

1 Introduction

The general theme of today's presentation was the global infrastructure of wireless networks. We covered two papers: "Constant-Time Distributed Dominating Set Approximation," by F. Kuhn and R. Wattenhoffer, 2003; and "What Cannot Be Computed Locally," by F. Kuhn, T. Moscibroda and R. Wattenhoffer, 2004.

2 The "Minimum Dominating Set" problem

A *dominating set* is a set of nodes S such that every node in the network graph G is a neighbor of at least one element of S . Here is an example.

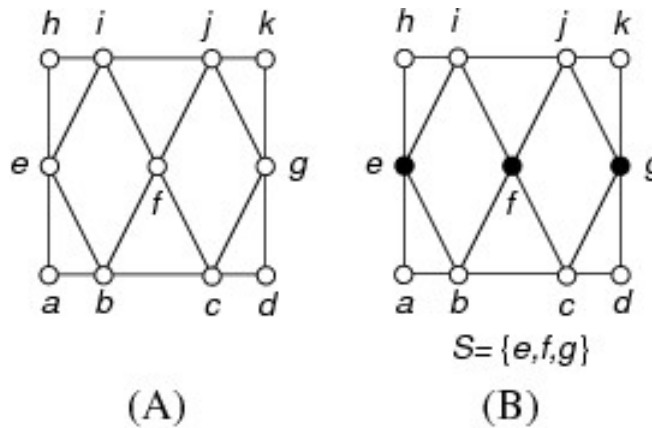


Figure 1: (A) is the network graph. (B) shows the dominating set S .

The Minimum Dominating Set (MDS) problem is to find a minimum such S for a given network graph. It is the fundamental math problem underlying routing: construct "cluster heads" in the network graph so you can rapidly, hierarchically, send messages from one cluster to another, and the cluster heads will oversee routing within and through the individual cluster. Of course, the members of S are the routers or cluster heads. So MDS asks, "Can we find the smallest set of efficient routers?"

Finding a minimum dominating set is NP-hard in general, but efficient approximation algorithms do exist. However, these approximation algorithms require a central brain capable of performing global computation, so the question of interest to us is whether there is a *local, practical* algorithm for MDS.

In our context, “local” computation means computation that can be performed in a distributed manner, in which each node only communicates with its immediate neighbors (i.e., no multi-hop packet forwarding). “Practical” means the computation (1) doesn’t take too much time, and (2) doesn’t use much bandwidth.

3 Main result of first paper

The main result of the first paper (“Constant-time Distributed Dominating Set Approximation”) is to demonstrate the first local, distributed algorithm that computes a nontrivial approximation of a minimal dominating set in a constant number of rounds.

Formally, we define the following parameters:

- Δ maximum node degree of the network graph
- k an arbitrary parameter (more explanation below)
- DS_{opt} the cardinality of the optimal (i.e., minimal) dominating set

The main algorithm of the paper computes a dominating set of size $\mathcal{O}(k\Delta^{2/k} \log \Delta |DS_{opt}|)$ in $\mathcal{O}(k^2)$ rounds, where each node has to send $\mathcal{O}(k^2\Delta)$ messages of size $\mathcal{O}(\log \Delta)$.

As the paper’s title indicates, the algorithm takes only a constant number of rounds, regardless of the size of the graph. (Unlike many other algorithms we have studied, the parameter n , number of nodes, does not appear in the size or time bounds.) The parameter k can be set depending on the needs of the particular application. If k is larger, the algorithm will run longer, and produce an approximation closer to the value of $|DS_{opt}|$. If k is smaller, the approximation will not be as good, but the algorithm will halt sooner.

Also important is the algorithm’s relatively small use of bandwidth. The messages are on the scale of the log of the maximum node degree—so for most network graphs, the size of the message packet will be quite small relative to the diameter of the graph.

An example of a “trivial” dominating set would be G , the entire network graph itself. Clearly all nodes are neighbors of some node in G . The algorithm’s approximation will be off the optimal value by a factor only of k and Δ , which are (presumably) both small relative to the size of the network graph, so the size of the approximation is nontrivially close to the optimal value.

4 Main proof technique of first paper

The main proof technique of the first paper is as follows.

1. Construct an “integer program” whose solution is equivalent to solving MDS.
2. “Relax” the integer program to a linear program.
3. Demonstrate a local, distributed algorithm that computes an approximate solution to the linear program.

4. Demonstrate a local, distributed algorithm which, given an approximate solution to the linear program, computes an approximate solution to the integer program (and hence to the MDS problem).
5. Conclude that the algorithm is a local, practical method of approximating MDS.

4.1 Linear Programming Relaxation of the MDS Problem

Recall that an integer program looks like a linear program optimization problem, except that the feasible solution values must be integer points. Solving an integer program is NP-hard, in general. By contrast, it is feasible (via well-studied techniques such as the simplex or ellipsoid methods) to solve linear programs. Figure 2 provides a graphic example of the relation between an integer program, and its linear program relaxation.

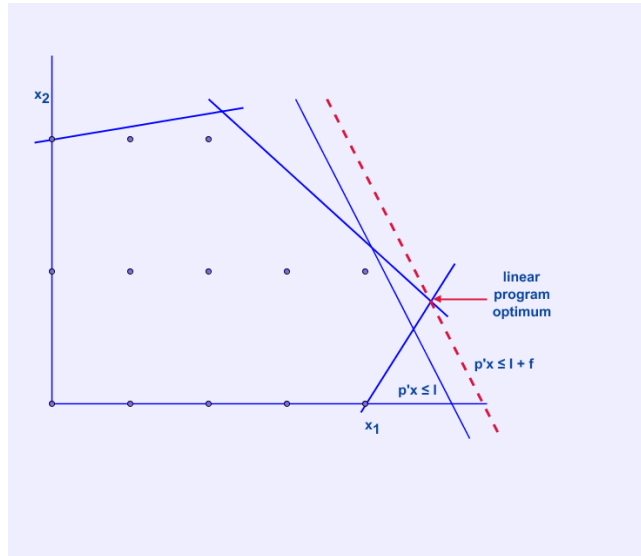


Figure 2: The lattice points are the feasible values for an integer program. The relaxed linear program solution is near one of the feasible points.

We can define an integer program version of MDS, IP_{MDS} , by modifying the adjacency matrix of the network graph to construct a “neighborhood matrix” which we will denote by N . We will say that each node is a neighbor of itself, so N is identical to the adjacency matrix, except that it has 1’s along its principal diagonal. Let n be the number of nodes in the network graph, and let $\{x_i\}_{i \leq n}$ be a set such that $x_i = 1$ if node i is in the dominating set we have chosen, and $x_i = 0$ otherwise. We use vector notation, for example writing $\vec{x} = \vec{0}$ to say that $\forall i[x_i = 0]$.

We can then write IP_{MDS} formally as follows.

$$\begin{aligned} \min \sum_{i=1}^n x_i \\ N \cdot \vec{x} \geq \vec{1} \\ \vec{x} \in \{0, 1\}^n \end{aligned}$$

In words, we choose the fewest possible x_i such that every node in the graph is a neighbor of some x_i .

To relax this integer program, we allow each x_i value to be greater than 0, instead of requiring either 0 or 1. In symbols:

$$\begin{aligned} \min \sum_{i=1}^n x_i \\ N \cdot \vec{x} \geq \vec{1} \\ \vec{x} \geq \vec{0}. \end{aligned}$$

4.2 Solving IP_{MDS} given LP_{MDS}

Let's jump ahead in the process a bit, to consider the simplest algorithm first. Suppose we have a good approximation of LP_{MDS} . Algorithm 1 gives us a good approximation for IP_{MDS} . The algorithm's pseudocode appears as Figure 3.

<p>Algorithm 1 $LP_{MDS} \rightarrow IP_{MDS}$</p> <p>Input: feasible solution $\underline{x}^{(\alpha)}$ for LP_{MDS}</p> <p>Output: IP_{MDS}-solution \underline{x}_{DS} (dom. set)</p> <p>1: calculate $\delta_i^{(2)}$</p> <p>2: $p_i := \min\{1, x_i^{(\alpha)} \cdot \ln(\delta_i^{(2)} + 1)\}$</p> <p>3: $x_{DS,i} := \begin{cases} 1 & \text{with probability } p_i \\ 0 & \text{otherwise} \end{cases}$</p> <p>4: send $x_{DS,i}$ to all neighbors</p> <p>5: if $x_{DS,j} = 0$ for all $j \in N_i$ then</p> <p>6: $x_{DS,i} := 1$</p> <p>7: fi</p>
--

Figure 3: The integer-program-approximation algorithm.

This algorithm is fully distributed, meaning each node i runs the identical code, and gets information only by talking to its immediate neighbors. The parameter $\delta_i^{(2)}$ is the maximum node degree of nodes within two hops of node i . Each node can discover that parameter after two rounds: in the first round, each node sends its own degree to all neighbors, and then in the second round, each node sends the maximum node degree it heard about to all neighbors.

Node i elects itself to the dominating set with probability dependent on the linear program approximation and the log of $\tilde{\delta}_i^{(2)}$. As a failsafe, at the end of all such elections, if a node hears that none of its neighbors are in the dominating set, it elects itself as a member of the dominating set.

This method produces a dominating set that is a logarithmic plus a constant factor greater than the optimal possible value.

4.3 Solving LP_{MDS}

So if we can solve the linear program, we can solve the integer program. Now let's examine how to solve the linear program.

As a first step, assume we know the parameter Δ , the maximum node degree of the graph. Pseudocode for an algorithm that solves LP_{MDS} given that information appears in Figure 4.

```

Algorithm 2  $LP_{MDS}$  approximation ( $\Delta$  known)
1:  $x_i := 0$ ;
2: for  $\ell := k - 1$  to 0 by  $-1$  do
3:   (*  $\tilde{\delta}(v_i) \leq (\Delta + 1)^{(\ell+1)/k}$ ,  $z_i := 0$  *)
4:   for  $m := k - 1$  to 0 by  $-1$  do
5:     (*  $a(v_i) \leq (\Delta + 1)^{(m+1)/k}$  *)
6:     send  $color_i$  to all neighbors;
7:      $\tilde{\delta}(v_i) := |\{j \in N_i \mid color_j = \text{'white'}\}|$ ;
8:     if  $\tilde{\delta}(v_i) \geq (\Delta + 1)^{\ell/k}$  then
9:        $x_i := \max \left\{ x_i, \frac{1}{(\Delta+1)^{m/k}} \right\}$ 
10:    fi;
11:    send  $x_i$  to all neighbors;
12:    if  $\sum_{j \in N_i} x_j \geq 1$  then  $color_i := \text{'gray'}$  fi;
13:  od
14:  (*  $z_i \leq 1/(\Delta + 1)^{(\ell-1)/k}$  *)
15: od

```

Figure 4: Solution of the linear program.

This algorithm paints nodes one of two colors: white or gray. All nodes are initially white. Once node i elects itself to the dominating set, i and all its neighbors get colored gray. Demonstrating that the upper bound of the size of the dominating set is nontrivially close to the optimum value requires a lot of technical manipulation, but the critical point is the use of the “dynamic degree” parameter, $\tilde{\delta}(v_i)$.

The dynamic degree of node i is the number of neighbors of i that are currently colored white. If that value is greater than a certain root of $\Delta + 1$, then it is more probable that node i needs to be included in the dominating set. It is a use of global parameter Δ and local neighborhood parameter $\tilde{\delta}(v_i)$ to allow an individual node to make a local decision.

Of course, if we want a 100% locally-executable algorithm, we can't assume access to a global parameter like Δ . That's OK, because using the method of inductive counting, we can build an algorithm that will work correctly just knowing $\Delta_{G'}$, for $G' \subsetneq G$ (where G is the network graph), and then expanding the diameter of G' step-by-step. Conceptually, it is very similar to a proof that we did recently in Com S 531: **NL=co-NL**, by showing that UNREACHABILITY is in **NL**. The technical details of this construction are more complicated, though.

5 Concluding remarks about first paper

This is the first algorithm that achieves a nontrivial approximation ration for the MDS problem in a constant number of rounds. Part of the algorithm's strength is that it divides up all the work into fully local, distributed steps, so few messages need be transmitted.

6 Addendum: Connected Dominating Set

There is another type of find-the-dominating set algorithm that is used in networks: construct a dominating set D that is connected. This set D can then function as a communication backbone for the network, and in fact multiple such D can be computed so that each backbone is used for a while, then gets to rest, so the nodes do not lose too much power.

7 “Minimum Vertex Cover” problem

The second paper considers the Minimum Vertex Cover problem (MVC). MVC is tightly related to MDS. The idea of MVC is: given a network graph G with edge set E , find a minimal set S of nodes such that for each edge $(u, v) \in E$, at least one of u, v is a member of S . Like MDS, MVC is useful for routing algorithms. It also has application in fault-tolerance algorithms, and in detecting illegal configurations of possible network graphs.

Another reason for MVC's importance is that many other graph-theoretic problems reduce to it. Two examples are Maximal Independent Set and Maximal Matching. These problems are defined as follows:

Maximal Independent Set Find a maximal set of nodes that are not connected to one another by any edge.

Maximal Matching Find a maximal set of edges that do not share common endpoints.

Finding a minimum dominating set is strictly harder computationally than finding a minimum vertex cover. This is true in both single-processor and distributed environments. MDS is equivalent to the “generalized set cover problem”, which presents an arbitrary collection C of sets of nodes of G , and asks for the minimal family $F \subseteq C$ such that F covers all nodes of G . MVC is the special case of this problem in which $C = \{\{p_1\}, \{p_2\}, \dots\}$ for each processor $p_i \in G$.

8 What cannot be computed locally!

Unlike the first paper, which provided a positive solution to a longstanding problem, this second paper proves a limitation to local computation. In particular, it provides a lower bound for the estimation of minimal dominating set. If n is the number of nodes in the graph, and c is a constant greater than $1/4$, then after k communication rounds, the optimal value of MVC can only be approximated by a set of size $\Omega(n^{c/k^2}/k)$.

9 Main proof technique: “Equality of Views”

Conceptually, this argument is very similar to the clock synchronization lower-bound proofs we studied toward the beginning of the course. Even the bounds obtained are similar. Consider the Fan/Lynch gradient clock synchronization lower bound is that the clock skew for two nodes distance d apart in a graph of diameter D . The worst case clock skew will be on the order of

$$\Omega\left(d + \frac{\log D}{\log \log D}\right).$$

Compare this with the results in our second paper, which state that to get a good approximation ratio for MVC, there are graphs that require time

$$\Omega\left(\sqrt{\frac{\log n}{\log \log n}}\right)$$

where n is the number of nodes in the graph, or time

$$\Omega\left(\frac{\log \Delta}{\log \log \Delta}\right)$$

where Δ is the maximum degree of the graph.

The key idea underlying both the gradient clock synchronization argument and the current “Equality of Views” argument is this: in k rounds of communication, nodes can only gather information from other nodes that are at most k hops away.

The second paper uses this fact, along with the existence of a certain type of graph, to construct a situation such that two different nodes, which optimally should make different decisions about whether to be in the Vertex Cover Set, have exactly the same view of the network graph (i.e., identical information) and so are not both able to decide correctly at the end of round k . The specific graph used is complicated, and the proof of its existence is highly abstract, but one key idea relates to the *girth* of a graph. The girth of G is the length of the shortest cycle in G . For this argument, if $\text{girth}(G) \geq 2k+1$, then after k rounds the graph will still appear like a tree for two nodes at opposite sides of the same cycle. See Figure 5.

A vertex v_{C_0} in C_0 and a vertex v_{C_1} in C_1 will have equal views of the network graph after k rounds. Pictorially, their views are represented in Figure 6.

10 Conclusion

In this presentation, we have considered both positive and negative results about local, distributed computation. On the one hand, there is a new, feasible, practical algorithm for the approximate MDS problem. On the other hand, there are worst-case scenarios that demonstrate an absolute lower-bound for computation of MVC. A deterministic algorithm requires global information to get a good approximation of MVC in all cases; the problem cannot be computed locally.

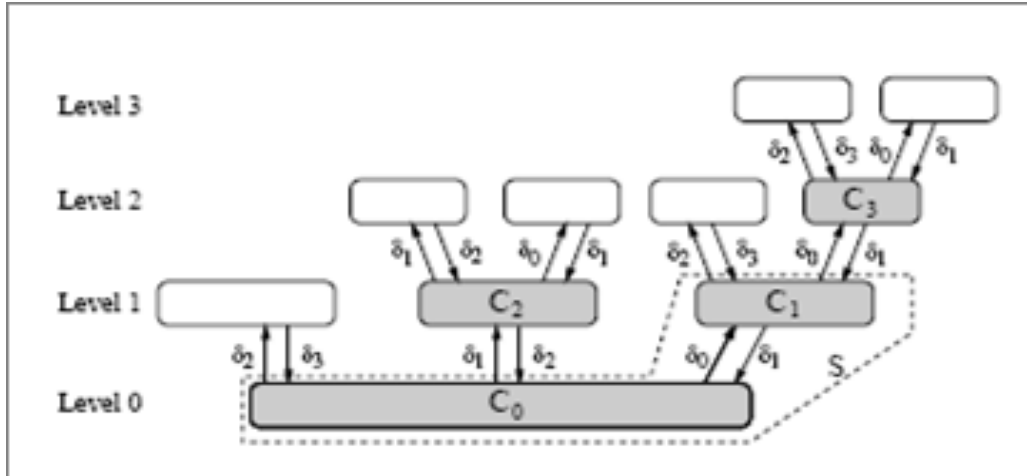


Figure 5: Consider one node in C_0 and another in C_1 . All nodes in C_0 are neighbors of nodes in C_1 , so a minimum dominating set does not need to include any member of C_0 . However, the nodes of C_0 cannot figure this out after k rounds, because their world-view is identical to that of the nodes in C_1 .

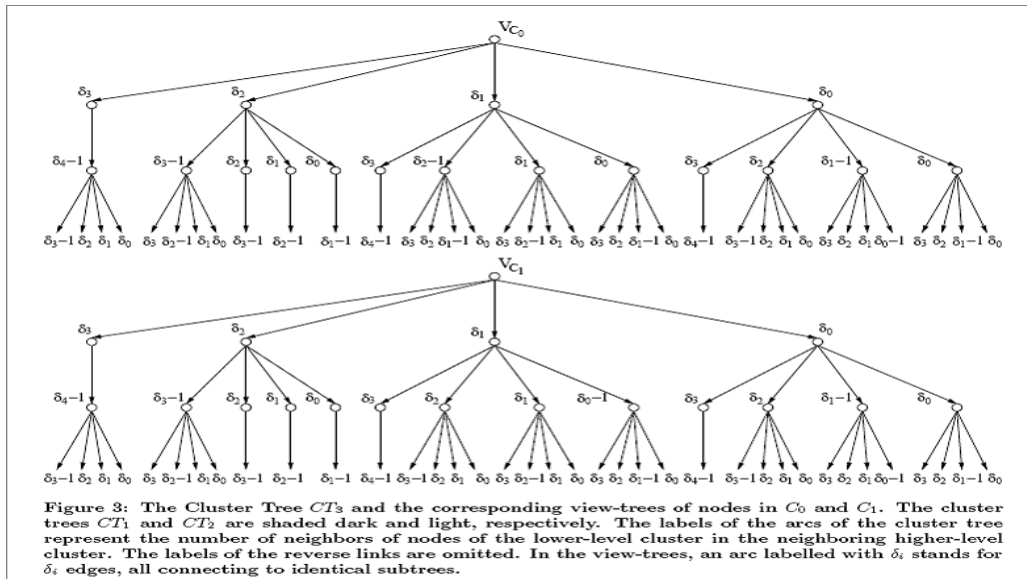


Figure 3: The Cluster Tree CT_1 and the corresponding view-trees of nodes in C_0 and C_1 . The cluster trees CT_1 and CT_2 are shaded dark and light, respectively. The labels of the arcs of the cluster tree represent the number of neighbors of nodes of the lower-level cluster in the neighboring higher-level cluster. The labels of the reverse links are omitted. In the view-trees, an arc labelled with δ_i stands for δ_i edges, all connecting to identical subtrees.

Figure 6: The equality of views of v_{C_0} and v_{C_1} .