Local computation

Seminar: Algorithms for Wireless Networks

Hendrik Thüs[*]
hendrik.thues@rwth-aachen.de

ABSTRACT

The main aspect of this paper is about computations that can be made locally. This means that a given problem can be solved, independent of the size of the network. In a given network, an algorithm decides the later defined output for a vertex by taking a look at the direct neighborhood.
There are some problems that can be solved locally. For example, the coloring of a graph or a variation of the dining philosophers problem.
But there are also some problems which might be hard to be computed locally. This is for example the minimum vertec cover (MVC). For this problem, some lower bounds are presented. That means that there has to be a certain number of rounds and a certain amount of time to solve the MVC. Also, there are some other problems, like the maximal matching (MM), the maximal independent set (MIS) or the minimum dominating set (MDS). Those problems will be reduced to the MVC, so the lower bounds will nearly be the same.

1. INTRODUCTION

Since distributed systems grow larger, there is a need of algorithms that compute their output independent of the size of the system. Mostly, it is impossible that every vertex in these networks knows the whole system. There is only a small neighborhood that a vertex gets to know. How can a global problem now be solved if there is no global authority which tells each vertex what to do?

So, there is a need of methods to structure a system, where some algorithms can compute an output for a single vertex. The output of all the vertices will be a solution of a global problem. Thus, there is no need of a global authority, if some special problems are concerned.

In this paper, a network is seen as an undirected graph of vertices and edges $G(V, E)$. A vertex represents a processor. An edge is a direct connection of two processors. These two can then communicate by using this edge. One vertex can have several edges, so it can communicate directly to more than one other processor. The number of edges is called the *degree* of a vertex. These degrees are mostly bounded. The vertices are only allowed to communicate with their direct neighbors. After a certain number of rounds, a given algorithm has to decide which output a certain vertex, it operates on, has to receive.

The chapter 2 is about some special problems that can be solved locally. These are the coloring of a graph and the so called *formal dining philosophers problem*. The coloring problem of a given graph is modified a bit. Vertices with a given color are allowed to have neighbors with the same color as long as they have at least one neighbor which is colored differently. This problem is called the *weak coloring problem*. The formal dining philosopher problem is also a modification of the well known *dining philisophers problem* [1]. A philosopher does not only have two possibilities to gain the two needed ressources, but he has at least three possibilities to gain two ressources. The content of this chapter is from the paper 'What Can Be Computed Locally' by Naor and Stockmeyer [5].

In chapter 3, some lower bounds for some special problems are presented. These problems are the computation of a minimum vertex cover and, as reduction to this problem, the minimum dominating set, the maximal matching and the maximal independent set. These lower bounds show that there are some restrictions if these local algorithms are concerned. An algorithm needs a certain amount of time and a certain number of communication rounds to compute the output of a given vertex. The results presented in this chapter are from the paper 'What Cannot Be Computed Locally' by Kuhn, Moscibroda and Wattenhofer [3].

[*]Matrikel-Nr.: 234858

2. POSSIBLE LOCAL COMPUTATION

A possible local computation can be made if an algorithm produces an output for a single vertex independent of the size of the system. That means that this algorithm can only run for a constant time. The output that is produced has to solve a problem concerning the whole system. Thus, there is no need of a global authority that solves a given problem by knowing the whole system. The vertices create a globally valid solution in constant time on their own without knowing everything about the system, they are part of.

2.1 Definitions

The distance between two vertices u and v of a graph $G(V, E)$ is defined as $dist_G(u, v)$. For an edge $e = (v, w)$ with the vertices $v \in G$ and $w \in G$, $dist_G(u, e)$ equals to

$min(dist(u,v), dist(u,w)) + 1$ for any $u \in V(G)$. $B_G(u,r)$ for a vertex u and a positive integer r is a subgraph of G such that $dist_G(u,v) \leq r$ and $dist_G(u,e) \leq r \ \forall v, e \in G$. A pair (H, s) is called a *centered graph* if H is a graph and s is a vertex of H. The maximum distance from s to any other vertex or edge of H is called the *radius* of this centered graph.

For *locally checkable labeling* (LCL), the graph is initially labeled with *input labels* which are elements of the finite set Σ. The *output labels* are elements of the set Γ. The input labels, also called IDs, are unique positive integer values, so every vertex can be identified uniquely. The IDs are used to show the relative order of the vertices. An algorithm $\mathcal{A}$ which runs t steps at a vertex u is supposed to collect all information about the subgraph $B_G(u,t)$. Based on this, $\mathcal{A}$ chooses an output label for u.

Note, that the IDs of the vertices do not have to be strictly incremented by one. That is that not all IDs from $\{1, \ldots, n\}$ have to be assigned to the vertices of an n-vertex graph. Two id numberings η and η' are *order-equivalent* if η' respects the relative order of the IDs of η. A local algorithm $\mathcal{A}$ is *order-invariant* if it produces the same output with the order-equivalent id numberings η and η'.

THEOREM 2.1. *Fix an LCL $\mathcal{L}$, a class $\mathcal{G}$ of graphs and a time bound t. Let d be the maximum degree of a graph in $\mathcal{G}$. There is a number R, depending only on d, t and $\mathcal{L}$, such that the following holds. For every local Algorithm $\mathcal{A}$ with time bound t and every set S of IDs with $|S| \geq R$, there is an order-invariant local algorithm $\mathcal{A}'$ with time bound t such that, for every $G \in \mathcal{G}$ and every input labeling of G, if $\mathcal{A}$ labels G correctly for every id numbering drawn from S then $\mathcal{A}'$ labels G correctly for every id numbering.*

The graphs in this chapter are *d-regular* or they are restricted to the maximum degree of d for any constant d.

2.2 Weak Coloring

A graph is weak *c-colored* with colors from $\{1, \ldots, c\}$ if these colors are assigned to the vertices in a way that every non-isolated $v \in G$ has at least one neighbor $w \in G$ which received a different color. A graph G can be weak-colored locally if all the vertices of G have odd degree. First, $d(d+1)^{d+2}$ colors will be assigned to the vertices of a d-regular graph, $d \geq 3$. If there are less vertices than possible colors, then the IDs can be used as colors. In the second step, the colors will be reduces to two. If there are more vertices than colors, the following algorithm has to be used. The colors are defined as vectors in the following way: $C_v = (C_v[0], C_v[1], \ldots, C_v[d+1])$, $C_v[i] \in \{1, \ldots, d+1\}$. $id(v)$ is defined as the id of a vertex v. The coloring of a vertex v happens in two steps:

1. Get the IDs of all neighbors w of v including $id(v)$, sort them and store as array IDS_v. $r_v(w)$ is now the position of $id(w)$ in IDS_v. $C_v[-1]$ is set to the the degree of v. $C_v[0]$ is set to $r_v(v)$.

2. Set $C_v[r_v(w)] = r_w(v)$ for neighbors w of v.

CLAIM 2.2. *The coloring archived by this algorithm is a legal weak coloring if d is odd.*

PROOF. If all the $r_v(v)$ are different, then we are done because $C_v[0] = r_v(v)$ for all $v \in G$. Otherwise, we assume that v has more neighbors with an ID, higher than $id(v)$, than with a lower ID. $r_w(w) = r_v(v) \Rightarrow r_w(v) < r_v(v)$ follows. So, we can assume that there are at least two vertices a and b where id(v) is at the same position j. So, the jth position of the color of a and b should be the same. But $r_v(a) \neq r_v(b)$ and so, a and b could not have the same color. Thus, v has at least one neighbor with a different color. The second case (more lower than higher IDs) is similar. □

If the vertices have different odd degrees, add $C_v[-1]$ with the degree of v to the colors.

To reduce the colors, there are two methods presented here. The Cole-Vishkin [2] method allows a logarithmical color-reduction to the limit of four. The other method only reduces the color by one in every round.

In the first method, choose the smallest c' with $\binom{c'}{\lfloor c'/2 \rfloor} \geq c$ with c as the number of colors. Now, associate a different subset $S_i \subset \{1, \ldots, c'\}$ of size $\lfloor c'/2 \rfloor$ to every $i \in \{1, \ldots, c\}$. v has at least one neighbor colored differently, it now recolors itself to a color which is element of $S_{\text{color}(v)}$ but not of $S_{\text{color}(w)}$. A weak 4-coloring can be found in $O(log^* d)$ rounds.

The second method is used to reduce the c colors to 2. In the ith round, the vertex with the color i will recolor itself according to the following rules:

1. v recolors itself 0, if all neighbors of v still have their original color.

2. If all recolored neighbors of v have the color 1, then v recolors itself 0. If at least one neighbor of v recolored itself to 0, then v recolors itself 1.

It is easy to see that this algorithm works fine. But there can be some graphs which colors cannot be reduced this way. When a vertex v is not weak-colored properly, then (1) its degree d is even, (2) its rank in its neighborhood is $d/2 + 1$ and (3) every neighbor w of v has degree d and rank $r_w(w) = d/2 + 1$ as well.

THEOREM 2.3. *Let $\mathcal{O}_d$ be the class of graphs of maximum degree d where the degree of every vertex is odd. For every fixed d there is a local algorithm with time bound $O(log^* d)$ which solves the weak 2-coloring problem for $\mathcal{O}_d$.*

2.3 Formal-Dining Philosophers Problem

This problem is a variation of the dining philosophers problem. There are more than only two ressources availlable to a philosopher.A philosopher can only eat if he has obtained any two ressources. For the solution of this problem (no philosopher should die by starvation), the weak-coloring will be used. The vertices of a graph are colored with the colors $\{0, 1, *\}$. If the weak-2-coloring fails at a vertex v, then this vertex will be colored $*$. v and all its neighbors have an even degree. For this problem, the minimum degree of the vertices is assumed as three. Every vertex colored $*$ picks two edges to neighbors and never will release them. Since these vertices can eat when they want to, they will be ignored in the further description. Let's assume that every vertex colored $c \in \{0, 1\}$ has at least one neighbor colored 1 and one colored 0. This assumption will later be corrected, because this is not always the case. Each vertex colored $c \in \{0, 1\}$ chooses a neighbor colored c as its *first neighbor* and an other neighbor colored $1 - c$ as its *second neighbor*. The vertices now act upon a dynamic algorithm to tell them, which edge to request and so on. Requesting means that a vertex tries to allocate this edge.

1. Request edge from the first neighbor
2. Request edge from the second neighbor
3. Eat
4. Release edges

Claim 2.4. *The maximum length, a vertex has to wait to get the two edges is two.*

Proof. The worst case is that a vertex is waiting at step 1. Then the other vertex, which is sharing this edge, has already allocated this edge. And since this vertex has the same color, it is at least at step two, which means that it has to wait one single step. □

We cannot assume that each vertex has two neighbors with different colors. Every vertex colored 1 chooses a neighbor colored 0 as the second neighbor. This decision is distributed to this neighbor. A vertex w colored 0 now takes on of the vertices that have chosen w as second neighbor. If there is none, then it just take an arbitrary 1-colored neighbor. Each vertex chooses an arbitrary adjacent vertex as the first neighbor, but not the second neighbor and not a neighbor colored with a $*$ if this vertex permanently blocks this shared edge.

Claim 2.5. *The maximum length, a vertex has to wait to get the two edges is four if the description above is used.*

Proof. Consider a setting, where three vertices w_1, w_2 and w_3 are colored 1, 0 and 1, respectively. Let w_2 be the first neighbor of w_3, w_2 the second neighbor of w_1 and w_3 the second neighbor of w_2. This can not happen, because w_1 has chosen w_2 as a second neighbor, w_3 has not chosen w_2 as a second neighbor. The vertex w_2 has to take a vertex as second neighbor that has chose w_2. So, this vertex cannot take w_3. Consider now a chain of vertices u_0 to u_5 where u_0 is colored c, u_1 is colored $1 - c$ and so on. Let's assume that u_i is waiting for u_{i+1}, $0 \leq 1 \leq 4$. This is a waiting-chain with a length of 5. For $1 \leq i \leq 3$, u_i has to be the first neighbor of u_{i+1} and for $1 \leq i \leq 4$, u_{i+1} has to be the second neighbor of u_i. Now, no matter, if $c = 0$ or $c = 1$, we get a forbidden configuration as mentioned above. So, there cannot be a waiting-chain with the length of 5. □

Theorem 2.6. *The protocol presented here solves the formal dining philosophers problem locally.*

3. IMPOSSIBLE LOCAL COMPUTATION

In this chapter, some lower bounds for the minimum vertex cover are presented. A vertex v collects all the information it can get in k communication rounds. It is shown that the runtime of any algorithm does not only depend on the number of the communication rounds k but also on the number of vertices or on the highest degree in the system. For this, a cluster-tree with some special properties is created. In subchapter 3.3, the lower bounds are derived.

3.1 Definitions

$\mathcal{T}_{v,k}$ is the environment, a vertex v gets to know in k rounds. A given algorithm now decides with the labeling of $\mathcal{T}_{v,k}$ and $\mathcal{T}_{v,k}$ which output is calculated for v. $G_k = (V, E)$ for each positive integer k, is a graph that can be grouped into disjoint sets of vertices. These sets are linked to each other as bipartite graphs, they are called *clusters*. $CT_k = (\mathcal{C}, \mathcal{A})$ is called *cluster tree*. It is a directed tree with doubly labeled arcs $l : \mathcal{A} \to \mathbb{N} \times \mathbb{N}$. An arc is defined as follows: $a = (C, D) \in \mathcal{A}$ with $l(a) = (\delta_C, \delta_D)$. This means that the clusters C and D are linked as a bipartite graphs and each vertex $v \in C$ has δ_C neighbors in D and the other way around. The clusters that do not have any child-clusters are called *leaf-clusters*. The others are called *inner-clusters*.

Definition 3.1. *CT_k is recursively defined as follows:*

$$\begin{array}{rcl} CT_1 & := & (\mathcal{C}_1, \mathcal{A}_1) \\ \mathcal{C}_1 & := & \{C_0, C_1, C_2, C_3\} \\ \mathcal{A}_1 & := & \{(C_0, C_1), (C_0, C_2), (C_1, C_3)\} \\ l(C_0, C_1) & := & (\delta_0, \delta_1) \\ l(C_0, C_2) & := & (\delta_1, \delta_2) \\ l(C_1, C_3) & := & (\delta_0, \delta_1) \end{array}$$

Given CT_{k-1}, we obtain CT_k in two steps:

- *For each inner-cluster C_i, add a new leaf-cluster C'_i with $l(C_i, C'_i) := (\delta_k, \delta_{k+1}$.*
- *For each leaf-cluster C_i of CT_{k-1} with $(C'_i, C_i) \in \mathcal{A}$ and $l(C'_i, C_i) = (\delta_p, \delta_{p+1}$, add k new leaf-clusters C'_j with $l(C_i, C'_j) = (\delta_j, \delta_{j+1}$ for $j = 0, \cdots, k, j \neq p + 1$.*

The *level* of a cluster of a cluster tree is the distance between this cluster and C_0. The *depth* of a cluster C is its distance to the furthest leaf-cluster in the subtree rooted at C. $g(G)$, the *girth* of G, is the length of the shortest cycle in G. G_k has to have a girth of at least $2k+1$. So it is guaranteed that all the vertices in G_k see a tree in k communication rounds.

3.2 Construction

For appropriate r, i.e. $r = 2k - 4$, $r \geq 3$, r is odd and a prime-power q, let $D(r, q)$ be a graph with $2q^r$ vertices and a girth of at least $2k + 1$. The construction of such a graph $D(r, q)$ can be found in [4]. Let G'_k be an arbitrary cluster tree. With this G'_k as basis, a graph G_k is built with a girth that is large enough. Both graphs are bipartit. The two sets $V_1(G'_k)$ and $V_2(G'_k)$ represent the odd and the even levels of the tree, respectively. Let m be the number of vertices of the larger set. And let q be the smallest prime power, which is greater or equal to m. The vertices v of G'_k are labeled with elements $c(v) \in \mathbb{F}_q$. G_k is a subgraph of $D(r, q)$. Two vertices (p) and (l) of $D(r, q)$ (which are arrays) are now connected in G_k, if they are connected in $D(r, q)$ and if there is an edge between $u \in V_1(G'_k)$ and $v \in V_2(G'_k)$ so that $c(u) = p_1$ and $c(v) = l_1$. At the end, all the edges of G_k are deleted that have not been relevant. Then, G_k has at most $2mq^{2k-5}$ vertices.

Now, it will be proven that two adjacent vertices in the clusters C_0 and C_1 see the same topology $\mathcal{T}_{v,k}$. Each vertex only sees a tree in k communication rounds, these trees are called *view trees*, which can be derived by recursivly following all the neighbors of a vertex v.

Definition 3.2. *Let $V_1 = \bigcup_{i=0\cdots k} b_i$ and $V_2 = \bigcup_{i=0\cdots k} b'_i$ be view-trees; b_i and b'_i are subtrees entered on δ_i. Then, V_1*

and V_2 *are* r-equal *if all corresponding subtrees are (r-1)-equal,*

$$V_1 \overset{r}{=} V_2 \Leftarrow b_i \overset{r-1}{=} b_i', \forall i \in \{0, \cdots, k\}.$$

Further, all subtrees are 0-equal. It does not matter on which level two cluster are, which depth they have or on which δ*-link they can be reached. If the algorithm is in the last communication round, these are the last clusters. And so, they can be seen as equal.*

To show that a given vertex in C_0 sees exactly the same topology as a given vertex in C_1, it has to be proven that $V_{C_0} \overset{k}{=} V_{C_1}$ holds in G_k.

Lemma 3.3. *Two view-trees* V_{v_1} *and* V_{v_2} *which can be entered via an equal* δ_i *and which have the same depth are* r*-equal if both of them have a subtree* β *and* β' *which are* $(r-1)$*-equal. This also holds if* V_{v_1} *and* V_{v_2} *are in different levels.*

$$V_{v_1} \overset{r}{=} V_{v_2} \Leftarrow \beta \overset{r-1}{=} \beta'$$

Proof. Due to the construction of the cluster tree, subtrees with equal depth and entry-link grow identically. So, all paths, that do not return to the startingclusters, are identical. The only difference is that other paths which may return to the startingclusters could not use the link δ_{i+1}. They have to use $\delta_{i+1} - 1$. But this does not affect β and β' and therefore,

$$V_{v_1} \overset{r}{=} V_{v_2} \Leftarrow V_{v_1}' \overset{r-s}{=} V_{v_2}' \wedge \beta \overset{r-1}{=} \beta', s > 1$$

This can be repeated until $r = s$ and due to $V_{v_1}' \overset{0}{=} V_{v_2}'$, the lemma follows. □

Lemma 3.4. *Let* β *and* β' *be sets of subtrees of the view-trees* V_{v_1} *and* V_{v_2}*, respectively.* V_{v_1} *and* V_{v_2} *are entered via an equal* δ_i*.* V_{v_1} *has depth* d *and is on level* x *and* V_{v_2} *has depth* d' *and is on level* y*. Then, for all* x *and* y *and for all* $r \leq min(d, d')$*,*

$$V_{v_1} \overset{r}{=} V_{v_2} \Leftarrow \beta \overset{r-1}{=} \beta'$$

Proof. It is easy to see that the depth of the subtrees is not important anymore. Since the entrylinks are the same, the subtrees have grown identically (eccept from the depth). For $r \leq min(d, d')$, the given subtrees are the same. And so, this is proven by using lemma 3.3. □

Theorem 3.5. *Conider a graph* G_k*. Let* V_{C_0} *and* V_{C_1} *be view-trees of two adjacent vertices in the clusters* C_0 *and* C_1*, respectively. Then,* $V_{C_0} \overset{k}{=} V_{C_1}$*.*

It is easy to see that this can be proven via induction. Interested readers are refered to [3].

3.3 Lower Bounds and Reductions

In this subsection, the lower bounds of a k-local MVC are derived. Let OPT be the optimal and ALG an arbitrary solution to this problem. OPT will try to cover the vertices of C_0 by putting all the vertices of C_1 in the vertex cover. ALG might try to put quite many vertices of C_0 to the vertex cover.

Lemma 3.6. *Let ALG run for at most k rounds. When applied to* G_k*, ALG contains in the worst case at least half of the vertices of* C_0*.*

Proof. Since the view trees of arbitrary adjacent vertices v of C_0 and w of C_1 are the same, the cannot be distinguished by an algorithm. So, they have the same probability P to end in the vertex cover. But one of them has to, due to the definition of MCV. So, $P(v) + P(w) \geq 1 \Leftrightarrow P(v) = P(w) \geq 1/2$. It follows that each vertex of C_0 has the probability of $1/2$ to end in the vertex cover. The lemma follows. □

We know that the vertices of C_0 are not compulsory in the vertex cover. Thus, there is an upper bound: $|OPT| \leq n - n_0$. Now, δ_i is definded as follows for some value δ:

$$\delta_i := \delta^i, \forall i \in \{0, \cdots, k+1\} \tag{1}$$

Lemma 3.7. *If* $k + 1 < \delta$*, the number of vertices* n *of* G_k *is*

$$n \leq n_0 \left(1 + \frac{k+1}{\delta - (k+1)}\right)$$

Proof. Due to (1), we can assume that a cluster on level l contains n_0/δ^l vertices. Each cluster has at most $k+1$ upper neighbor-clusters. Thus, the number of vertices n_l on level l is upper bounded by

$$n_l \leq (k+1)^l \cdot \frac{n_0}{\delta^l}.$$

Summing up the vertices over all the level l:

$$n_0 \leq n_0 \cdot \sum_{l=0}^{k+1} \left(\frac{k+1}{\delta}\right)^l \leq n_0 \cdot \sum_{l=0}^{\infty} \left(\frac{k+1}{\delta}\right)^l$$

$$= n_0 \left(1 + \frac{k+1}{\delta - (k+1)}\right)$$

□

The number of vertices per cluster decrease by a factor δ on each level of CT_k. Due to (1), it is now possible to obtain G_k'. CT_k has $k+2$ levels, and at most δ^k vertices are in each cluster. If we assume that the clusters on level $k+1$ have δ^k vertices, then there are δ^{2k+1-l} vertices in a given cluster on level l. By lemma 3.7, it follows that $n \leq 2n_0$ for $k+1 \leq \delta/2$. By using the construction of subsection 3.2, we get $n_0 \leq n_0' \cdot \langle n' \rangle^{2k-5}$ with $\langle n' \rangle$ as the smallest primepower equal to n', that is $\langle n' \rangle < 4n_0'$. It follows:

$$n_0 \leq (4n_0')^{2k-4} \leq 4^{2k-4}\delta^{4k^2} \tag{2}$$

Theorem 3.8. *There are graphs G, such that in k communication rounds, every distributed algorithm for the minimum vertex cover problem on G has an approximation ratio of at least*

$$\Omega\left(\frac{n^{c/k^2}}{k}\right) \text{ and } \Omega\left(\frac{\Delta^{1/k}}{k}\right)$$

for some sonstant $c \geq 1/4$*, where* n *and* Δ *denote the number of vertices and the highest degree in G, respectively.*

Proof. We can choose $\delta \geq 4^{-1/(2k)} n_0^{1/(4k^2)}$ due to inequality (2). Finally, using lemmas 3.6 and 3.7, the approximation ratio α is at least

$$\alpha \geq \frac{n_0/2}{n - n_0} \geq \frac{n_0/2 \cdot \delta/2}{n_0 \cdot (k+1)} \geq \frac{(n/2)^{1/(4k^2)}}{4^{1+1/(2k)}(k+1)} \in \Omega\left(\frac{n^{1/(4k^2)}}{k}\right)$$

The second lower bound follows from $\Delta = \delta^{k+1}$. □

THEOREM 3.9. *In order to obtain a polylogarithmic or even constant approximation ratio, every distributed algorithm for the MVC problem requires at least* $\Omega\left(\sqrt{\frac{\log n}{\log\log n}}\right)$ *and* $\Omega\left(\frac{\log\Delta}{\log\log\Delta}\right)$ *communication rounds.*

PROOF. For $k = \beta\sqrt{\log n/\log\log n}$ with $\beta > 0$, we get

$$\alpha \geq \gamma n^{\frac{c\log\log n}{\beta^2\log n}} \cdot \frac{1}{\beta}\sqrt{\frac{\log\log n}{\log n}}$$

as the first lower bound of theorem 3.8 with γ as the hidden constant of the Ω-notation. After a few calculation steps, we get that α is element of $\Omega\left(\log(n)^{\left(\frac{c}{\beta^2}-\frac{1}{2}\right)}\right)$ There is always a β such that the obove expression is a polylogarithmic term $\alpha(n)$. The first lower bound of the theorem 3.8 follows. The second lower bound of this theorem follows analogous with $k = \beta\log\Delta/\log\log\Delta$. □

Now, other problems are reduces to the MVC problem. For the maximal matching (MM) and the maximal independent set (MIS), the best known lower bound for distributed computation is $\Omega(\log^* n)$.

THEOREM 3.10. *There are graphs G on which every distributed algorithm requires time*

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

to compute a MM or a MIS.

PROOF. Since the set of all end-vertices of the edges of a MM form a 2-approximation for MVC, the lower bound follows directly from theorem 3.9. To see that these lower bounds hold for MIS, consider a line graph $L(G_k)$ of G_k. The MM problem on G is the same as the MIS problem on $L(G)$. And even though there is a different amount of vertices (n in G_k and less than $n^2/2$ in $L(G_k)$) and a different maximum degree (Δ in G_k and 2Δ in $L(G_k)$, the theorem holds. □

THEOREM 3.11. *There are graphs G, such that in k communication rounds, every distributed algorithm for the minimum dominating set (MDS) problem on G has approximation ratios at least*

$$\Omega\left(\frac{n^{c/k^2}}{k}\right) \text{ and } \Omega\left(\frac{\Delta^{1/k}}{k}\right)$$

for some constant c, where n denotes the number of vertices and Δ the highest degree in G.

PROOF. Let $G' = (V', E')$ be a graph in which the MVC problem is solved. The graph $G = (V, E)$ for the MDS is now constructed as follows: Every $v' \in V'(G')$ corresponds to a $v_n \in V(G)$, every $e' \in E'(G')$ corresponds to a $v_e \in V(G)$. Two vertices $v'_n, u'_n \in G$ are adjacent, if $v', u' \in G'$ are adjacent. Two vertices $v_n, w_e \in G'$ are adjacent, if the vertex $v' \in V'(G')$ is adjacent to the edge $w' \in E'(G')$. There are no other edges in G. If C is a vertex cover for G', the vertices $v_n \in G$, which correspond to vertices in C form a valid dominating set on G. All e-vertices are covered. The n-vertices are covered due to the fact that a valid vertex cover is a valid dominating set as well. The other direction: If D is a valid dominating set on G, every e-vertex in D can be replaced by one of its two neighbors. And thus, we get a different dominating set D' which also has the needed properties. But D' only consists of n-vertices. D' dominates all e-vertices and it also forms a valid vertex cover for G'. It follows that MVS on G' is as hard as MDS on G. □

4. OPEN QUESTIONS

If the weak-2-coloring is applicable, it can be made very efficient and with a low failure rate. Based on the weak-2-coloring, the formal dining philosophers problem can also be solved efficiently. But it is very unusual that an algorithm can be ashured that the degrees of the vertices of a graph are bounded.

For some other problems, e.g. the MVC, there are lower bounds for the runtime depending on the highest degree of a graph and on the number of vertices. They cannot be solved locally, because the vertices do not know the whole system. Due to fast growing networks, it would be good, if some problems could be solved locally. But there are only a few special locally solvable problems.

5. REFERENCES

[1] Dining philosophers problem. `http://en.wikipedia.org/wiki/Dining_philosophers_problem`.

[2] R. Cole and U. Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Inf. Control*, 70(1):32–53, 1986.

[3] F. Kuhn, T. Moscibroda, and R. Wattenhofer. What cannot be computed locally! In *PODC '04: Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 300–309, New York, NY, USA, 2004. ACM.

[4] F. Lazebnik and V. A. Ustimenko. Explicit construction of graphs with an arbitrary large girth and of large size. *Discrete Appl. Math.*, 60(1-3):275–284, 1995.

[5] M. Naor and L. Stockmeyer. What can be computed locally? pages 184–193, 1993.