Approximation Algorithms for Connected Dominating Sets*

Sudipto Guha†
Dept. of Computer Science
University of Maryland,College Park,
MD 20742

Samir Khuller‡
Dept. of Computer Science and UMIACS
University of Maryland,College Park,
MD 20742

Abstract

The dominating set problem in graphs asks for a minimum size subset of vertices with the following property: each vertex is required to either be in the dominating set, or adjacent to some node in the dominating set. We focus on the question of finding a *connected dominating set* of minimum size, where the graph induced by vertices in the dominating set is required to be *connected* as well. This problem arises in network testing, as well as in wireless communication.

Two polynomial time algorithms that achieve approximation factors of $O(H(\Delta))$ are presented, where Δ is the maximum degree, and H is the harmonic function. This question also arises in relation to the traveling tourist problem, where one is looking for the shortest tour such that each vertex is either visited, or has at least one of its neighbors visited. We study a generalization of the problem when the vertices have weights, and give an algorithm which achieves a performance ratio of $3 \ln n$. We also consider the more general problem of finding a connected dominating set of a specified subset of vertices and provide an $O(H(\Delta))$ approximation factor. To prove the bound we also develop an optimal approximation algorithm for the unit node weighted Steiner tree problem.

1 Introduction

The *connected dominating set problem* is defined as follows. Find a minimum size subset S of vertices, such that the subgraph induced by S is connected and S forms a dominating set. This problem is known to be NP-hard [7]. Recall that a dominating set is one in which each vertex is either in the dominating set, or adjacent to some vertex in the dominating set.

*A preliminary version of this paper will appear in the Proceedings of the Fourth Annual European Symposium on Algorithms (ESA 1996).
†Research supported by NSF Research Initiation Award CCR-9307462. Email addr: `sudipto@cs.umd.edu`
‡Research supported by NSF Research Initiation Award CCR-9307462, and NSF CAREER Award CCR-9501355. Email addr: `samir@cs.umd.edu`

A related problem is the *traveling tourist problem.* Given a graph $G = (V, E)$ find the shortest walk visiting a subset of vertices, such that each vertex is either visited, or has at least one of its neighbors visited. (The vertices of the graph correspond to monuments the tourist would like to see, and an edge between two vertices denotes visibility of one monument from another.) The shortest such walk would guarantee that the tourist sees all monuments of interest.

We show that a β approximation for the connected dominating set problem yields a 2β approximation for the traveling tourist problem. Consider a spanning tree of the connected dominating set S and perform a tree traversal. This yields a walk in which exactly $2(|S| - 1)$ edges are traversed. Any set of vertices visited by the tourist, form a connected dominating set. Thus $S \leq \beta \cdot OPT \leq \beta \cdot OPT_{TT}$, where OPT_{TT} denotes an optimal traveling tourist tour, and the result follows.

We also study the connected dominating set problem when the vertices have weights, and we wish to minimize the total weighted sum of the vertices that form the connected dominating set. This also yields an approximation algorithm for the weighted traveling tourist problem, where the weights could potentially denote the tourist's cost of buying a ticket to visit the monument.

We also consider Steiner generalizations, where only a specified subset of vertices have to be dominated by a connected dominating set.

1.1 Our Results

We present two approximation algorithms for this problem. The first algorithm develops a greedy algorithm for solving the problem. A naive greedy algorithm is shown to do badly. Surprisingly, with a simple modification we are able to show an approximation factor of $2(1 + H(\Delta))$ (in practice, this algorithm appears to do very well). We also provide a very efficient implementation of this algorithm.

The second algorithm is an improvement of the first algorithm. The algorithm finds a dominating set in the first phase, and in the second phase connects the dominating set. In an earlier version of this paper [8] we established a bound of $H(\Delta) + H(H(\Delta))$. Using Slavík's greedy set-cover bound [17], we were able to show that the approximation factor is $\ln n + O(1)$. Recently, Berman suggested a modification to the algorithm, which improves the approximation factor to $H(\Delta) + 2$. We describe this algorithm and give a simple proof for a performance guarantee of $\ln \Delta + 3$.

We also show an approximation preserving reduction from the set-cover problem to the connected dominating set problem, showing that it is hard to improve the approximation guarantee unless $NP \subseteq DTIME[n^{O(\log \log n)}]$ [13, 6]. We give a $3 \ln n$ approximation for the version when the vertices have weights. We also show that the upper bound of $2 \ln k$ for approximating node weighted Steiner trees [10], can be improved to $\ln k$, when all Steiner vertices have unit weight. We then use this result to give a $3 \ln k$ approximation for finding a connected dominating set for a specified subset of vertices. We also outline a second algorithm that gives an approximation factor of $(1 + c)H(\min(\Delta, k)) + O(1)$, where c is the best approximation ratio for the Steiner tree problem (currently $c = 1.644$ [12]). Even though this algorithm has a better approximation guarantee, it is not practical due to the high running time, albeit polynomial.

1.2 Preliminaries

The Steiner tree problem is defined as follows: given a subset of required vertices in an edge weighted graph, find a minimum weight tree spanning the required subset of vertices. (Note that the tree may include other vertices that are not required vertices.) The node weighted Steiner tree problem is essentially the same problem, except that the vertices of the graph have weights associated with them and the weight of the tree is the sum of the weights of its vertices. The Unit Node Weighted Steiner tree is the special case when all vertices that are not required, have the same weight. The required vertices all have zero weight.

The set cover problem is the following: given a set of elements U, and a set of subsets $\mathcal{S}$, of U, we wish to find the smallest collection of sets $\mathcal{S}' \subset \mathcal{S}$ such that $\cup_{S \in \mathcal{S}'} S = U$.

The set TSP problem is defined as follows: given an edge weighted graph $G = (V, E)$ and a partition of $V = (V_1 \cup V_2 \cup \ldots \cup V_k)$, find the shortest tour that contains at least one vertex from each V_i.

Given a graph $G = (V, E)$, we use Δ to denote the maximum degree of a vertex in the graph. We use n and m to denote the number of vertices and edges in G. We use $N(v)$ to denote the set of neighbors of a vertex v.

1.3 Applications

The paper by Paul and Miller [15] discusses applications related to testing nodes in a computer network using a short "traveling tourist tour". They also consider the related question of finding a tour that visits each edge of the graph (connected vertex cover). This is needed when one requires testing the links as well as the nodes. Approximation algorithms for the latter problem were given by Arkin, Halldórsson and Hassin [1]. We observe that there is a simple algorithm for the unweighted connected vertex cover problem that gives a factor 2 approximation (the one given in [1] is more complicated). Do a Depth First Search, and take all the non-leaf vertices as the nodes in the vertex cover. This clearly induces a connected graph, and the approximation ratio is 2, as shown by Savage [16]. In practice, however this method will probably give large connected vertex covers.

Other applications for the connected dominating set problem are in doing broadcasts for wireless computers in digital battlefields. The broadcast is done to the vertices in the connected dominating set. The nodes in the connected dominating set are responsible for relaying messages. Each node not in the dominating set, is not responsible for relaying any messages [9]. Other relevant issues are regarding the maintenance of the connected dominating set as the network topology changes.

2 Algorithm I

We introduce an algorithm that finds a connected dominating set, by "growing" a tree.

The idea behind the algorithm is the following: grow a tree T, starting from the vertex of maximum degree. At each step we will pick a vertex v in T and "scan it". Scanning a vertex, adds edges to T from v to *all* its neighbors not in T. In the end we will find a spanning tree T, and will pick the non-leaf nodes as the connected dominating set.

Initially all vertices are unmarked (white). When we scan a vertex (color it black), we mark all its neighbors that are not in T and add them to T (color them gray). Thus marked nodes that have not been scanned are leaves in T (gray nodes). The algorithm continues scanning marked nodes, until all the vertices are marked (gray or black). The set of scanned nodes (black nodes) will form the CDS in the end.

The main question is the following: what rule should we use for picking a vertex to be scanned? A natural choice is to pick the vertex that has the maximum number of unmarked (white) neighbors. We call this the "yield" of the scan step. Unfortunately, as the following example shows this may not work well (see Fig. 1).

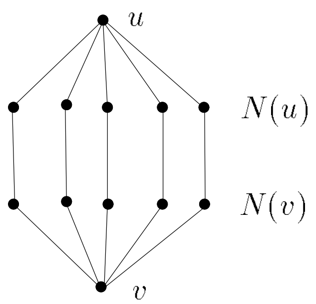

Figure 1: Example to show that the scanning rule fails.

Let u and v be vertices of degree d. There is a solution of size four, by picking a path from u to v as the CDS. The algorithm begins by marking and scanning u. This adds all of u's neighbors to T. We pick a vertex from $N(u)$ and scan it, adding its only unmarked neighbor (from $N(v)$) to T. At this point, each vertex has exactly one unmarked neighbor. We could pick a vertex from $N(u)$ again, and scan it, adding its only unmarked neighbor to T. This continues until all the vertices from $N(u)$ have been scanned. Finally we scan a vertex from $N(v)$ and mark v. At this point, the algorithm has picked $d+2$ vertices.

Implementation Issues: The above algorithm can be implemented in $O(m)$ time (and was implemented). To achieve this running time, we use a data structure DS that maintains all gray vertices in T with a key value equal to the number of white neighbors that they have. Rather than using a heap, we maintain an array of linked lists, where $DS[i]$ is a pointer to a (doubly linked) list containing all the gray vertices that have exactly i white neighbors. We also maintain an integer $maxd$ that records the maximum i, such that $DS[i] \neq \texttt{nil}$. This makes it easy to locate a gray vertex with the highest "yield". The main work is in updating the value of $maxd$ when $DS[maxd]$ becomes `nil`. The work that is done is at most $O(maxd)$, and since at this step, $maxd$ vertices are colored gray, we can "charge" the work done to the vertices that are colored gray at this step. (Equivalently, we could develop a potential function to prove this.) The other operations are easy to perform (for example, when a vertex is colored gray, we need to update the entries for its neighbors that are already in DS and create an entry in DS for this vertex). The entire algorithm runs in $O(m)$ steps. This implementation is useful because it leads to a heuristic for the *maximum leaf spanning tree* problem as well [11].

Modified Greedy Algorithm: We now modify the scanning rule to prove a good approximation ratio for this class of algorithms (that grow a connected dominating set). We define a new operation of scanning a pair of adjacent vertices u and v. Let u be gray and v be white. Scanning the pair means, first making u black (this makes v along with some other nodes, gray) and then coloring v black (makes more nodes gray). The total number of nodes that are colored gray is called the "yield" of the scan step. *At each step, we will either scan a single vertex, or a pair of vertices, whichever gives the higher yield.* (In some sense we are doing a "look-ahead" by one extra vertex, and are willing to scan a pair, if this has a higher yield.)

It is clear that this algorithm finds the optimal solution in the example shown in Fig. 1. What is perhaps a little surprising, is that this simple modification lets us prove the following theorem.

Theorem 2.1 *Using the scanning rule described above yields a connected dominating set of size at most* $2(1 + H(\Delta)) \cdot |OPT_{DS}|$.

Proof: Let OPT_{DS} be the set of vertices in an optimal dominating set. The sets of vertices of G dominated by vertex $i \in OPT_{DS}$ is called S_i (we assume that i also belongs to S_i. If a vertex is dominated by more than one vertex, we arbitrarily put it in one of the sets). The proof will be based on a charging scheme. Each time we scan a vertex, we add a new vertex to our connected dominating set. We will "charge" each new vertex marked (colored gray) in this step. Since each vertex in the graph gets marked exactly once, it is charged exactly once (the first time it is marked). We will then prove that the total charge on the vertices belonging to a set S_i (for any i) is at most $2(1 + H(\Delta))$. Since there are $|OPT_{DS}|$ sets in the optimal solution, the theorem follows.

Assume that when we pick a vertex to scan, we mark x new vertices. We will charge each such newly marked vertex $\frac{1}{x}$. In some steps we scan two vertices, and charge each newly marked vertex $\frac{2}{x}$. The main advantage of the "look-ahead" is the following. *The instant we mark some nodes in set S_i, even if vertex i has not been marked, since it is adjacent to a marked vertex, it becomes eligible to be scanned as part of a pair.* Without the look-ahead, only marked vertices were candidates to be scanned .

We now prove the upper bound on the total charges to vertices belonging to a single set S_i. At each step, some vertices may get marked. The number of unmarked vertices is initially u_0, and finally drops to 0. Let u_j denote the number of unmarked vertices after step j. For simplicity, let us assume that at each step some vertices of S_i are marked, so the number of unmarked vertices decreases at each step.

The number of marked vertices after the first step is $u_0 - u_1$. Each vertex gets a charge of at most $\frac{2}{u_0 - u_1}$ (the actual charge may be a lot smaller, if only one vertex was scanned at this step, or if we marked many other vertices as well). Once some vertex in S_i is marked, vertex i becomes an "eligible" vertex to be scanned as a part of a pair, since it is *adjacent* to a marked vertex. In the j^{th} step, the number of vertices of set S_i that get marked is $u_j - u_{j+1}$, and the charge to each vertex is at most $\frac{2}{u_j}$ as vertex i was an eligible vertex to be scanned. Let $u_k = 0$.

Adding up all the charges we get

$$\frac{2}{u_0 - u_1}(u_0 - u_1) + \sum_{j=1}^{k-1} \frac{2}{u_j}(u_j - u_{j+1})$$

$$\leq 2 + 2\sum_{j=1}^{k-1} \frac{(u_j - u_{j+1})}{u_j} .$$

(With some algebraic manipulation (see [5, page 977]), one can show that this is at most $2(1 + H(\Delta))$. □

Remark: We could modify the algorithm and at each step scan either one or two vertices, whichever results in a smaller charge to each vertex. In practice, this should give better solutions.

Implementation Issues: A naive implementation appears to give a worst case running time of $O(mn^2)$. In each iteration we choose either one vertex, or a pair of vertices, and color them black. It is clear that we may have $\Theta(n)$ iterations, since the optimal solution may have $\Theta(n)$ vertices. In each iteration, we wish to identify a pair of nodes with the highest yield. For each gray vertex u, we scan its adjacency list and consider all its white neighbors. For *each* white neighbor v of u, we wish to determine the number of vertices that would get marked if we scanned the pair (u, v). Since u and v have common white neighbors, we cannot simply add up the number of white neighbors of each vertex to obtain the "yield" of this pair. We need to identify the number of white neighbors of v that are *not* adjacent to u (since those will not be colored gray by u). The number of steps in a single iteration can be computed as follows.

Let G be the gray nodes in T. Let W be the white vertices that are adjacent to gray vertices. We can upper bound the total work done in a single iteration as follows:

$$S = \sum_{u \in G} \sum_{v \in N(u) \wedge v \in W} d(v) .$$

In the double summation each vertex in W is counted as many times as the number of its gray neighbors, we obtain the following.

$$S \leq \sum_{v \in W} d(v)^2 \leq \sum_{v \in W} n \cdot d(v) \leq O(mn) .$$

This yields a bound of $O(mn^2)$. We now show that the total number of steps over all iterations is $O(mn)$ by a more careful analysis.

For each vertex we can maintain two adjacency lists, one of its gray neighbors and one of its white neighbors. We use $d_W(u)$ to denote the number of white neighbors of u and $d_G(u)$ to denote the number of gray neighbors of u. The work done in a single iteration is as follows:

$$S = \sum_{u \in G} \sum_{v \in W \wedge v \in N(u)} d_W(v)$$

$$= \sum_{v \in W} d_W(v) \cdot d_G(v) \ .$$

(In the double summation, each vertex v is counted as many times as the number of its gray neighbors.) Observe that at this step, we will make a subset of white vertices gray.

Lemma 2.2 *The number of white vertices that are made gray in this iteration is at least*

$$\frac{1}{2} \max_{v \in W} d_W(v) \ .$$

Proof: We pick the pair of vertices that give the highest "yield"; we certainly consider all such vertices v, and color their white neighbors gray. We might pick a single vertex with a smaller yield, but only if its yield is at least half the yield of a pair of vertices. □

At this step, we can "charge" the vertices whose color changed from white to gray. The charge to each such vertex is at most

$$\frac{\sum_{v \in W} d_W(v) \cdot d_G(v)}{\frac{1}{2} \max_{v \in W} d_W(v)}$$

$$\leq 2 \sum_{v \in W} d_G(v) = 4m \ .$$

Since each vertex changes color from white to gray exactly once over the entire algorithm, and there are n such vertices the total number of steps is $O(mn)$.

The only remaining issue is maintaining the required adjacency lists. This can be done each time we change the color of a vertex from white to gray by scanning its adjacency list, and updating the structures for its neighbors.

3 Algorithm II

An alternate approach to growing one connected tree is to grow separate components that form a dominating set and to then connect them together. One straightforward approach is to find a dominating set using a greedy heuristic, and to use a Steiner tree approximation to connect it. Since members of the optimum connected dominating set along with the members of the dominating set we found, form a spanning tree, we can prove a performance guarantee of $c(1 + H(\Delta))$, where c is the best approximation ratio for the unweighted Steiner tree problem (currently $c = 1.644$ [12]).

For the special case when the required vertices form a dominating set in a graph and all edges have unit weight, Berman and Fürer [3] have announced a new algorithm with $c = \frac{4}{3}$. Thus we can improve the performance ratio to $\frac{4}{3}(1 + H(\Delta))$. By applying a simple greedy strategy to connect the vertices in the dominating set, we proved a bound of $H(\Delta) + H(H(\Delta))$ [8]. Here we present a modification of the above algorithm, as suggested by Berman [2], and are able to prove a performance guarantee of $\ln \Delta + 3$. (Berman has an alternate proof for a performance ratio of $H(\Delta) + 2$.)

The algorithm runs in two phases. At the start of the first phase all nodes are colored white. Each time we include a vertex in the dominating set, we color it black. Nodes that are dominated are colored gray (once they are adjacent to a black node). In the first phase the algorithm picks a node at each step and colors it black, coloring all adjacent white nodes gray. A *piece* is defined as a white node or a black connected component. At each step we pick a node to color black that gives the maximum (non-zero) reduction in the number of pieces.

We show that at the end of this phase if no vertex gives a non-zero reduction to the number of pieces, then there are no white nodes left.

In the second phase, we have a collection of black connected components that we need to connect. Recursively connect pairs of black components by choosing a chain of two vertices, until there is one black connected component. Our final solution is the set of black vertices that form the connected component.

Lemma 3.1 *At the end of the first phase there are no white vertices left.*

Proof: Suppose there is a white node v at the end of the phase. We will show that there is a vertex that strictly reduces the number of pieces. If v has a white neighbor then coloring v black, reduces the number of white nodes by two, and increases the number of black components by one, thus picking v would reduce the number of pieces. Otherwise, v has a gray neighbor u. Coloring u black would reduce the number of white nodes, and not increase the number of black components since u is adjacent to a black node. Thus picking u reduces the number of pieces. □

We show that at the end of the first phase there is always a pair of black components that can be connected by choosing a chain of two vertices. For each such component i, we consider the shortest path to component j. The path goes through vertices $u_1, u_2, u_3, \ldots, u_k$ not in components i or j. u_1 is dominated by a vertex in component i. Observe that u_2 is gray, otherwise picking u_1 would give a strict reduction in the number of pieces. Thus u_2 is adjacent to a black component ℓ ($\ell \neq i$ since we selected the shortest path from i to j). Components i and ℓ can be connected by choosing a chain of two vertices.

Theorem 3.2 *The connected dominating set found by the algorithm is of size at most* $(\ln \Delta + 3) \cdot |OPT|$.

Proof: Define a_i as the number of pieces left after i^{th} iteration, and $a_0 = n$. Since a node can connect up to Δ pieces, $|OPT| \geq \frac{a_0}{\Delta}$. Consider $i + 1^{st}$ iteration. The optimal solution can connect a_i pieces. Hence the greedy procedure is guaranteed to pick a node which connects at least $\left\lceil \frac{a_i}{|OPT|} \right\rceil$ pieces. This gives us the recurrence relation,

$$a_{i+1} \leq a_i - \left\lceil \frac{a_i}{|OPT|} \right\rceil + 1 \leq a_i(1 - \frac{1}{|OPT|}) + 1 \; .$$

Its solution is,

$$a_{i+1} \leq a_0(1 - \frac{1}{|OPT|})^i + \sum_{j=i}^{i-1}(1 - \frac{1}{|OPT|})^j \; .$$

Notice after $|OPT| \cdot \ln \frac{a_0}{|OPT|}$ iterations, the number of pieces left is less than $2 \cdot |OPT|$. For each node we choose, we will decrease the number of pieces by at least one. This will continue until the number of black components is at most $|OPT|$, thus at most $|OPT|$ more vertices are picked.

Assume from this point onwards, we stop after choosing a_f more nodes. The number of pieces left to connect is at most $|OPT| - a_f$. We connect the remaining pieces choosing chains of two vertices in the second phase. The total number of nodes chosen is at most $|OPT| \cdot \ln \frac{a_0}{|OPT|} + |OPT| + a_f + 2(|OPT| - a_f)$, and since $|OPT| \leq \frac{a_0}{\Delta}$, the solution found has at most $|OPT| \cdot (\ln \Delta + 3)$ nodes. □

Remark: Berman, [2], has an alternate proof of $H(\Delta) + 2$ of the same algorithm. However, since $\ln \Delta \approx H(\Delta) - 0.7$, the difference is very small.

4 Generalizations

4.1 Vertex Weighted Graphs

An approximation factor of $3 \ln n$ is possible when the vertices have weights. The algorithm first finds a dominating set, and then connects the nodes in the dominating set.

Step 1. Use a weighted set cover approximation algorithm to find a dominating set DS. (A set cover instance is created by making each vertex an element, and each vertex corresponds to a set that contains the vertex itself, together with its neighbors. The greedy algorithm picks sets based on the ratio of their weight to the number of new elements they cover.)

Step 2. To connect the vertices in DS we use a node-weighted Steiner tree approximation algorithm due to Klein and Ravi [10] to find a Steiner tree that includes all the vertices in DS, after making the weights of all vertices in DS equal to zero. This yields a connected dominating set CDS.

Theorem 4.1 *The weight of vertices in CDS is at most $3 \ln n \cdot |OPT|$ where OPT is the minimum weight connected dominating set in G.*

Proof: The weight of the vertices in DS is at most $\ln \Delta \cdot |OPT|$. We now run the algorithm by Klein and Ravi [10] for the node-weighted Steiner tree case. The approximation factor of the algorithm is $2 \ln k$, where k is the number of Steiner vertices. Consider the vertices in OPT; these together with the vertices in DS induce a connected subgraph. Hence there exists a node weighted Steiner tree of weight OPT. The total weight of the vertices in the connected dominating set is the weight of DS together with the weights of optional vertices chosen from G in the Steiner instance. Adding the weight of the two sets gives the required bound. □

Before looking at other generalizations, we first consider a problem closely related to our discussion.

4.2 Unit Node Weighted Steiner Trees

The best known algorithm for node weighted Steiner trees, has a performance ratio of $2 \ln k$, where k is the number of required vertices [10]. However, if the nodes have unit weight, there is a simpler algorithm, which gives a better performance ratio.

We have k required vertices in a graph $G = (V, E)$, which we want to connect using the least number of non-required vertices. We assume that the non-required vertices have weight 1, and the required vertices have weight 0.

Our algorithm runs in two phases. In the first phase, the algorithm greedily picks high degree stars (a star is a vertex that has at least two required vertices belonging to distinct components as neighbors) and merges them, until very few components are left. In the second phase, the algorithm runs a Steiner tree approximation algorithm with each edge having unit weight.

In a preprocessing phase we merge all adjacent required vertices into their connected components. We pick $\lambda = 2c + 1$ where c is the best approximation ratio for the unweighted Steiner tree problem.

Algorithm A

Step 1. In each iteration choose a vertex that merges the largest number of required vertices until we reach a stage that the number of components left to merge is less than $\frac{\textbf{iteration count}}{\ln k - \lambda} + e^{\lambda}$ or no merging is possible.

Step 2. Apply an (edge weighted) Steiner tree approximation algorithm, with each edge having unit weight.

Theorem 4.2 *The above algorithm finds a solution to the unit node weighted Steiner tree (UNST) problem with an approximation factor of* $\ln k$ *(which is best possible), when the optimal solution is greater than* $c \cdot e^{\lambda}$.

Proof: Assume that the set of components remaining after the first phase is A'. We claim that there is a Steiner tree with $|A'| + |OPT|$ edges. Thus when we apply an (edge weighted) Steiner tree approximation, we get a tree with at most $c \cdot (|A'| + |OPT|)$ edges.

If the number of iterations in the first phase is r, the final solution has a cost $r + c \cdot (|A'| + |OPT|)$. We now proceed to give a bound on r.

Let a_i components be left after i^{th} iteration. Since $|OPT|$ nodes are capable of merging these components, for each i, in the i^{th} iteration, there must be a node that merges $\left\lceil \frac{a_{i-1}}{|OPT|} \right\rceil$ components. This gives a bound on a_i,

$$a_i \leq a_{i-1} - \left\lceil \frac{a_{i-1}}{|OPT|} \right\rceil + 1 \leq a_{i-1}(1 - \frac{1}{|OPT|}) + 1.$$

We can easily verify that $a_i \leq a_0 \cdot (1 - \frac{1}{|OPT|})^i + \sum_{j=0}^{i-1}(1 - \frac{1}{|OPT|})^j$. The second term is a geometric series that sums to at most $|OPT|$. Thus when $i = (\ln k - \lambda) \cdot |OPT|$ the first term is at most e^λ, and the number of components $a_i \le |OPT| + e^\lambda \le \frac{i}{\ln k - \lambda} + e^\lambda$. This guarantees that the number of iterations, $r \le (\ln k - \lambda) \cdot |OPT|$.

If we stop because no merging by stars is possible, then the components have disjoint neighborhoods, and OPT has to pick at least one vertex from each neighborhood. Thus $|A'| \le |OPT|$. If we stop because the number of components is small, then $|A'| \le |OPT| + e^\lambda$. In any case, $|A'| \le |OPT| + e^\lambda$ and this yields a solution of cost at most $\ln k \cdot |OPT| + c \cdot e^\lambda + (2c - \lambda)|OPT|$. Putting $\lambda = 2c + 1$ gives at most $\ln k \cdot |OPT|$ vertices in our solution (when $|OPT| \ge c \cdot e^{2c+1}$. □

The optimality of this approximation ratio was established by Berman (see [10]).

We can modify the above algorithm, to run until no further merging is possible.

Algorithm B

Step 1. In each iteration choose a vertex that merges the largest number of required vertices (at least two).

Step 2. Apply an (edge weighted) Steiner tree approximation algorithm, with each edge having unit weight.

Theorem 4.3 *The above algorithm finds a solution to the unit node weighted Steiner tree (UNST) problem with an approximation factor of* $\ln \Delta + 2c + 1$.

Proof: As before, let a_i denote the number of vertices left after the i^{th} iteration and $a_0 = n$. Then after $|OPT| \cdot \ln \frac{a_0}{|OPT|}$, there are at most $2 \cdot |OPT|$ components to connect. Hence we will continue to merge by stars for $|OPT|$ more iterations then the number of components will be definitely less than $|OPT|$.

Since each Steiner vertex can be adjacent to at most Δ required vertices, $|OPT| \ge \frac{a_0}{\Delta}$.

If at this stage we use a_f more iterations before invoking the edge weighted Steiner tree algorithm, there is a tree with $|OPT| - a_f + |OPT|$ edges. So we find a solution of cost at most $c \cdot (|OPT| - a_f + |OPT|)$. The final solution has at most $|OPT| \cdot \ln \frac{a_0}{|OPT|} + |OPT| + a_f + c \cdot (|OPT| - a_f + |OPT|)$ nodes. Since $|OPT| \ge \frac{a_0}{\Delta}$, we get a performance guarantee of $\ln \Delta + 2c + 1$ for the algorithm.

□

4.3 Dominating a Subset of Vertices

We now address the connected dominating set problem when we are required to dominate only a specified subset S of the vertices. The cost of the solution is the size of the smallest connected dominating set that dominates the vertices in S. (Notice that the objective function is slightly different from the unit node weighted Steiner tree problem, where required vertices have zero cost. In the Steiner CDS problem, we are charged for all vertices in the final solution that are not leaf nodes in the tree that connects S.)

Unweighted Graphs

Let $|S| = k$, and OPT denote the optimal solution. we present two algorithm that solve this problem. A straightforward strategy is to first find a small dominating set A, of the vertices in S, and to then connect these nodes.

Algorithm A

Step 1. Greedily choose a dominating set of the vertices in S. We can transform this to a set cover problem in which corresponding to each vertex v we have a set that includes all its neighbors and itself. The greedy algorithm for set cover yields a dominating set A.

Step 2. For each element in A choose a representative element in S that is adjacent to it. Call this set $R(A)$. Run the unit node weighted Steiner tree approximation algorithm to find a Steiner tree with required set $R(A)$. The final solution is the union of A, $R(A)$, and the Steiner tree vertices.

Theorem 4.4 *The connected dominating set for the subset S of size k, is at most $3 \ln k$ times the optimal.*

Proof: Since we chose the cover greedily, we have that $|A| \leq |OPT| \cdot \ln k$, since OPT forms a dominating set for S.

Notice that $|A| \leq k$. We cannot claim that there is a Steiner tree of size $|OPT|$ connecting the set A. But there is a Steiner tree of size $|OPT|$ connecting the elements of set $R(A)$, since the connected dominating set also forms a Steiner tree on the members of S, and $R(A) \subseteq S$. Notice that $|R(A)| \leq |A| \leq k$. Apply Theorem 4.2, and obtain a Steiner tree of $R(A)$, of size at most $|OPT| \cdot \ln |R(A)|$. So the final solution is of cost less than $|A| + |R(A)| + |OPT| \cdot \ln |R(A)| \leq 3 \ln k \cdot |OPT|$. □

Algorithm B

Step 1. We modify the greedy set cover algorithm on the set S, to run until no vertex covers more than one uncovered vertex of S. We call the set of vertices chosen as B.

Step 2. We now choose the uncovered vertices of S , calling this set B'.

Step 3. For each member of B, choose a representative element of S that it dominates. Let this set be R(B). We apply an (edge weighted) Steiner tree approximation, with the set of required nodes as $R(B) \cup B'$. The final solution is the nodes of this tree and the nodes of B.

Theorem 4.5 *The connected dominating set for the subset S, is at most $(c+1)H(\delta) + c - 1$ times the optimal (where c is the Steiner ratio). We define δ as the size of the largest subset of S, adjacent to a node in the graph ($\delta \leq \min(\Delta, k)$).*

Proof: By a slight modification to the proof given in [5, page 977] we can prove, $|B| \leq (H(\delta) - 1) \cdot |OPT|$. (Since the first step reduces to finding a set cover with the size of the largest set being δ). Since OPT cannot dominate any two vertices of B' by one vertex, $|B'| \leq |OPT|$. Notice $B \cup B'$ dominates the set S.

Consider the set $R(B)$; there is a Steiner tree with $|R(B)|+|B'|+|OPT|$ edges that connects the nodes of $R(B) \cup B'$.

Apply an (edge weighted) Steiner tree approximation, with all edges having unit weight, and find a tree of size $c \cdot (|R(B)| + |B'| + |OPT|)$, where c is the Steiner ratio [12]. Since this tree is edge weighted, it has essentially the same number of nodes, including those of $R(B) \cup B'$. Since we have to add the vertices of B as well, we get an upper bound of $c \cdot (|R(B)|+|B'|+|OPT|)+|B|$. Notice that $|R(B)| \leq |B| \leq (H(\delta) - 1) \cdot |OPT|$, and $|B'| \leq |OPT|$. This gives us a solution of cost at most $((c+1) \cdot H(\delta) + c - 1) \cdot |OPT|$. □

This definitely is a better algorithm in terms of the worst case approximation guarantee. However the first algorithms is simpler and faster. Most of the approximation algorithms that reduce the Steiner ratio below 2, have a high running time [4, 12].

5 Lower Bounds

5.1 Hardness result for Connected Dominating Set

We can prove that the set-cover problem can be reduced to the connected dominating set problem by an approximation preserving reduction, thus showing that the approximation factor $H(\Delta)$ will be hard to improve. This is based on the hardness results for set cover proven by Lund and Yannakakis [13] and Feige [6].

Given a set cover instance we reduce it to a connected dominating set problem as follows:

Let the set cover instance be to cover the set U, with minimum number of sets from the collection $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$.

Construct a graph G, that has vertex set $U \bigcup \{u, v, v_1, v_2, \ldots, v_m\}$. An element $e \in U$, and v_i has an edge joining them iff $e \in S_i$. Each v_i has an edge to v. u has an edge only to v. (see Fig. 2)

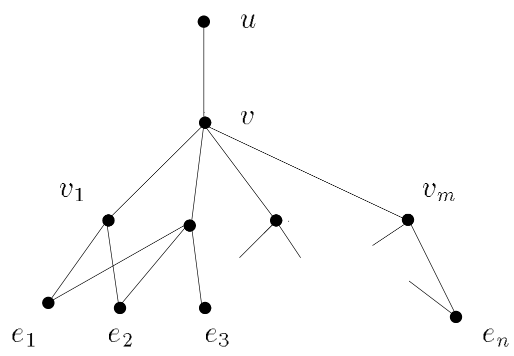

Figure 2: Reduction of set cover to connected dominating sets.

Let us look at a minimum connected dominating set of G. Vertex v belongs to any connected dominating set, and hence u does not belong to any minimal connected dominating set.

No vertex e_j is chosen in a minimal connected dominating set, since any node that it might potentially dominate, is already dominated by v, which also provides the connectivity. Hence we will only have v and some v_i's. These v_i's will correspond to the minimum cover for the given instance of set cover.

The size of the connected dominating set is one more than the minimum set cover. Thus approximating the connected dominating set with a factor of $(1-\epsilon)H(\Delta)$ would mean approximating minimum set cover within the same factor. This would imply that $NP \subseteq DTIME[n^{O(\log\log n)}]$ [6].

5.2 Hardness results for Generalizations

We show two simple reductions, that demonstrate that other generalizations of the CDS problem may be as hard to approximate as the "set TSP" problem for which no approximation algorithms are known. (For the Euclidean case, Mata and Mitchell [14] have given approximation algorithms for this problem.)

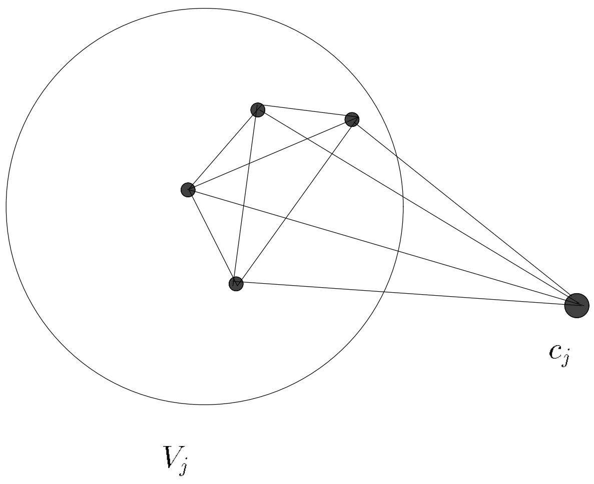

Figure 3: Reduction of set TSP problem to edge weighted CDS

Theorem 5.1 *A polynomial approximation algorithm for the edge weighted connected dominating set problem with factor $f(n)$ would imply a polynomial approximation algorithm for the set TSP problem with factor $2f(n)$.*

Proof: We show how to reduce the set TSP problem to the edge weighted connected dominating set problem. Consider a set TSP instance $G = (V, E)$ where $V = (V_1 \cup V_2 \cup \ldots \cup V_k)$. For each subset V_j, introduce a special vertex c_j, and add edges from c_j to all $v \in V_j$, with very high cost edges. For $u, v \in V_j$, if $(u, v) \notin E$, add the edge (u, v) with very high cost. Call this new graph G'.

Any set TSP tour in G chooses at least one vertex of V_j to visit. Thus all nodes of $V_j \cup \{c_j\}$ will be dominated by the corresponding node in the tour. Since every node of G occurs in some V_j, this yields a dominating set. Since these are nodes on a tour, they also form a connected set. Hence $OPT_{CDS} \leq OPT_{TOUR}$.

If we have a connected dominating set of G', then it must have a vertex of V_j to dominate c_j. Hence the dominating set must have at least one vertex from each set V_j. If the cost of this connected dominating set is small ($\leq f(n)OPT_{CDS}$), since we are not using the high cost edges in G', we are using only the edges of the graph G. By traversing this tree twice, we can produce a tour in G, with cost at most $2f(n)OPT_{CDS} \leq 2f(n)OPT_{TOUR}$. Thus, if we can approximate the connected dominating set with edge weights to a factor $f(n)$, we can approximate set TSP within a factor $2f(n)$. □

Theorem 5.2 *A polynomial approximation algorithm for the node weighted Steiner connected dominating set problem with factor $f(n)$ would imply a polynomial approximation algorithm for the set TSP problem with factor $2f(n)$.*

Proof: The proof is similar to the proof of the previous theorem. Given a set TSP instance $G = (V, E)$ where $V = (V_1 \cup V_2 \cup \ldots \cup V_k)$ we construct a graph G'. First convert the edge weights of the set TSP problem into node weights. For every edge $e = (v_p, v_q) \in E$, create an extra node v_{pq} of the same cost, connected to v_p and v_q. All other nodes are given 0 cost. For every subset V_j, introduce a special vertex c_j (of very high cost), and connect it to all $v \in V_j$. We show that the problem reduces to finding a node weighted connected dominating set of the subset $U = \{c_j | j = 1 \ldots k\}$ of nodes of G'.

Any set TSP tour in G chooses at least one vertex of V_j to visit. Thus each c_j will be dominated. The weight of the edges $e = (v_p, v_q)$ translates to the weight of the corresponding vertices v_{pq}. Since the nodes form a tour, they also form a connected set in G', together with the new nodes that subdivide edges. Thus $OPT_{CDS} \leq OPT_{TOUR}$.

Consider a connected dominating set that dominates U. To dominate c_j, it must pick a vertex from V_j. (W.l.o.g, the connected dominating set does not contain c_j.) If the cost of this connected dominating set is small ($\leq f(n)OPT_{CDS}$), (since we are not using the high cost nodes in G'), we are using only the nodes of the graph G along with nodes that correspond to the subdivided edges. Thus the dominating set chooses vertices that are also in G, and the corresponding vertices for each edge of G that it includes. This yields a tree that connects at least one element from each V_j using edges of G. By traversing this tree twice, we can produce a tour in G, of cost at most $2f(n)OPT_{CDS} \leq 2f(n)OPT_{TOUR}$. Thus, if we able to approximate the connected dominating set of a subset with node weights to a factor $f(n)$, we can approximate set TSP within a factor $2f(n)$. □

Acknowledgements: We thank Ray Miller and Azriel Rosenfeld for first mentioning the traveling tourists problem. We thank Vaduvur Bharghavan for re-igniting our interest in the connected dominating sets problem. We would like to thank Estie Arkin, and Randeep Bhatia for useful discussions. We thank Balaji Raghavachari and Serge Plotkin for raising the questions about the vertex weighted case, and the Steiner case respectively. We thank Piotr Berman for allowing us to include his improvement to Algorithm II.

References

[1] E. Arkin, M. Halldórsson and R. Hassin, "Approximating the tree and tour covers of a graph", *Information Processing Letters*, 47: 275–282, (1993).

[2] P. Berman, personal communication, May (1996).

[3] P. Berman and M. Fürer, personal communication, May (1996).

[4] P. Berman and V. Ramaiyer, "Improved approximation algorithms for the Steiner tree problem", *J. Algorithms*, 17:381–408, (1994).

[5] T. Cormen, C. Leiserson, and R. Rivest, *Introduction to Algorithms*, The MIT Press, 1989.

[6] U. Feige, "A threshold of $\ln n$ for approximating set-cover", *28th ACM Symposium on Theory of Computing*, pages 314–318, (1996).

[7] M. R. Garey and D. S. Johnson, "Computers and Intractability: A guide to the theory of NP-completeness", *Freeman, San Francisco* (1978).

[8] S. Guha and S. Khuller, "Approximation algorithms for connected dominating sets", *Proc. of 4th Annual European Symposium on Algorithms*, (1996).

[9] A. Kothari and V. Bharghavan, "Algorithms for unicast and multicast routing in ad-hoc networks", manuscript.

[10] P. N. Klein and R. Ravi, "A nearly best-possible approximation algorithm for node-weighted Steiner trees", *J. Algorithms*, 19(1):104–114, (1995).

[11] D. Kleitman and D. West, "Spanning trees with many leaves", *SIAM Journal on Discrete Mathematics*, 4(1):99–106, (1991).

[12] M. Karpinsky and A. Zelikovsky, "New approximation algorithms for the Steiner tree problem", *Technical Report, Electronic Colloquium on Computational Complexity* (ECCC): TR95-030, (1995).

[13] C. Lund and M. Yannakakis, "On the hardness of approximating minimization problems", *Journal of the ACM*, 41(5): 960–981, (1994).

[14] C. S. Mata and J. S. B. Mitchell "Approximation algorithms for geometric tour and network design problems", *Proc. of the 11th Annual Symp. on Computational Geometry*, pages 360–369, (1995).

[15] S. Paul and R. Miller, "Locating faults in a systematic manner in a large heterogeneous network", *IEEE INFOCOM*, pages 522–529, (1995).

[16] C. Savage, "Depth-First search and the vertex cover problem", *Information Processing Letters*, 14(5): 233–235, (1982).

[17] P. Slavík "A tight analysis of the greedy algorithm for set cover" *28th ACM Symposium on Theory of Computing*, pages 435–441, (1996).