Global Infrastructure

Readings:

Global infrastructures
Elkin: Distributed approximations—a survey
Kuhn, Wattenhofer: Distributed dominating set approximation
Kuhn, Moscibroda, Wattenhofer: What cannot be computed locally!

Constant Time Distributed Dominating Set Approximation

1 Introduction

In the first part of this lecture, we study a distributed algorithm to compute an approximate minimum dominating set of a graph. Given a graph $G = (V, E)$, a *dominating set* of G is a set of nodes $V' \subseteq V$ such that every node in V is either in V', or is a neighbor of a node in V'. Our goal is to compute the smallest such set V'. We call such a set a *minimum dominating set (MDS)*.

Dominating sets have many applications in distributed computing, and especially in ad-hoc radio networks. One way they are used is in routing: given a graph G representing the communication graph of an ad-hoc network, we can choose the nodes of a dominating set to act as *cluster heads*. Subsequently, routing is done from cluster head to cluster head, as opposed to from node to node. This reduces the number of nodes we have to route through, and consequently improves the routing efficiency. The smaller the dominating set we use, the more the efficiency improves.

There are many sequential (centralized) algorithms for computing a graph's (approximate) MDS. Therefore, if we can collect the entire communication graph at a single node, then that node can compute an MDS and distribute the answer to the rest of the network. However, if the communication graph has diameter D, then such an algorithm takes at least $O(D)$ time, simply from the collection and broadcast portions. The Kuhn, Wattenhofer algorithm computes a dominating set in $o(D)$ time. Though the dominating set is in general not the smallest possible, it is guaranteed to be not much larger than the MDS. In particular, for any $k \geq 1$, the KW algorithm uses $O(k^2)$ time to compute a dominating set which is at most $O(k\Delta^{2/k} \log \Delta)$ times larger than the smallest dominating set of the graph. Here, Δ is the largest degree of any node in the graph. The algorithm consists of several parts. First, they formulate the MDS problem as a *linear program (LP)*. Then, they give a *distributed algorithm* to solve the LP. Lastly, they use *randomized rounding* to extract a dominating set from the solution to the LP. Before going over the steps of the algorithm, we first present some background information.

2 Background

Finding the MDS of a graph is known to be NP-complete. Furthermore, even finding a DS which is not much larger than the MDS is hard. We say an algorithm has *approximation ratio* $\alpha \geq 1$ for the

MDS problem if whenever a graph has an MDS of size s, the algorithm returns a dominating set of size at most αs. It has been shown that unless all NP problems can be solved in deterministic $n^{O(\log \log n)}$ time, then the smallest approximation ratio of any polynomial time algorithm for MDS is $\ln \Delta$. In fact, this approximation ratio is achieved by a simple greedy algorithm: use an iterative algorithm, where at each stage, we add the node with the highest degree to the dominating set, then remove the node and all its edges from the graph.

The greedy algorithm described above is centralized, because it needs to know the entire graph at each iteration. We are more interested in *round-based* distributed algorithms. In these algorithms, computation proceeds in rounds, where in each round, a node can send a (possibly different) message to each of its neighbors, and receive a message from each of its neighbors. We do not restrict the size of the messages, nor the amount of computation a node performs to compute the messages. Notice therefore, that in $O(D)$ rounds, where D is the diameter of the graph, we can compute an optimal dominating set by performing all computations at a centralized node. However, if we want to use $o(D)$ rounds, then the problem becomes more challenging.

There is a randomized distributed algorithm which computes an $O(\log \Delta)$-approximate DS using expected $O(\log n \log \Delta)$ rounds, where n is the number of nodes in the graph. The MDS problem is related to the problem of solving *positive* LP's, i.e., LP's where all the coefficients are positive. There are distributed algorithms which produce a $(1+\epsilon)$-approximation to an LP, for any $\epsilon > 0$, in a polylogarithmic number of rounds.

3 The Kuhn/Wattenhofer Algorithm

As stated earlier, the KW algorithm consists of three parts. First, we formulate MDS as an LP. Then we use a distributed algorithm to solve the LP. Finally, we use rounding to produce a dominating set from the solution to the LP. Below, we describe these three steps. We will actually describe third step before the second, since it is simpler.

3.1 Formulating MDS as an LP

MDS is formulated as a combinatorial problem, and many algorithms for solving MDS are combinatorial in nature. However, another powerful technique for solving MDS, and many other graph problems, is to formulate them algebraically, and then apply algebraic techniques. In particular, in this section we describe how to formulate MDS as a linear program. A *linear program (LP)* consists of optimizing a linear function subject to linear inequality constraints. In general, this can be expressed in the following way. Here, b, c and x are vectors, and A is a matrix.

$$\begin{aligned} \min \quad & c \cdot x^T \\ \text{subject to} \quad & Ax^T \geq b \end{aligned}$$

How can we model MDS as an LP? Let $G = (V, E)$ be a graph, and suppose we want to compute a dominating set S of G. For each node $v_i \in V$, let N_i denote the *neighbors* of v_i, including v_i itself. That is, $N_i = \{v_j \,|\, (v_i, v_j) \in E\} \cup \{v_i\}$. We will associate a variable x_i with every node $v_i \in V$. x_i will be set to a value in $[0, 1]$. The idea is that $x_i = 1$ if and only if v_i belongs to the dominating set. For S to be a dominating set, we need to satisfy the condition

$$\forall i \in [n] : \sum_{j \in N_i} x_j \geq 1$$

This follows because for every node v_i, we need at least one of v_i's neighbors to be in S. That is, we need there to be some $v_j \in N_i$ such that $x_j = 1$. Minimizing the size of the dominating set corresponds to minimizing the sum of the x_i's. Let $x = [x_1, x_2, \ldots, x_n]^T$ be the array of the x_i values, and let $\mathbf{0}, \mathbf{1}$ be the all 0 and all 1 vectors, respectively. Also, let N be the adjacency matrix of G. That is, $N_{i,j} = 1$ if $(v_i, v_j) \in E$, and $N_{i,j} = 0$ otherwise. Then, we can formulate the MDS problem in the following way:

$$\begin{aligned} \min \quad & \sum_{i=1}^{n} x_i \\ \text{subject to} \quad & N \cdot x \geq \mathbf{1} \\ & x \geq \mathbf{0} \end{aligned} \tag{LP1}$$

3.2 Rounding LP1 to an MDS

In the previous section, we formulated MDS as the linear program LP1. Before discussing how to solve LP1, we first describe how to relate a solution to LP1 to a solution to MDS. In fact, we will relate an *approximate* solution of LP1 to an *approximate* solution to MDS.

Let x^* be an optimal solution to LP1, and let $x^{(\alpha)}$ be an α-approximation for LP1. That is, $x^{(\alpha)}$ satisfies all the constraints of LP1, and

$$\sum_{i=1}^{n} x_i^{(\alpha)} \leq \alpha \sum_{i=1}^{n} x_i^* \tag{1}$$

Notice that $x^{(\alpha)}$ may contain fractional values. Our goal when creating LP1 was to include v_i in the dominating set S if and only if $x_i = 1$. To produce an integral vector from $x^{(\alpha)}$, we will round the i'th component of the vector to 1 with probability proportional to $x_i^{(\alpha)}$. This procedure is called *randomized rounding*. For any $i \in [n]$, let δ_i be the degree of node v_i. Also, let $\delta_i^{(1)}$ and $\delta_i^{(2)}$ be the *maximum degree* of any 1 and 2 hop neighbors of v_i. Consider the algorithm given in figure 1.

Let $S = \{i \,|\, x_{DS,i} = 1\}$ be the components of x_{DS} which are set to 1 after running algorithm 1. We check that S is a dominating set of G. That is, for any node $v \in V$, either $v \in S$, or v has a neighbor in S. This is easy to see. Indeed, line 5 of the algorithm checks, for every node v_i, whether none of v_i's neighbors are in the DS. If this is the case, line 6 adds v_i to the DS.

Let S^* be a MDS of G. We now show that S is not too much larger than S^*. Notice that because algorithm 1 is randomized, the size of S is a random variable. We have the following theorem.

Theorem 3.1 $E[|S|] \leq (1 + \alpha \ln(\Delta + 1))|S^*|$

That is, in expectation, S is at most $(1 + \alpha \ln(\Delta + 1))$ times larger than the size of an MDS of G. To prove theorem 1, we first show

Algorithm 1 $\text{LP}_{\text{MDS}} \longrightarrow \text{IP}_{\text{MDS}}$

Input: feasible solution $\underline{x}^{(\alpha)}$ for LP_{MDS}
Output: IP_{MDS}-solution $\underline{x}_{\text{DS}}$ (dom. set)

1: calculate $\delta_i^{(2)}$
2: $p_i := \min\{1,\ x_i^{(\alpha)} \cdot \ln(\delta_i^{(2)} + 1)\}$
3: $x_{\text{DS},i} := \begin{cases} 1 & \text{with probability } p_i \\ 0 & \text{otherwise} \end{cases}$
4: **send** $x_{\text{DS},i}$ to all neighbors
5: **if** $x_{\text{DS},j} = 0$ for all $j \in N_i$ **then**
6: $\quad x_{\text{DS},i} := 1$
7: **fi**

Figure 1: Producing a solution for MDS from a solution for LP1

Claim 3.2 Let x^* be an optimal solution to LP1. Then $|S^*| \geq \sum_{i=1}^n x_i^*$.

Proof. Create an n-vector $x^{(S^*)}$, where $x_i^{(S^*)} = 1$ if and only if $v_i \in S^*$. Then, it is easy to check that $x^{(S^*)}$ satisfies the constraints of LP1. Indeed, for any $i \in [n]$, let N_i be the i'th row of N, which gives the neighbors of v_i. Since S^* is a dominating set, we know that either $v_i \in S^*$, or some neighbor v_j of v_i is in S^*. In the first case, we have $N_i \cdot x^{(S^*)} \geq N_{i,i} \cdot x_i^{(S^*)} = 1 \cdot 1 = 1$. In the second case, we have $N_i \cdot x^{(S^*)} \geq N_{i,j} \cdot x_j^{(S^*)} = 1 \cdot 1 = 1$. Now, since x^* is an optimal solution to LP1, we have that $\sum_{i=1}^n x_i^* \leq \sum_{i=1}^n x_i^{(S^*)} = |S^*|$. ∎

We can now prove theorem 1.

Proof. A node can join S in either line 3 or 6 of algorithm 1. Let X be the number of nodes that join S in line 3, and let Y be the number of nodes that join S in line 6. We have

$$\begin{aligned} E[X] &= \sum_{i=1}^n p_i \\ &\leq \sum_{i=1}^n x_i^{(\alpha)} \cdot \ln(\delta_i^{(2)} + 1) \\ &\leq \ln(\Delta + 1) \sum_{i=1}^n x_i^{(\alpha)} \\ &\leq \alpha \ln(\Delta + 1) \sum_{i=1}^n x_i^* \\ &\leq \alpha \ln(\Delta + 1) |S^*| \end{aligned}$$

The first inequality follows because of line 2 of algorithm 1. The third inequality follows because of equation 1. The last inequality follows by claim 3.2.

To compute the expected value of Y, let q_i be the probability that none of node v_i's neighbors were added to S in step 3 of the algorithm. We compute

$$\begin{aligned} q_i &= \prod_{j \in N_i} (1 - p_j) \\ &\leq \prod_{j \in N_i} (1 - x_j^{(\alpha)} \cdot \ln(\delta_i^{(1)} + 1)) \\ &\leq \left(1 - \frac{\sum_{j \in N_i} x_j^{(\alpha)} \ln(\delta_i^{(1)} + 1)}{\delta_i + 1}\right)^{\delta_i + 1} \\ &\leq \left(1 - \frac{\ln(\delta_i^{(1)} + 1)}{\delta_i + 1}\right)^{\delta_i + 1} \leq e^{-\ln(\delta_i^{(1)} + 1)} \\ &= \frac{1}{\delta_i^{(1)} + 1} \end{aligned}$$

The expressions may look more complicated than they actually are. The first equality follows because the neighbors of v_i choose independently whether to join S, and the probability that $v_j \in N_i$ does not join S is $1 - p_j$. The second inequality follows from the arithmetic-geometric mean inequality, which states that for any set A of positive real numbers,

$$\prod_{x \in A} x \leq \left(\frac{\sum_{x \in A} x}{|A|}\right)^{|A|}$$

In our case, we set $A = N_i$, and $x = -x_j^{(\alpha)} \cdot \ln(\delta_i^{(1)} + 1)$. The third inequality follows because $\sum_{j \in N_i} x_j^{(\alpha)} \geq 1$, since $x^{(\alpha)}$ satisfies the constraints of LP1. The last inequality follows from a standard expression for e^x.

Since a node is added to Y only in the case that none of its neighbors were added to X, then node v_i is added to Y with probability q_i. Thus,

$$E[Y] = \sum_{i=1}^{n} q_i \leq \sum_{i=1}^{n} \frac{1}{\delta_i^{(1)} + 1}$$

To finish the proof of the theorem, we claim that

$$\sum_{i=1}^{n} \frac{1}{\delta_i^{(1)} + 1} \leq |S^*|$$

This can be seen in several ways, for example by considering LP duality, as in lemma 1 of the KW paper. More directly, one can consider, for every node $v \in S^*$, splittings its cost (of 1) equally among its $\delta_i + 1$ neighbors (including itself), then summing up the cost of S^* this way. We leave the details to the reader. To conclude, we have shown that $E[|S|] = E[X] + E[Y] \leq (1 + \alpha \ln(\Delta + 1))|S^*|$. ■

Theorem 3.1 shows that if we round an α approximate solution to LP1 to an integral solution to MDS, as in algorithm 1, then the resulting dominating set has cost at most $(1 + \alpha \ln(\Delta + 1))$ times the size of the MDS.

3.3 Approximating LP1

We saw in the previous section that if we can find a good approximation for LP1, then we can find a good approximation for MDS. In this section, we describe an algorithm which finds an $O(k\Delta^{2/k})$ approximation of LP1 in $O(k^2)$ rounds. We will describe an algorithm in which each node knows the maximum node degree Δ. A similar, but more complicated algorithm also works when nodes do not know Δ; see the KW paper for details. Before describing the algorithm, we note a peculiar property of the approximation ratio. The function $O(k\Delta^{2/k})$ is convex, and has minimum value $O(\log \Delta)$ for $k = O(\log \Delta)$. It tends to infinity for $k \to 0$ and $k \to \infty$. This seems to suggest that the algorithm performs arbitrarily bad if we run it for an arbitrary number of rounds, which is clearly impossible. In fact, it is possible to show that the approximation ratio of the algorithm approaches $O(\log \Delta)$ as $k \to \infty$. We leave this as an exercise for the reader.

Algorithm 2 LP_{MDS} approximation (Δ known)

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

Figure 2: Algorithm for approximating LP1

The approximate LP algorithm is given in figure 2. It follows a natural greedy idea. Consider our task in solving LP1: we are trying to find small values of x_i which satisfy all the constraints. Each constraint is given by a row of the matrix N. Consider a node v_i with high degree; it is involved in many constraints. By increasing x_i, we can satisfy many rows of N, that is, many constraints, at the same time. Therefore, a greedy method to solve LP1 is to increase the x_i for nodes v_i of high degree. However, we do not want to increase their x values too quickly. Indeed, consider a case where there are multiple nodes of high degree. We may be able to satisfy most of the constraints by just increasing the x value of a few of the nodes. Another way to say this is that we would like to break the symmetry between multiple nodes of high degree, and only select a few of them. We will achieve the effect of symmetry breaking as follows: we increase the x values of high degree nodes slowly. After each increase, each node v_i checks to see if its local constraints have been satisfied; that is, whether $N_i \cdot x \geq 1$. If so, v_i stops increasing x_i; otherwise, it continues.

We now compare this description with algorithm 2. We first define some terminology. At any point during the algorithm, we say a node v_i is *white* if $\sum_{j \in N_i} x_j < 1$. That is, v_i's constraint is not yet satisfied. If v_i's constraint is satisfied, we say it is *gray*. For every node v_i, the algorithm maintains a variable $\widetilde{\delta}(v_i)$, which equals the number of white neighbors of v_i. This is called v_i's *dynamic degree*. Initially, all nodes are white, and so $\widetilde{\delta}(v_i) = \delta_i + 1$ for all v_i. The algorithm consists of an outer and an inner loop. In each iteration of the outer loop, the algorithm increases the x values of nodes whose dynamic degree exceeds a certain threshold. In particular, in round ℓ, where ℓ decreases from $k-1$ to 0, it considers nodes whose dynamic degree exceeds $(\Delta+1)^{\ell/k}$. For such a node v_i, it increases x_i in the inner loop, first slowly, then faster. In particular, in iteration m of the inner loop, where m decreases from $k-1$ to 0, it sets $x_i \leftarrow \max\{x_i, \frac{1}{(\Delta+1)^{m/k}}\}$. Note that $\frac{1}{(\Delta+1)^{m/k}}$ increases as m decreases. After each increase of x_i, v_i send x_i to all its neighbors. Using the received values, it checks if it should now be colored gray; that is, whether $\sum_{j \in N_i} x_j \geq 1$. Then, v_i sends its color to its neighbors, and updates $\widetilde{\delta}(v_i)$ to be the number of white neighbors.

Since there are k iterations of both the outer and inner loops, and each loop involves one local broadcast, the algorithm finishes in $O(k^2)$ rounds. We now show that it gives an $O(k\Delta^{2/k})$ approximation to LP1. We need several lemmas.

Lemma 3.3 At the beginning of each iteration ℓ of the outer loop, we have for any node v_i that $\widetilde{\delta}(v_i) \leq (\Delta+1)^{(\ell+1)/k}$.

Proof. This lemma says each node's dynamic degree is not too large at the beginning of iteration ℓ. Indeed, choose any node v_i, and suppose we are in iteration ℓ of the outer loop. Then in the previous iteration $\ell+1$ of the outer loop, in the final iteration $m=0$ of the inner loop, if $\widetilde{\delta}(v_i) \geq (\Delta+1)^{(\ell+1)/k}$, then we will set $x_i \leftarrow \frac{1}{(\Delta+1)^{m/k}} = 1$. After this, the constraints of all the neighbors of v_i are satisfied. Therefore, all the neighbors of v_i become gray, and the dynamic degree of v_i becomes 0. Thus, at the beginning of iteration ℓ, the dynamic degrees of all nodes are $\leq (\Delta+1)^{(\ell+1)/k}$. ∎

In each iteration of the outer loop, only nodes with $\widetilde{\delta}(v_i) \geq (\Delta+1)^{\ell/k}$ increase x_i. We call such nodes *active nodes*. For any v_i, let $a(v_i)$ be the number of neighbors of v_i which are active. We define $a(v_i) = 0$ if v_i is gray. Note that $a(v_i)$ changes during the execution of the algorithm. We have the following lemma.

Lemma 3.4 At the beginning of each iteration of the inner loop, we have $a(v_i) \leq (\Delta+1)^{(m+1)/k}$.

Proof. This lemma says that the number of active neighbors of v_i decreases during the course of the inner loop. We show that any node v_i with $a(v_i) > (\Delta+1)^{(m+1)/k}$ is gray, and so $a(v_i) = 0$. Indeed, suppose $a(v_i) > (\Delta+1)^{(m+1)/k}$. Let v_j be any active neighbor of v_i in the m'th iteration; then v_j was also active in the previous, $m+1$'st iteration. v_j set $x_j \geq \frac{1}{(\Delta+1)^{(m+1)/k}}$ in the $m+1$'st iteration of the inner loop. So at the beginning of the m'th iteration, we have

$$\begin{aligned} \sum_{(j \in N_i) \wedge (v_j \text{is active})} x_j &\geq a(v_i) \frac{1}{(\Delta+1)^{(m+1)/k}} \\ &\geq (\Delta+1)^{(m+1)/k} / (\Delta+1)^{(m+1)/k} = 1 \end{aligned}$$

That is, after the $m+1$'st iteration of the inner loop, v_i is gray. ■

Next, we bound the total amount of increase of all the x values in one iteration of the outer loop. We associate a variable z_i with each v_i. At the beginning of every iteration of the outer loop, we reset z_i to 0. During an iteration of the outer loop, whenever a node v_i increases x_i, we distribute the increase equally among the z_j's of all neighbors v_j of v_i *which were white* before the increases of x_i. Note that the sum of all the increases in the x values in an iteration of the outer loop is equal to the sum of all the z values at the end of that iteration. We have the following.

Lemma 3.5 At the end of an iteration of the outer loop, we have $z_i \leq \frac{1}{(\Delta+1)^{(\ell-1)/k}}$.

Proof. Note that $\frac{1}{(\Delta+1)^{(\ell-1)/k}}$ increases as ℓ decreases. This lemma says the amount of increase in all the z values in an iteration of the outer loop is not too large. To show this, consider any node v_i which is white at the beginning of the iteration. Note that z_i can only increase if v_i is white, because increases in x_j are only distributed among white neighbors. We divide the outer loop into two phases. The first consists of all iterations of the inner loop where v_i is still white. The second phases consists of the iteration when v_i becomes gray, and all subsequent iterations. We bound the value z_i in the two phases separately.

In the first phase, since v_i is white, we have $\sum_{j\in N_i} x_j < 1$. Any neighbor v_j of v_i which increases x_j must satisfy the condition in line 6 of the algorithm; that is, it must have at least $(\Delta+1)^{\ell/k}$ white neighbors. Therefore, the increase in x_j is distributed among at least $(\Delta+1)^{\ell/k}$ z values. Thus, the increase to z_i during the first phase due to all its neighbors is

$$z_i < \frac{\sum_{j\in N_i} x_j}{(\Delta+1)^{\ell/k}} \leq \frac{1}{(\Delta+1)^{\ell/k}}$$

Suppose v_i becomes gray in iteration m of the inner loop. Then, in the previous iteration, v_i's active neighbors, which are the nodes which contributed to the increase in z_i, increased their x_j values from at least $\frac{1}{(\Delta+1)^{(m+1)/k}}$ to $\frac{1}{(\Delta+1)^{m/k}}$. Again, these increases were distributed among at least $(\Delta+1)^{\ell/k}$ z values. Therefore, the increase to z_i is at most

$$\frac{(\Delta+1)^{-m/k} - (\Delta+1)^{-(m+1)/k}}{(\Delta+1)^{\ell/k}} a(v_i)$$

By lemma 3.4, we have that $a(v_i) \leq (\Delta+1)^{(m+1)/k}$. Plugging this into the above formula and also adding in the increase in z_i from the first phase, we get that $z_i \leq \frac{1}{(\Delta+1)^{(\ell-1)/k}}$. ■

Now, we are finally ready to prove the main theorem.

Theorem 3.6 In $2k^2$ rounds, algorithm 2 produces a $k(\Delta+1)^{2/k}$ approximate solution to LP1.

Proof. It is easy to see that algorithm 2 produces a feasible solution to LP1, because for iteration $\ell = 0, m = 0$, if any node v_i has any white neighbors (including itself), then line 6 of the algorithm is triggered, and v_i sets $x_i \leftarrow 1$.

To see the approximation guarantee, we show that in each of the k iterations of the outer loop, the total increase in all the x values is at most $(\Delta+1)^{2/k}$. Thus, in the k iterations, the sum of all the x values is at most $k(\Delta+1)^{2/k}$.

By lemma 3.3, we know that at the beginning of iteration ℓ of the outer loop, we have $\widetilde{\delta}(v_i) \leq (\Delta+1)^{(\ell+1)/k}$ for every v_i. Thus, since we only increase the z values of white nodes, then there are at most $(\Delta+1)^{(\ell+1)/k}$ nonzero z values during iteration ℓ. Also, from lemma 3.5, we know that for any v_j, $z_j \leq (\Delta+1)^{-(\ell-1)/k}$. Thus, we have

$$\sum_{j \in N_i} z_j \leq \frac{(\Delta+1)^{(\ell+1)/k}}{(\Delta+1)^{(\ell-1)/k}} = (\Delta+1)^{2/k} \tag{2}$$

We claim that equation 2 implies

$$\sum_{i=1}^{n} z_i \leq (\Delta+1)^{2/k}|S^*| \tag{3}$$

where S^* is an MDS. Indeed, we have

$$\begin{aligned} \sum_{i=1}^{n} z_i &\leq \sum_{i \in S^*} \sum_{j \in N_i} z_j \\ &\leq |S^*|(\Delta+1)^{2/k} \end{aligned}$$

Here, the first inequality follows because S^* is a dominating set, and therefore, for any term z_j in the first sum, there exists a v_i such that either $v_i = v_j$ or $v_j \in N_i$. In either case, z_j also occurs as a term in the second sum. The second inequality follows because by equation 2, we have that for any v_i, $\sum_{j \in N_i} z_j \leq (\Delta+1)^{2/k}$.

Thus, we have shown that equation 3 holds. Recall that in any iteration, the increase in the x values is equal to the sum of the z values. Therefore, the increase in x values in iteration ℓ is at most $(\Delta+1)^{2/k}|S^*|$, which proves the theorem. ■

If we set $k = O(\log \Delta)$, then algorithm 2 gives an $O(\log \Delta)$ approximation to LP1, and by theorem 3.1, the expected size of the dominating set produced by algorithm 1 is within $O(\log^2 \Delta)$ times the size of the MDS. Note however that these results are not the best known. The algorithm of Jia, Rajaraman and Suel produces a $O(\log \Delta)$ approximate DS in $O(\log n \log \Delta)$ rounds. Furthermore, the algorithm of Bartal, Byers and Raz, and an improved algorithm by Kuhn and Wattenhofer compute an $1+\epsilon$ approximate solution to any positive LP, including LP1, in polylogarithmic time. However, for small (e.g. constant) values of k, the KW algorithm is the best known.

What Cannot be Computed Locally!

We now give a very brief overview of the Kuhn, Moscibroda and Wattenhofer paper showing that problems like minimum vertex cover (MVC)[1] and minimum dominating set (MDS) cannot be approximated well if we use only a few rounds. In particular, they show that any algorithm which only runs for k rounds cannot approximate MVC and MDS to better than a factor of $\Omega(n^{c/k^2}/k)$ and $\Omega(\Delta^{1/k}/k)$, where n and Δ denote the number of nodes and the largest degree of

[1] The MVC problem consists of finding the smallest set of vertices in a graph such that for every edge in the graph, at least one endpoint of the edge is in the set.

any node, respectively. Some of these lower bounds are tight or nearly tight. For example, Kuhn and Wattenhofer have shown an algorithm which approximates MVC to within polylogarithmic factors in $O(\log \Delta / \log \log \Delta)$ rounds, matching the lower bound.

Note that this overview does not, and is not meant to do justice to this very interesting and important, though perhaps slightly technical paper. We will just give the main ideas behind the result, so that the interested reader might find the actual paper somewhat easier to understand.

The basic idea for the lower bound is as follows. In k rounds, a node can receive messages from nodes which are at most k hops away. Based on these messages, the node must make some decision, e.g. whether to join the MVC. Another way to say this is that the node's decision is completely determined by the topology of its k-hop neighborhood. Therefore, if two nodes both run for k rounds and see the same k-hop topology, they will make the same decision. The KMW lower bound is for the MVC problem (it proves lower bounds for other problems by reduction), and consists of constructing a graph such that two sets of node, one large and one small, both see the same k-hop topology. Therefore, given any algorithm which only runs for k rounds, there is an execution of the algorithm in which both sets of nodes must decide to join the MVC (the sets cannot both not join the MVC because then not all edges would be covered). However, in the minimum VC, it suffices for the smaller set of nodes to join the VC. Therefore, the approximation ratio of the algorithm is at least the size of the large set divided by the set of the small set.

In a bit more detail, to show the lower bound for any particular k, the paper starts by constructing a *cluster tree* of height $k+2$. This tree is constructed in a way that the root of the tree, which we call v_0, and one of its children, v_1, see the same topology for up to k hops. Then, each node in the cluster tree is replaced by a set (cluster) of nodes. In particular, the root of the tree is replaced by δ^{2k} nodes, for some δ, and the number of nodes in the cluster for a node distance d away from the root is δ^{2k-d}. After replacing nodes in the cluster tree by clusters of nodes, we also replace edges in the cluster tree by sets of edges. In particular, if (v, w) is an edge in the cluster tree, and v and w are replaced by clusters $C(v)$ and $C(w)$, then (v, w) is replaced by the edges of a bipartite graph with vertex sets $C(v)$ and $C(w)$. The bipartite graph is regular, meaning that all nodes in each bipartition have the same degree. The degrees need to be carefully chosen. The idea at this point is that the new graph, which we call the *cluster graph*, inherits the properties of the cluster tree. Namely, all the nodes in $C(v_0)$, the cluster replacing v_0, are supposed to have the same k hop view as all the nodes in $C(v_1)$. However, it may be difficult to show this directly. To make the task easier, we perform the so called *Lazebnik-Ustimenko* transformation on the cluster graph to increase its *girth* to $2k+1$. The girth of a graph is the length of the shortest cycle it contains. In a graph of girth $2k+1$, the k-hop neighborhood of every node looks like a tree. This makes proving the equivalence of views between the nodes in $C(v_0)$ and $C(v_1)$ easier (actually, when performing the LU transformation, the clusters $C(v_0)$ and $C(v_1)$ are transformed into other sets of nodes). At this point, the paper goes through a technically detailed proof that the views from $C(v_0)$ and $C(v_1)$ actually are equal. Thus, we conclude that for any k round algorithm, there is some execution in which both the nodes of $C(v_0)$ and $C(v_1)$ are included in the vertex cover. However, it turns out that the minimum vertex cover only needs to include the nodes of $C(v_1)$. Recall that $|C(v_0)| = \delta^{2k}$ while $|C(v_1)| = \delta^{2k-1}$[2], because v_1 is one hop away from v_0 in the cluster tree. Therefore, the vertex cover the algorithm computes is at least δ times larger than necessary. It turns out that δ can be chosen to make the claimed lower bounds work out.

[2] Actually, I fudged some of these numbers to simplify the discussion.