree until we find one. We now consider returning $X = f(\tau_\alpha)$ and $B_X = f(\alpha)$. We will show that these sets satisfy the required properties.

Since the graph has treewidth $k$ we know $|B_X| = |f(\alpha)| \le k + 1$. Consider any $(x, y) \in E$ with $x \in X$ and $y \in V - X$. From the definition of treewidth, there must exist some node $\beta$ with $x, y \in f(\beta)$. Since $y$ is not in $f(\tau_\alpha)$ we conclude that $\beta$ is not in $\tau_\alpha$. On the other hand, there is some node $\gamma \in \tau_\alpha$ such that $x \in f(\gamma)$ (this follows from $x \in X$). The path from $\gamma$ to $\beta$ must pass through $\alpha$, so the treewidth definition implies $x \in f(\alpha)$ and therefore $x \in B_X$ as desired. The selection of $\alpha$ guarantees that $S \bigcup (V - X)$ is not a feasible source set. This leaves only the fourth condition for us to prove.

We consider the children of $\alpha$. A pair of children $\gamma_1, \gamma_2$ must have $f(\gamma_1) \bigcap f(\gamma_2) \in f(\alpha)$ because of the contiguity property of tree decomposition. Thus the sets of nodes represented by the subtrees of the children can intersect only in the nodes of $B_X = f(\alpha)$. We know that for each child $\gamma$, the set of nodes $S \bigcup (V - f(\tau_\gamma))$ would be feasible. It follows that we can cover all the demand of $f(\tau_\gamma)$ using nodes of $S$ and nodes external to the set. Since the children sets are disjoint except for nodes which we have declared to be sources, the flows to satisfy each child subtree are edge-disjoint; any flow from external nodes must also pass through $B_X$, and we conclude that we can cover $f(\tau_\gamma)$ using $S \bigcup B_X$. It follows that we can cover all of $X$ using the sources $S \bigcup B_X$, making $S \bigcup B_X \bigcup (V - X)$ a feasible source set for the entire graph. □

**Theorem 5.4.** *Algorithm SL produces a $k+1$-approximation to the SSL problem on a graph of treewidth $k$.*

## 5.4 Bounded Treewidth with Capacity Stretch

We will describe an exact algorithm for the SSL problem on graphs of bounded treewidth. The running time of this algorithm will be exponential in the treewidth, and also depends polynomially on the maximum edge capacity. In the general case where the capacities may be large, we will use the technique of Section 8 to obtain a solution with $1 + \epsilon$ stretch on the edge capacities.

Suppose we have found a nice tree decomposition $(\tau, f)$. We will construct a set of vectors of dimension $k + 3$. Each vector has the following form:

$$(\alpha, i, f_1, f_2, ..., f_{k+1})$$

Here $\alpha \in V_\tau$, $0 \le i \le |V|$, and the $f_i$ are feasible flow values (we assume these are from a polynomially-bounded range of integers). Let $S_\alpha = f(\tau_\alpha) - f(\alpha)$ represent the nodes represented by the subtree rooted at $\alpha$ minus the nodes of its boundary (the nodes of $f(\alpha)$ itself). A feasible vector represents the excess flow needed to be sent directly from the nodes of $f(\alpha)$ to $S_\alpha$ to satisfy all the demand in $S_\alpha$ if $S_\alpha$ contains $i$ sources.

We observe that if $k$ is a constant and flow is polynomially-bounded, then the number of possible vectors is polynomial in size. We will determine which such vectors are feasible in polynomial time, then use this to solve SSL. Our algorithm is in Figure 4.

**Theorem 5.5.** *If there are $F$ possible flow values, the BTW algorithm shown in Figure 4 runs in time $O(kn^2NF^{2k+2})$ where $n = |V|$ and $N = |V_\tau|$ and $k$ is the treewidth.*

*Proof.* The total number of possible feasible vectors is upper bounded by $NnF^{k+1}$. Consider what happens when we look at node $\alpha$. If this is a leaf, we do $O(1)$ work. If it is an add node, we will have to build one vector for each of the vectors for $\beta$, of which there are at most $nF^{k+1}$. Each of these constructions takes at most $O(k)$ steps, yielding $knF^{k+1}$ work. If $\alpha$ is a merge node, then we must construct a vector for each pair of vectors for the children; there are at most $n^2F^{2k+2}$ such pairs and constructing the sum takes $O(k)$ time, for $kn^2F^{2k+2}$ work. If $\alpha$ is a subtract node, then for each of the child's vectors we will (at worst) consider every possible allocation of flow. There are at most $F^{k+1}$ such allocations for each of the $nF^{k+1}$ child vectors. Combining these, we see that in the worst case, we will perform $O(kn^2F^{2k+2})$ operations for each node of the tree decomposition, for $O(kn^2NF^{2k+2})$ total time. □

This running time is polynomial assuming that $F$ is polynomial and $k$ is a constant. If the number of possible flow values is not polynomial, we can use the result of Section 8 to effectively reduce the number of flow values. This will cause us to exceed the graph capacities by a factor of $1+\epsilon$.

*Algorithm BTW*

1. Start with the empty set of feasible vectors.
2. Consider each node $\alpha$ from the bottom up:
   - If $\alpha$ is a leaf, then add vector $(\alpha, 0, 0)$.
   - If $\alpha$ is an add node with child $\beta$, then for each feasible vector for $\beta$, copy that vector for $\alpha$, placing flow 0 in the new position corresponding to the additional node in $f(\alpha) - f(\beta)$.
   - If $\alpha$ is a merge node with children $\beta, \gamma$ then for each pair of feasible vectors $x_\beta, x_\gamma$ for the children, create a vector for $\alpha$: $x_\alpha = x_\beta + x_\gamma$ (adding the number of sources and the flows while changing the choice of node from $V_\tau$ to $\alpha$).
   - If $\alpha$ is a subtract node with child $\beta$, then consider each feasible vector $x_\beta$ for the child. Let the subtracted node be $b \in f(\beta) - f(\alpha)$. This node requires some flow $r_b$ which is the sum of the demand $d_b$ and the flow value for $b$ in $x_\beta$. We consider all feasible allocations of flow $F(a)$ to nodes $a \in f(\alpha)$ such that $|F(a)| \leq u(b,a)$ and $\sum_{a \in f(\alpha)} F(a) \geq r_b$. For each such allocation we construct a vector $x_\alpha$ whose number of sources is equal to $x_\beta$ and with flow value at $a$ equal to the flow value in $x_\beta$ plus $F(a)$. This corresponds to refusing to make $b$ a source. We now consider making $b$ a source. This corresponds to creating a vector $x_\alpha$ with one more source than the vector $x_\beta$. We set the flow value for a node $a \in f(\alpha)$ to be the flow value in $x_\beta$ minus $u(b,a)$.
3. Now consider all vectors for the root node $\rho$. These are simply pairs $(\rho, i)$ since $f(\rho)$ is empty. We return the minimum value of $i$ such that $(\rho, i)$ is in the feasible set.

Figure 4: Exact Algorithm for SSL with Treewidth $k$

## 5.5 Lower Bound for Treewidth 2 Graphs

We show that the SSL problem is NP-Hard even on graphs with treewidth two.

**Theorem 5.6.** *SSL is NP-Hard even on graphs with treewidth two.*

*Proof.* The proof is by reduction from subset sum. We are given a set of numbers $\{x_1, x_2, \ldots, x_n\}$, and would like to determine whether some subset of the given inputs sums to $A$. Suppose the sum of all the numbers is $S$. We construct an SSL instance with $2n+2$ nodes. Each number will be represented by a pair of nodes of demand $S$ with an edge of capacity $S$ between them. Our example is a four level graph. On the first level we have a node of demand $A$ which connects to one side of every pair of nodes that represent a number. The capacity on the edge between the $A$ node and the number $x_i$ node is exactly $x_i$. The number nodes from the second level are paired with the nodes from level 3. All nodes from level 3 are connected to a single node at level 4 with an edge of capacity corresponding to their number. The node at level 4 has demand $S - A$. This graph is shown in Figure 5. If there exists a subset of the numbers summing to $A$, then we can place sources on the lefthand node for each of the numbers in that subset and the righthand node for all the other numbers; it is straightforward to see that this is a feasible SSL making use of $n$ sources. On the other hand, consider a SSL solution. We must select one of the two nodes for each of the numbers (otherwise there is not enough capacity to satisfy them). It follows that the SSL uses at least $n$ sources. If exactly $n$ sources are used, then the result corresponds to a subset sum solution. It follows that solving source location exactly on this graph will solve subset sum. The graph given has treewidth two; we can see this because if we remove the node of demand $A$, the remaining graph is a tree. We take the (treewidth one) tree decomposition and add the node of demand $A$ to the subset $f(\alpha)$ for all $\alpha$. This is a tree decomposition of width two. □

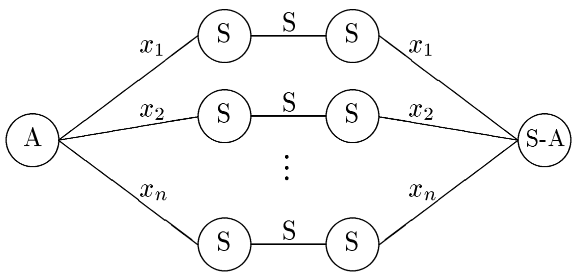


Figure 5: Simultaneous Source Location is NP-Hard even on graphs of treewidth two. To satisfy each demand in this graph using only $n$ sources we must find a partition of $\{x_1, x_2, \ldots, x_n\}$ whose sums are $A$ and $S - A$.

# 6 Simultaneous Source Location on Directed Graphs

## 6.1 Greedy $O(\log n)$ Approximation

We are given a directed graph $G = (V, E)$ for which we would like to approximate the SSL problem. We propose a simple greedy algorithm. We start with no sources and no demand satisfied. We add the source which maximizes the increase in the total satisfied demand. We repeat this until all demand is satisfied.

**Theorem 6.1.** *The greedy algorithm gives an $O(\log n)$ approximation on the number of sources with no violation of the capacities.*

*Proof.* Suppose that the optimum solution uses $t$ sources. At some point in time, suppose our current set of sources can cover demand $d$ and the total demand in the graph is $D$. Consider the residual graph after adding the flows from our sources to satisfy demand $d$. The residual demand is $D - d$, and if we were to add all the optimum sources we would be able to satisfy the full residual demand (note that this is essentially single commodity flow since sources are equivalent). It follows that we can add one source to the residual graph to satisfy demand $\frac{D-d}{t}$. We apply the standard greedy analysis for problems like $SETCOVER$ to show that the full demand will be covered in $O(\log D)$ steps. Assuming that the maximum and minimum demand are within a polynomial factor of one another, we are done. Otherwise, we can apply scaling arguments of Section 8 to give the desired $O(\log n)$ factor. □

## 6.2 Lower Bound for Directed Graphs

We show that $O(\log n)$ is the best approximation factor we can expect to obtain for the directed SSL problem in polynomial time, due to a reduction from set cover.

**Theorem 6.2.** *We cannot approximate directed SSL to better than $O(\log n)$ in the worst case, unless $NP \subset DTIME(n^{O(\log \log n)})$.*

*Proof.* Suppose we would like to solve a set cover instance. We construct a graph with one node for each set, one node for each element, and one additional node. The additional node is connected by a directed edge of capacity one to each of the sets. Each set is connected by a directed edge of capacity $N$, where $N$ is more than the sum of the number of sets and elements, to each of its elements. The additional node has demand one, each set has demand one, each element has demand $N$. We solve the SSL problem to some approximation factor $\rho$ on this graph. We first observe that no element should be selected as a source; otherwise we could simply select one of the sets containing that element as a source instead. Second, we observe that the additional node will be selected as a source. We have a solution consisting of $\rho t$ nodes, where $t$ is the optimum. Consider the set nodes that we selected as sources. Every element node must be connected to one of these set nodes in order for its demand to be satisfied (note $N$ is greater than the number of sets). It follows that we have a set cover of size $\rho t - 1$. Similarly, observe that any set cover, plus the additional node, forms a feasible SSL solution. So we have obtained a $\frac{\rho t-1}{t-1} \geq \rho$ approximation to set cover. This is unlikely for $\rho$ smaller than $\log n$ due to the results of Feige [10]. □

We observe that even if we are allowed to violate edge capacities by some large factor (say less than $N$), the reduction from set cover still holds.

# 7 Lower Bound for Undirected Graphs

We show that SSL does not have a polynomial-time approximation scheme via an approximation-preserving reduction from vertex cover.

To see this first consider a vertex $v$ of degree two whose demand is some value $x$, and the capacity of each edge adjacent to $v$ is also $x$. We claim that if $v$ is not a source that $v$ cannot produce any net outflow in a valid solution. Since $v$ consumes $x$ demand itself and only $x$ flow can be routed to $v$ along either of its adjacent edges, any additional outflow would have to be routed to $v$ via the same edge by which it leaves. The outflow from $v$ can therefore be combined with the inflow within the same edge, creating an equivalent solution with no outflow. We use a large number of these objects in our vertex cover reduction.

Now consider a vertex cover problem on a graph $G = (V, E)$. We create a SSL problem on a flowgraph $G' = (V', E', c, d)$, prove that a size $k$ vertex cover implies a set of $k$ sources that satisfies $G'$, and also prove that any solution requiring $k \leq n$ sources has a corresponding vertex cover of size $k$ in $G$, where $n = |V|$ is the number of vertices in $G$. For each vertex $v \in V$, create a "base" vertex $v' \in V'$ with 0 demand. For each edge $e = (v_i, v_j) \in E$ create $3n + 1$ "stack" vertices in $V'$, each with demand 1 and having capacity 1 edges to the associated base vertices $v'_i$ and $v'_j$ as shown in Figure 6. Notice that just as selecting a vertex covers all of its adjacent edges in the vertex cover problem, making the base vertex a source in our flow graph satisfies all the demand of its adjacent edge stacks. Thus, a vertex cover of size $k$ in $G$ implies a set of $k$ sources in $G'$ that satisfies all the demand in the graph. Now suppose we have a set of $k \leq n$ sources in $G'$ that satisfies all the demand in the graph. Suppose for sake of contradiction that there is an edge stack $s$ in $G'$ such that neither of its associated base vertices is a source. This stack comprises a total demand of size $3n+1$ that must somehow be satisfied. Consider now the set of all vertices in $s$ and all vertices adjacent to $s$'s base vertices, noticing that each of these is a stack vertex, which is an object of the form discussed in the previous paragraph. Each of these vertices can create no net outflow unless it is itself chosen as a source. However, at least $3n + 1$ flow must be generated from these sources to satisfy the vertices in $s$. But each of these vertices can satisfy at most a demand of three even if chosen as a source – one unit satisfying its own demand, and two units of total outflow out of its edges. At least $\frac{3n+1}{3}$ of these vertices therefore must be sources, contradicting the claim that our solution is of size at most $n$. We then have that each edge stack is satisfied by at least one of its associated base vertices, and since at most $k$ base vertices are sources in the flow graph we have a vertex cover of size at most $k$ in $G$. Hastad [12] showed that vertex cover is inapproximable within $\frac{7}{6}$.

**Theorem 7.1.** *Simultaneous Source Location is $1.36067 - \epsilon$ hard on general undirected graphs if edge capacities are not violated.*

The proof follows from the reduction above and the recent hardness results by Dinur and Safra [9].

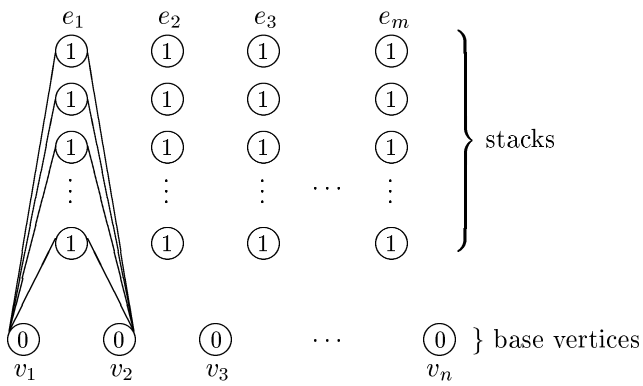


Figure 6: Simultaneous Source Location is as hard as vertex cover on undirected graphs. In this example, edge $e_1$ connects vertices $v_1$ and $v_2$ in the original graph.

# 8 Dealing with Super-Polynomial Capacities

Some of our analyses assume that there are only a polynomial number of capacities. Here we present a method to extend those analyses to when the capacities are super-polynomial, allowing us to satisfy demands while exceeding capacities by a $1+\epsilon$ factor in those cases.

When capacities are super-polynomial, we express the flow and capacity of each edge as $\alpha F^i$ for some value of $i$ (which might be different from edge to edge) and some value of $\alpha$ which is between $F$ and $F^2$. This can be done by rounding every flow up to the nearest value. Once this is done, we might no longer have a feasible flow. The new capacities (which might be larger than the old capacities by a $1+\frac{1}{F}$ factor) will not be exceeded. However, flow conservation may no longer hold. Consider any node. If the original inflow was $f$ then the outflow was $f$ also. But it is possible that after this rounding upwards, the inflow is still $f$ and the outflow has increased to $f(1+\frac{1}{F})$. We consider such a flow to be feasible. This means that a node might effectively "generate" some amount of flow, but this amount is at most $\frac{1}{F}$ of the inflow.

Now consider any graph with a feasible flow under the representation described above. We would like to transform this into a real flow. If we consider splitting the "generated" flow equally among outgoing edges, we will see that as flow passes through a node the percentage of "real" flow might decrease by a factor of $\frac{F}{F+1}$. In the worst case, some edge might have a fraction of "real" flow equal to $(\frac{F}{F+1})^n \geq 1-\frac{n}{F}$.

We let $F=\frac{n}{\epsilon}$. It follows that we can satisfy at least $1-\epsilon$ of each demand while exceeding capacities by a factor of $1+\frac{\epsilon}{n}$. Since we can scale the flows and capacities, this means we can satisfy the demands while exceeding capacities by $1+\epsilon$ factor.

This is useful in bounding the running time and approximation factor of various techniques for approximating SSL.

# 9 Conclusions

We define the Simultaneous Source Location problem and solve the problem exactly on trees. We present a $(1+\epsilon)$ violation of the capacities PTAS for graphs of bounded treewidth. On general graphs we find a solution with exact number of sources which can exceed the capacities by at most a factor of $O(\log n \log\log n)$. We show a $O(\log n)$ factor approximation on the number of sources with no violation of the capacities for general directed graphs. We believe that many interesting applications of this problem involve graphs of low treewidth; many of the connectivity graphs of real networks have been observed to have low tree width [8].

The main open problem is the approximability of SSL on undirected graphs of large treewidth. No constant approximation on the number of sources is known, even if we allow constant violation of the capacities. The only lower bound on approximability with exact capacities is $1.36067-\epsilon$. An approximation factor asymptotically better than 2 would be a surprising improvement on existing vertex cover results [15]. One can also consider adding costs on the edges and/or the vertices.

## References


[1] W. A. Aiello, F. T. Leighton, B. M. Maggs, M. Newman Fast algorithms for bit-serial routing on a hypercube *Mathematical Systems Theory*, 1991

[2] K. Arata, S. Iwata, K. Makino, and S. Fujishige. Locating sources to meet flow demands in undirected networks. *Journal of Algorithms, 42*, 2002.

[3] V. Arya, N. Garg, R. Khandekar, V. Pandit, A. Meyerson, and K. Munagala. Local search heuristics for $k$-median and facility location problems. *Proceedings of the 33rd ACM Symposium on Theory of Computing*, 2001.

[4] Y. Aumann, Y. Rabani An $O(\log k)$ approximate mincut max-flow theorem and approximation algorithms *SIAM Journal of Computing, 27(1)*, 1998

[5] Y. Azar, E. Cohen, A. Fiat, H. Kaplan, and H. Räcke. Optimal oblivious routing in polynomial time. *Proc. 35th Annual ACM Symposium on Theory of Computing*, 2003.

[6] M. Bienkowski, M. Korzeniowski, and H. Räcke. A practical algorithm for constructing oblivious routing schemes. *Fifteenth ACM Symposium on Parallelism in Algorithms and Architectures*, 2003.

[7] H. L. Bodlaender. Treewidth: Algorithmic Techniques and Results. *Proceedings 22nd International Symposium on Mathematical Foundations of Computer Science*, 1997

[8] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Compututer Science, 209(1)*, 1998.

[9] I. Dinur and S. Safra. On the importance of being biased. *Proceedings of the 34rd ACM Symposium on Theory of Computing*, 2002.

[10] U. Feige. A threshold of $\ln n$ for approximating set cover. *Journal of the ACM*, 1995.

[11] C. Harrelson, K. Hildrum, and S. Rao. A polynomial-time tree decomposition to minimize congestion. *Fifteenth ACM Symposium on Parallelism in Algorithms and Architectures*, 2003.

[12] J. Hastad, Some optimal inapproximability results. *Proceedings 29th Ann. ACM Symp. on Theory of Computing, ACM, 1-10*, 1997

[13] P. Klein, S. A. Plotkin, S. Rao Excluded minors, network decomposition, and multicommodity flow. *Proceedings of the 25th ACM Symp. on Theory of Computing*, 1993

[14] T. Leighton, S. Rao An approximate max-flow min-cut theorem for uniform multicommodity flow problems with applications to approximation algorithms *Proceedings of the 29th Symp. on Foundations of Computer Science*, 1988

[15] S. Khot and O. Regev. Vertex cover might be hard to approximate within $2 - \epsilon$. *Proceedings of the 17th IEEE Conference on Computational Complexity*, 2002.

[16] M. Mahdian, Y. Ye, and J. Zhang. Improved approximation algorithms for metric facility location problems. *APPROX*, 2002.

[17] M. Pál, É Tardos, and T. Wexler. Facility location with nonuniform hard capacities. *Proceedings of the 42nd IEEE Symposium on the Foundations of Computer Science*, 2001.

[18] H. Räcke. Minimizing congestion in general networks. *IEEE Symposium on Foundations of Computer Science*, 2002.

[19] N. Robertson, P. D. Seymour Graph Minors II. Algorithmic aspects of tree width *Journal of Algorithms 7*, 1986.

[20] D. Shmoys, É Tardos, and K. Aardal. Approximation algorithms for facility location problems. *Proceedings of the 29th ACM Symposium on Theory of Computing*, 1997.