Local Computation: Lower and Upper Bounds[1]

Fabian Kuhn[1], Thomas Moscibroda[2], Roger Wattenhofer[3]

[1]fkuhn@csail.mit.edu, MIT, USA

[2]moscitho@microsoft.com, Microsoft Research, Redmond, USA

[3]wattenhofer@tik.ee.ethz.ch, ETH Zurich, Switzerland

The question of what can be computed, and how efficiently, are at the core of computer science. Not surprisingly, in distributed systems and networking research, an equally fundamental question is what can be computed in a *distributed* fashion. More precisely, if nodes of a network must base their decision on information in their local neighborhood only, how well can they compute or approximate a global (optimization) problem? In this paper we give the first substantial lower bound on such local computation for (optimization) problems including minimum vertex cover, minimum (connected) dominating set, maximum matching, maximal independent set, and maximal matching. In addition we present a new distributed algorithm for solving general covering and packing linear programs. For some problems this algorithm is tight with the lower bounds, for others it is a distributed approximation scheme. Together, our lower and upper bounds establish the local computability and approximability of a large class of problems, characterizing how much local information is required to solve these tasks.

1. INTRODUCTION

Many of the most fascinating systems in the world are large and complex networks, such as the human society, the Internet, or the brain. Such systems have in common that they are composed of a multiplicity of individual entities, so-called *nodes*; human beings in society, hosts in the Internet, or neurons in the brain. Each individual node can directly communicate only to a small number of neighboring nodes. For instance, most human communication is between acquaintances or within the family, and neurons are directly linked with merely a relatively small number of other neurons. On the other hand, in spite of each node being inherently "near-sighted," i.e., restricted to *local* communication, the entirety of the system is supposed to work towards some kind of *global* goal, solution, or equilibrium.

In this work we investigate the possibilities and limitations of *local computation*, i.e., to what degree local information is sufficient to solve global tasks. Many

---

[1]This paper is based in part on work that has appeared in the following two preliminary versions: *What Cannot Be Computed Locally*, In *Proceedings of the 23rd ACM Symposium on the Principles of Distributed Computing (PODC)*, St. John's, Canada, 2004 [Kuhn et al. 2004] and *The Price of Being Near-Sighted*, In *Proceedings of the 17th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Miami, Florida, 2006 [Kuhn et al. 2006]. Notice that the key lower bound results presented in this journal version are improved over the ones presented in the conference versions, thus rendering many of our upper bounds tight.

tasks can be solved entirely locally, for instance, how many friends of friends one has. Clearly, only local communication is required to answer this question. Many other tasks are inherently global, for instance, counting the total number of nodes or determining the diameter of the system. To solve such global problems, some information must traverse across the entire graph distances.

It is natural to ask whether there are natural tasks that are in the middle of these two extremes; tasks that are neither completely local nor inherently global. In this paper we answer this question affirmatively. Assume for example that the nodes want to organize themselves, some nodes should be masters, the others will be slaves. The rules are that no two masters shall be direct neighbors, but every slave must have at least one master as direct neighbor. In graph theory, this problem is known as the *maximal independent set* (MIS) problem. Intuitively, this problem seems local since the rules are completely local. Consequently it might be expected that every node can communicate with its neighbors a few times, and together they can decide who will become master and who will become slave. However, as we show in this paper, this intuition is misleading. Even though the problem seems local, it cannot be solved using local information only! No matter how the system tackles the problem, no matter what protocol or algorithm the nodes use, non-local information is vital to solve the task. On the other hand, the problem is also not global: Mid-range information is enough to solve the problem. As such the MIS problem establishes an example that is neither local nor global, but in-between these extremes. Since at first sight it looks local, we call it *pseudo-local*. Using *locality-preserving reductions* we are able to show that there exists a whole class of pseudo-local problems.

We show that this class of pseudo-local problems also includes many combinatorial optimization problems, such as minimum vertex cover, minimum dominating set, or maximum matching. In such problems, each node must base its decision (for example whether or not to join the dominating set) only on information about its local neighborhood, and yet, the goal is to collectively achieve a good approximation to the globally optimal solution. Studying such *local approximation algorithms* is particularly interesting because it sheds light on the trade-off between the amount of available local information and the resulting global optimality. Specifically, it characterizes the amount of information needed in distributed decision making: what can be done with the information that is available within some fixed-size neighborhood of a node. Positive and negative results for local algorithms can thus be interpreted as information-theoretic upper and lower bounds; they give insight into the value of information.

We believe that studying the fundamental possibilities and limitations of local computation is of interest to theoreticians in approximation theory, distributed computing, and graph theory. Furthermore, our results may be of interest for a wide range of scientific areas, for instance dynamic systems that change over time. Our theory shows that small changes in a dynamic system may cause an intermediate (or pseudo-local) "butterfly effect," and it gives non-trivial bounds for self-healing or self-organizing systems, such as self-assembling robots. It also establishes bounds for further application areas, initially in engineering and computing, possibly extending to other areas studying large-scale systems, e.g. social science,

finance, neural networks, or ant colonies.

1.1   Model and Notation

Local Computations: We consider a distributed system in which distributed decision makers at the nodes of a graph must base their computations and decisions on the knowledge about their local neighborhoods in the graph. Formally, we are given a graph $G = (V, E)$, $|V| = n$, and a parameter k (k might depend on n or some other property of G). At each node $v \in V$ there is an independent agent (for simplicity, we identify the agent at node v with v as well). Every node $v \in V$ has a unique identifier $id(v)$[2] and possibly some additional input. We assume that each node $v \in V$ can learn the complete neighborhood $\Gamma_k(v)$ up to distance k in G (see below for a formal definition of $\Gamma_k(v)$). Based on this information, all nodes need to make independent computations and need to individually decide on their outputs without communicating with each other. Hence, the output of each node $v \in V$ can be computed as a function of it's k-neighborhood $\Gamma_k(v)$.

Synchronous Message Passing Model: The described graph-theoretic local computation model is equivalent to the classic *message passing* model of distributed computing. In this model, the distributed system is modeled as a point-to-point communication network, described by an undirected graph $G = (V, E)$, in which each vertex $v \in V$ represents a node (host, device, processor, . . .) of the network, and an edge $(u, v) \in E$ is a bidirectional communication channel that connects the two nodes. Initially, nodes have no knowledge about the network graph; they only know their own identifier and potential additional inputs. All nodes wake up simultaneously and computation proceeds in synchronous *rounds*. In each round, every node can send one, arbitrarily long message to each of its neighbors. Since we consider point-to-point networks, a node may send different messages to different neighbors in the same round. Additionally, every node is allowed to perform local computations based on information obtained in messages of previous rounds. Communication is reliable, i.e., every message that is sent during a communication round is correctly received by the end of the round. An algorithm's *time complexity* is defined as the number of communication rounds until all nodes terminate.[3]

The above is a standard model of distributed computing and is generally known as the LOCAL model [Peleg 2000; Linial 1992]. It is the strongest possible model when studying the impact of locally-restricted knowledge on computability, because it focuses entirely on the locality of distributed problems and abstracts away other issues arising in the design of distributed algorithms (e.g. need for small messages, fast local computations, congestion, asynchrony, packet loss, etc.). It is thus the most fundamental model for proving lower bounds on local computation [Linial 1992]; because any lower bound is a true consequence of locality restrictions.

Equivalence of Time Complexity and Neighborhood-Information: There is a one-to-one correspondence between the *time complexity of distributed algo-*

---

[2]All our results hold for any possible ID space including the standard case where IDs are the numbers $1, \ldots, n$.

[3]Notice that this synchronous message passing model captures many practical systems, including for example, Google's Pregel system, a practically implemented computational model suitable for computing problems in large graphs [Malewicz et al. 2010].

rithms in the LOCAL model and the graph theoretic notion of *neighborhood-information*. In particular, a distributed algorithm with time-complexity k (i.e., in which each node performs k communication rounds) is equivalent to a scenario in which distributed decision makers at the nodes of a graph must base their decision on (complete) knowledge about their k-hop neighborhood $\Gamma_k(v)$ only. This is true because with unlimited sized messages, every node $v \in V$ can easily collect all IDs and interconnections of all nodes in its k-hop neighborhood in k communication rounds. On the other hand, a node v clearly cannot obtain any information from a node at distance $k+1$ or further away, because this information would require more than k rounds to reach v. Thus, the LOCAL model relates distributed computation to the *algorithmic theory of the value of information* as studied for example in [Papadimitriou and Yannakakis 1993]: the question of *how much local knowledge* is required for distributed decision makers to solve a global task or approximation a global goal is equivalent to the question of *how many communication rounds* are required by a distributed algorithm to solve the task.

Notation: For nodes $u, v \in V$ and a graph $G = (V, E)$, we denote the shortest-path distance between u and v by $d_G(u, v)$. Let $\Gamma_k(v)$ be the k-hop neighborhood of a node $v \in V$. Formally, we define $\Gamma_k(v) := \{u \in V : d_G(u, v) \le k\}$. We also use the shortcut $\Gamma_v := \Gamma_1(v)$, that is, Γ_v is the (inclusive) neighborhood of v. In a local computation with k-hop neighborhood information (or equivalently, in any distributed algorithm with time complexity k), each node has a *partial view* of the graph and must base its algorithm's outcome solely on information obtained in $\Gamma_k(v)$. Formally, let $\mathcal{T}_{v,k}$ be the topology seen by v after k rounds in a distributed algorithm, i.e. $\mathcal{T}_{v,k}$ is the graph induced by the k-neighborhood of v where edges between nodes at exactly distance k are excluded. The *labeling* (i.e. the assignment of identifiers to nodes) of $\mathcal{T}_{v,k}$ is denoted by $\mathcal{L}(\mathcal{T}_{v,k})$. The *view* of a node v is the pair $\mathcal{V}_{v,k} := (\mathcal{T}_{v,k}, \mathcal{L}(\mathcal{T}_{v,k}))$. Any deterministic distributed algorithm can be regarded as a function mapping $(\mathcal{T}_{v,k}, \mathcal{L}(\mathcal{T}_{v,k}))$ to the possible outputs. For randomized algorithms, the outcome of v is also dependent on the randomness computed by the nodes in $\mathcal{T}_{v,k}$.

1.2   Problem Definitions

In this paper, we study several standard combinatorial optimization problems (and their natural relaxations) that intuitively appear to be local, yet turn out to be neither completely local nor global. Specifically, we consider the following standard optimization problems in graphs:

—**Minimum Vertex Cover (MVC):** Given a graph $G = (V, E)$, find a minimum vertex subset $S \subseteq V$, such that for each edge in E, at least one of its endpoints is in S.

—**Minimum Dominating Set (MDS):** Given a graph $G = (V, E)$, find a minimum vertex subset $S \subseteq V$, such that for each node $v \in V$, either $v \in S$ or at least one neighbor of v must be in S.

—**Minimum Connected Dominsting Set (MCDS):** Given a graph $G = (V, E)$, find a minimum dominating set $S \subseteq V$, such that the graph $G[S]$ induced by S is connected.

—**Maximum Matching (MaxM):** Given a graph $G = (V, E)$, find a maximum edge subset $T \subseteq E$, such that no two edges in T are adjacent.

In all these cases, we consider the respective problem on the network graph, i.e., on the graph representing the network. In addition to the above mentioned problems, we study their natural linear programming relaxations as well as a slightly more general class of linear programs (LP) in a distributed context. Consider an LP and its corresponding dual LP in the following canonical forms:

$$
\begin{array}{llll}
\min & \underline{c}^{\mathrm{T}}\underline{x} & \min & \underline{b}^{\mathrm{T}}\underline{y} \\
\text{s. t.} & A \cdot \underline{x} \geq \underline{b} \quad \text{(P)} & \text{s. t.} & A^{\mathrm{T}} \cdot \underline{y} \leq \underline{c} \quad \text{(D)} \\
& \underline{x} \geq \underline{0}. & & \underline{y} \geq \underline{0}.
\end{array}
$$

We call an LP in form (P) to be in primal canonical form (or just in canonical form) and an LP in form (D) to be in dual canonical form. If all the coefficients of $\underline{b}$, $\underline{c}$, and A are non-negative, primal and dual LPs in canonical forms are called *covering* and *packing* LPs, respectively. The relaxations of vertex cover and dominating set are covering LPs, whereas the relaxation of matching is a packing LP.

While there is an obvious way to interpret graph problems such as vertex cover, dominating set, or matching as a distributed problem, general LPs have no immediate distributed meaning. We use a natural mapping of an LP to a network graph, which was introduced in [Papadimitriou and Yannakakis 1993] and applied in [Bartal et al. 1997]. For each primal variable x_i and for each dual variable y_j, there are nodes v_i^p and v_j^d, respectively. We denote the set of primal variables by V_p and the set of dual variables by V_d. The network graph $G_{\mathrm{LP}} = (V_p \dot\cup V_d, E)$ is a bipartite graph with the edge set

$$
E := \left\{ (v_i^p, v_j^d) \in V_p \times V_d \mid a_{ji} \neq 0 \right\},
$$

where a_{ji} is the entry of row j and column i of A. We define $n_p := |V_p|$ and $n_d := |V_d|$, that is, A is a $(n_d \times n_p)$-matrix. Further, the maximum primal and dual degrees are denoted by Δ_p and Δ_d, respectively. In most real-world examples of distributed LPs and their corresponding combinatorial optimization problems, the network graph is closely related to the graph G_{LP} such that any computation on G_{LP} can efficiently be simulated in the actual network.

In the context of local computation, each node $v \in V$ has to independently decide whether it joins a vertex cover or dominating set, which of its incident edges should participate in a matching, or what variable its corresponding variable gets assigned when solving an LP. Based on local knowledge, the nodes thus seek to produce a feasible *approximation* to the global optimization problem. Depending on the number of rounds nodes communicate—and thus on the amount of local knowledge available at the nodes—, the quality of the solution that can be computed differs. We seek to understand the trade-off between the amount of local knowledge (or communication between nodes) and the resulting approximation to the global problem.

In addition to these optimization problems, we also consider important binary problems, including:

—**Maximal Independent Set (MIS):** Given a graph $G = (V, E)$, select an

inclusion-maximal vertex subset $S \subseteq V$, such that no two nodes in S are neighbors.

—**Maximal Matching (MM):** Given a $G = (V, E)$, select an inclusion-maximal edge subset $T \subseteq E$, such that no two edges in T are adjacent.

For such problems, we are interested in the question, how much local information is required such that distributed decision makers are able to compute fundamental graph-theoretic structures, such as a MIS or a MM. Whereas the described combinatorial optimization problems are NP-hard and thus, unless P = NP, even with global knowledge, algorithms can compute only approximations to the optimum, a MIS or a MM can trivially be computed with global knowledge. The question is thus how much *local* knowledge to solve these tasks.

1.3 Contributions

Our main results are a lower bound on the distributed approximability of the minimum vertex cover problem in Section 3 as well as a generic algorithm for covering and packing LPs of the form (P) and (D) in Section 5, respectively. Both results are accompanied by various extensions and adaptations to the other problems introduced in Section 1.2. It follows from our discussion that these results imply strong lower and upper bounds on the amount of local information required to solve/approximate global tasks.

For the MVC lower bound, we show that for every $k > 0$, there exists a graph G such that every k-round distributed algorithm for the MVC problem has approximation ratios at least

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

for positive constants c and c', where n and Δ denote the number of nodes and the highest degree of G, respectively. Choosing k appropriately, this implies that to achieve a polylogarithmic (or even a constant) approximation ratio, every MVC algorithm requires at least $\Omega(\sqrt{\log n})$ and $\Omega(\log \Delta)$ rounds, respectively.[4] All bounds also hold for randomized algorithms. Using reductions that preserve the locality properties of the considered graph, we show that the same lower bounds also hold for the distributed approximation of the minimum dominating set and maximum matching problem. Because MVC and MaxM are covering and packing problems with constant integrality gap, the same lower bounds are also true for general distributed covering and packing LPs of the form (P) and (D). Furthermore, using locality-preserving reductions, we also derive lower bounds on the amount of local information required at each node to collectively compute important structures such as a MIS or a maximal matching in the network graph. Finally, a simple girth argument can be used to show that for the connected dominating set problem, even stronger lower bounds are true. We show that in k rounds, no algorithm can have an approximation ratio that is better than $n^{c/k}$ for some positive constant c. This implies that for a polylogarithmic approximation ratio, $\Omega(\log n)$ rounds are needed.

---

[4]Notice that all these bounds are improved compared to the results reported in the conference version of this paper [Kuhn et al. 2004] by factors of $\Theta(\sqrt{\log \log n})$ and $\Theta(\log \log \Delta)$, respectively. This improvement renders some of our upper bound algorithms in Section 5 tight.

We show that the above lower bound results that depend on Δ are asymptotically tight for the MVC and MaxM problem by giving an algorithm that obtains $O(\Delta^{c/k})$ approximations with k hops of information for a positive constant c. That is, a constant approximation to MVC can be computed with every node having $O(\log \Delta)$-hop information. Our main upper bound result is a distributed algorithm to solve general covering and packing LPs of the form (P) and (D). We show that with k hops of information, again for some positive constant c, a $n^{c/k}$-approximation can be computed. As a consequence, by choosing k large enough, we also get a distributed approximation scheme for this class of problems. For $\varepsilon > 0$, the algorithm allows to compute an $(1 + \varepsilon)$-approximation in $O(\log(n)/\varepsilon)$ rounds of communication. Using a distributed randomized rounding scheme, good solutions to fractional covering and packing problems can be converted into good integer solutions in many cases. In particular, we obtain the currently best distributed dominating set algorithm, which achieves a $(1 + \varepsilon) \ln \Delta$-approximation for MDS in $O(\log(n)/\varepsilon)$ rounds for $\varepsilon > 0$. Finally, we extend the MDS result to connected dominating sets and show that up to constant factors in approximation ratio and time complexity, we can achieve the same time-approximation trade-off as for the MDS problem also for the CDS problem.

2. RELATED WORK

Local Computation: Local algorithms have first been studied in the Mid-1980s [Luby 1986; Cole and Vishkin 1986]. The basic motivation was the question whether one can build efficient network algorithms, where each node only knows about its immediate neighborhood. However, even today, relatively little is known about the fundamental limitations of local computability. Similarly, little is known about *local approximability*, i.e., how well combinatorial optimization problems can be approximated if each node has to decide individually based only on knowledge available in its neighborhood.

Linial's seminal $\Omega(\log^* n)$ time lower bound for constructing a maximal independent set on a ring [Linial 1992] is virtually the only non-trivial lower bound for local computation.[5] Linial's lower bound shows that the non-uniform $O(\log^* n)$ coloring algorithm by Cole and Vishkin [Cole and Vishkin 1986] is asymptotically optimal for the ring. It has recently been extended to other problems [Czygrinow et al. 2008; Lenzen and Wattenhofer 2008]. On the other hand, it was later shown that there exist non-trivial problems that can indeed be computed *strictly locally*. Specifically, Naor and Stockmeyer present locally checkable labelings which can be computed in constant time, i.e., with purely local information [Naor and Stockmeyer 1995].

There has also been significant work on (parallel) algorithms for approximating packing and covering problems that are faster than interior-point methods that can be applied to general LPs (e.g. [Fleischer 2000; Plotkin et al. 1995; Young 2001]). However, these algorithms are not local as they need at least some global informa-

---

[5]There are of course numerous lower bounds and impossibility results in distributed computing [Fich and Ruppert 2003], but they apply to computational models where locality is not the key issue. Instead, the restrictive factors are usually aspects such as bounded message size [Elkin 2004a], asynchrony, or faulty processors.

tion to work.[6] The problem of approximating positive LPs using only local information has been introduced in [Papadimitriou and Yannakakis 1991; 1993]. The first algorithm achieving a constant approximation for general covering and packing problems in polylogarithmic time is described in [Bartal et al. 1997]. Distributed (approximation) algorithms targeted for specific covering and packing problems include algorithms for the minimum dominating set problem [Dubhashi et al. 2003; Jia et al. 2001; Rajagopalan and Vazirani 1998; Kuhn and Wattenhofer 2003] as well as algorithms for maximal matchings and maximal independent sets [Alon et al. 1986; Israeli and Itai 1986; Luby 1986]. We also refer to the survey in [Elkin 2004b].

While local computation was always considered an interesting and elegant research question, it is currently experiencing an Indian summer because of several new application domains, such as overlay or sensor networks. Partly driven by these new application domains, and partly due to the lower bounds presented in this paper, research in the last five years has concentrated on restricted graph topologies, such as unit disk graphs, or bounded-growth graphs. A survey covering this more recent work is [Suomela 2009].

Self-Organization & Fault-Tolerance: Looking at the wider picture, one may argue that local algorithms even go back to the early 1970s when Dijkstra introduced the concept of *self-stabilization* [Dijkstra 1973; 1974]. A self-stabilizing system must survive arbitrary failures, including for instance a total wipe out of volatile memory at all nodes. The system must self-heal and eventually converge to a correct state from any arbitrary starting state, provided that no further faults occur.

It seems that the world of self-stabilization (which is asynchronous, long-lived, and full of malicious failures) has nothing in common with the world of local algorithms (which is synchronous, one-shot, and free of failures). However, as shown 20 years ago, this perception is incorrect [Awerbuch and Sipser 1988; Afek et al. 1990; Awerbuch and Varghese 1991]; indeed it can easily be shown that the two areas are related. Intuitively, this is because (i) asynchronous systems can be made synchronous, (ii) self-stabilization concentrates on the case after the last failure, when all parts of the system are correct again, and (iii) one-shot algorithms can just be executed in an infinite loop. Thus, efficient self-stabilization essentially boils down to local algorithms and hence, local algorithms are the key to understanding fault-tolerance [Lenzen et al. 2009].

Likewise, local algorithms help to understand *dynamic networks*, in which the topology of the system is constantly changing, either because of churn (nodes constantly joining or leaving as in peer-to-peer systems), mobility (edge changes because of mobile nodes in mobile networks), changing environmental conditions (edge changes in wireless networks), or algorithmic dynamics (edge changes because of algorithmic decisions in overlay networks). In dynamic networks, no node in the network is capable of keeping up-to-date global information on the network. Instead, nodes have to perform their intended (global) task based on local information only. In other words, all computation in these systems is inherently local!

---

[6]In general, a local algorithm provides an efficient algorithm in the PRAM model of parallel computing, but a PRAM algorithm is not necessarily local [Wattenhofer and Wattenhofer 2004].

By using local algorithms, it is guaranteed that dynamics only affect a restricted neighborhood. Indeed, to the best of our knowledge, local algorithms yield the best solutions when it comes to dynamics. Dynamics also play a natural role in the area of self-assembly (DNA computing, self-assembling robots, shape-shifting systems, or claytronics), and as such it is not surprising that local algorithms are being considered a key to understanding self-assembling systems [Sterling 2009; Goldstein et al. 2005].

Other Applications: Local computation has also been considered in a non-distributed (sequential) context. One example are *sublinear time algorithms*, i.e. algorithms that cannot read the entire input, but must give (estimative) answers based on samples only. For example, the local algorithms given in Section 5 are used by Parnas and Ron [Parnas and Ron 2007] to design a sublinear- or even constant-time sequential approximation algorithms. In some sense the local algorithm plays the role of an oracle that will be queried by random sampling, see also [Nguyen and Onak 2008].

There has recently been significant interest in the database community about the Pregel system [Malewicz et al. 2010], a practically implemented computational model suitable for computing problems in large graphs. All our lower bounds directly apply to Pregel, i.e., they show how many iterations are required to solve certain tasks; while our upper bounds provide optimal or near-optimal algorithms in a Pregel-like message-passing system.

Finally, the term "local(ity)" is used in various different contexts in computer science. The most common use may be *locality of reference* in software engineering. The basic idea is that data and variables that are frequently accessed together should also be physically stored together in order to facilitate techniques such as caching and pre-fetching. At first glance, our definition of locality does not seem to be related at all with locality in software engineering. However, such a conclusion may be premature. One may for instance consider a multi-core system where different threads operate on different parts of data, and sometimes share data. Two threads should never manipulate the same data at the same time, as this may cause inconsistencies. At runtime, threads may figure out whether they have conflicts with other threads, however, there is no "global picture". One may model such a multi-thread system with a virtual graph, with threads being nodes, and two threads having a conflict by an edge between the two nodes. Again, local algorithms (in particular maximal independent set or vertex coloring) might help to efficiently schedule threads in a non-conflicting way. At this stage, this is mostly a theoretical vision [Schneider and Wattenhofer 2009], but with the rapid growth of multi-core systems, it may get practical sooner than expected.

3. LOCAL COMPUTATION: LOWER BOUND

The proofs of our lower bounds are based on the *timeless indistinguishability* argument [Fischer et al. 1985; Lamport et al. 1982]. In k rounds of communication, a network node can only gather information about nodes which are at most k hops away and hence, only this information can be used to determine the computation's outcome. If we can show that within their k-hop neighborhood many nodes see exactly the same graph topology; informally speaking, all these nodes are equally

qualified to join the MIS, dominating set, or vertex cover. The challenge is now to construct the graph in such a way that selecting the wrong subset of these nodes is ruinous.

We first construct a hard graph for the MVC problem because i) it has a particularly simple combinatorial structure, and ii) it appears to be an ideal candidate for local computation. Intuitively, a node should be able to decide whether or not to join the vertex cover using information from its local neighborhood only; very distant nodes appear to be superfluous for its decision. Our proof shows that this intuition is misleading and even such a seemingly simple problem such as MVC is not purely local; it cannot be approximated well in a constant number of communication rounds. Our *hardness of distributed approximation* lower bounds for MVC holds even for *randomized algorithms* as well as for the *fractional* version of MVC. We extend the result to other problems in Section 4.

Proof Outline: The basic idea is to construct a graph $G_k = (V, E)$, for each positive integer k. In G_k, there are many neighboring nodes that see exactly the same topology in their k-hop neighborhood, that is, no distributed algorithm with running time at most k can distinguish between these nodes. Informally speaking, both neighbors are equally qualified to join the vertex cover. However, choosing the wrong neighbors in G_k will be ruinous.

G_k contains a bipartite subgraph S with node set $C_0 \cup C_1$ and edges in $C_0 \times C_1$ as shown in Figure 1. Set C_0 consists of n_0 nodes each of which has δ_0 neighbors in C_1. Each of the $n_0 \cdot \frac{\delta_0}{\delta_1}$ nodes in C_1 has δ_1, $\delta_1 > \delta_0$, neighbors in C_0. The goal is to construct G_k in such a way that all nodes in $v \in S$ see the same topology $\mathcal{T}_{v,k}$ within distance k. In a globally optimal solution, all edges of S may be covered by nodes in C_1 and hence, no node in C_0 needs to join the vertex cover. In a local algorithm, however, the decision of whether or not a node joins the vertex cover depends only on its local view, that is, the pair $(\mathcal{T}_{v,k}, \mathcal{L}(\mathcal{T}_{v,k}))$. We show that because adjacent nodes in S see the same $\mathcal{T}_{v,k}$, every algorithm adds a large portion of nodes in C_0 to its vertex cover in order to end up with a feasible solution. In other words, we construct a graph in which the symmetry between two adjacent nodes cannot be broken within k communication rounds. This yields suboptimal local decisions and hence, a suboptimal approximation ratio. Throughout the proof, C_0 and C_1 denote the two sets of the bipartite subgraph S.

The proof is organized as follows. The structure of G_k is defined in Subsection 3.1. In Subsection 3.2, we show how G_k can be constructed without small cycles, ensuring that each node sees a tree within distance k. Subsection 3.3 proves that adjacent nodes in C_0 and C_1 have the same view $\mathcal{T}_{v,k}$ and finally, Subsection 3.4 derives the local approximability lower bounds.

3.1 The Cluster Tree

The nodes of graph $G_k = (V, E)$ can be grouped into disjoint sets which are linked to each other as bipartite graphs. We call these disjoint sets of nodes *clusters*. The structure of G_k is defined using a directed tree $CT_k = (\mathcal{C}, \mathcal{A})$ with doubly labeled arcs $\ell : \mathcal{A} \to \mathbb{N} \times \mathbb{N}$. We refer to CT_k as the *cluster tree*, because each vertex $C \in \mathcal{C}$ represents a cluster of nodes in G_k. The *size* of a cluster $|C|$ is the number of nodes the cluster contains. An arc $a = (C, D) \in \mathcal{A}$ with $\ell(a) = (\delta_C, \delta_D)$ denotes that the clusters C and D are linked as a bipartite graph, such that each node $u \in C$ has
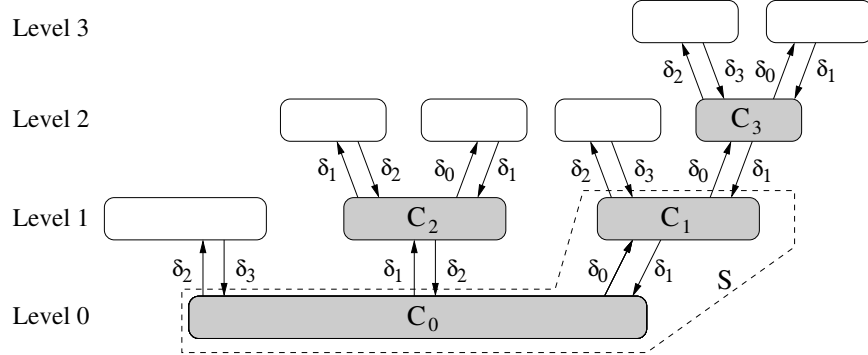
Fig. 1. Cluster-Tree CT_2.

δ_C neighbors in D and each node $v \in D$ has δ_D neighbors in C. It follows that $|C| \cdot \delta_C = |D| \cdot \delta_D$. We call a cluster *leaf-cluster* if it is adjacent to only one other cluster, and we call it *inner-cluster* otherwise.

DEFINITION 1. *The cluster tree CT_k is recursively defined as follows:*

$$CT_1 := (\mathcal{C}_1, \mathcal{A}_1), \quad \mathcal{C}_1 := \{C_0, C_1, C_2, C_3\}$$
$$\mathcal{A}_1 := \{(C_0, C_1), (C_0, C_2), (C_1, C_3)\}$$
$$\ell(C_0, C_1) := (\delta_0, \delta_1), \quad \ell(C_0, C_2) := (\delta_1, \delta_2),$$
$$\ell(C_1, C_3) := (\delta_0, \delta_1)$$

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 $\ell(C_i, C_i') := (\delta_k, \delta_{k+1})$.*

—*For each leaf-cluster C_i of CT_{k-1} with $(C_{i'}, C_i) \in \mathcal{A}$ and $\ell(C_{i'}, C_i) = (\delta_p, \delta_{p+1})$, add $k-1$ new leaf-clusters C_j' with $\ell(C_i, C_j') := (\delta_j, \delta_{j+1})$ for $j = 0 \ldots k, j \neq p+1$.*

Further, we define $|C_0| = n_0$ for all CT_k.

Figure 1 shows CT_2. The shaded subgraph corresponds to CT_1. The labels of each arc $a \in \mathcal{A}$ are of the form $\ell(a) = (\delta_l, \delta_{l+1})$ for some $l \in \{0, \ldots, k\}$. Further, setting $|C_0| = n_0$ uniquely determines the size of all other clusters. In order to simplify the upcoming study of the cluster tree, we need two additional definitions. The *level* of a cluster is the distance to C_0 in the cluster tree (cf. Figure 1). The *depth* of a cluster C is its distance to the furthest leaf in the subtree rooted at C. Hence, the depth of a cluster plus one equals the height of the subtree corresponding to C. In the example of Figure 1, the depths of C_0, C_1, C_2, and C_3 are 3, 2, 1, and 1, respectively.

Note that CT_k describes the general structure of G_k, i.e. it defines for each node the number of neighbors in each cluster. However, CT_k does not specify the actual adjacencies. In the next subsection, we show that G_k can be constructed so that each node's local view is a tree.

3.2   The Lower-Bound Graph

In Subsection 3.3, we will prove that the topologies seen by nodes in C_0 and C_1 are identical. This task is greatly simplified if each node's topology is a tree (rather than a general graph) because we do not have to worry about cycles. The *girth* of a graph G, denoted by $g(G)$, is the length of the shortest cycle in G. We want to construct G_k with girth at least $2k + 1$ so that in k communication rounds, all nodes see a tree. Given the structural complexity of G_k for large k, constructing G_k with large girth is not a trivial task. The solution we present is based on the construction of the graph family $D(r, q)$ as proposed in [Lazebnik and Ustimenko 1995]. For given r and q, $D(r, q)$ defines a bipartite graph with $2q^r$ nodes and girth $g(D(r, q)) \geq r + 5$. In particular, we show that for appropriate r and q, we obtain an instance of G_k by deleting some of the edges of $D(r, q)$. In the following, we introduce $D(r, q)$ up to the level of detail which is necessary to understand our results.

For an integer $r \geq 1$ and a prime power q, $D(r, q)$ defines a bipartite graph with node set $P \cup L$ and edges $E_D \subset P \times L$. The nodes of P and L are labeled by the r-vectors over the finite field $\mathbb{F}_q$, i.e. $P = L = \mathbb{F}_q^r$. In accordance with [Lazebnik and Ustimenko 1995], we denote a vector $p \in P$ by (p) and a vector $l \in L$ by $[l]$. The components of (p) and $[l]$ are written as follows (for $D(r, q)$, the vectors are projected onto the first r coordinates):

$$(p) = (p_1, p_{1,1}, p_{1,2}, p_{2,1}, p_{2,2}, p'_{2,2}, p_{2,3}, p_{3,2}, \ldots, p_{i,i}, p'_{i,i}, p_{i,i+1}, p_{i+1,i}, \ldots) \quad (1)$$

$$[l] = [l_1, l_{1,1}, l_{1,2}, l_{2,1}, l_{2,2}, l'_{2,2}, l_{2,3}, l_{3,2}, \ldots l_{i,i}, l'_{i,i}, l_{i,i+1}, l_{i+1,i}, \ldots]. \quad (2)$$

Note that the somewhat confusing numbering of the components of (p) and $[l]$ is chosen in order to simplify the following system of equations. There is an edge between two nodes (p) and $[l]$, exactly if the first $r - 1$ of the following conditions hold (for $i = 2, 3, \ldots$).

$$
\begin{aligned}
l_{1,1} - p_{1,1} &= l_1 p_1 \\
l_{1,2} - p_{1,2} &= l_{1,1} p_1 \\
l_{2,1} - p_{2,1} &= l_1 p_{1,1} \\
l_{i,i} - p_{i,i} &= l_1 p_{i-1,i} \\
l'_{i,i} - p'_{i,i} &= l_{i,i-1} p_1 \\
l_{i,i+1} - p_{i,i+1} &= l_{i,i} p_1 \\
l_{i+1,i} - p_{i+1,i} &= l_1 p'_{i,i}
\end{aligned}
\quad (3)
$$

In [Lazebnik and Ustimenko 1995], it is shown that for odd $r \geq 3$, $D(r, q)$ has girth at least $r + 5$. Further, if a node u and a coordinate of a neighbor v are fixed, the remaining coordinates of v are uniquely determined. This is concretized in the next lemma.

LEMMA 1. *For all $(p) \in P$ and $l_1 \in \mathbb{F}_q$, there is exactly one $[l] \in L$ such that l_1 is the first coordinate of $[l]$ and such that (p) and $[l]$ are connected by an edge in $D(r, q)$. Analogously, if $[l] \in L$ and $p_1 \in \mathbb{F}_q$ are fixed, the neighbor (p) of $[l]$ is uniquely determined.*

PROOF. The first $r - 1$ equations of (3) define a linear system for the unknown coordinates of $[l]$. If the equations and variables are written in the given order,

the matrix corresponding to the resulting linear system of equations is a lower triangular matrix with non-zero elements in the diagonal. Hence, the matrix has full rank and by the basic laws of (finite) fields, the solution is unique. Exactly the same argumentation holds for the second claim of the lemma. □

We are now ready to construct G_k with large girth. We start with an arbitrary instance G_k' of the cluster tree which may have the minimum possible girth 4. An elaboration of the construction of G_k' is deferred to Subsection 3.4. For now, we simply assume that G_k' exists. Both G_k and G_k' are bipartite graphs with odd-level clusters in one set and even-level clusters in the other. Let m be the number of nodes in the larger of the two partitions of G_k'. We choose q to be the smallest prime power greater than or equal to m. In both partitions $V_1(G_k')$ and $V_2(G_k')$ of G_k', we uniquely label all nodes v with elements $c(v) \in \mathbb{F}_q$.

As already mentioned, G_k is constructed as a subgraph of $D(r, q)$ for appropriate r and q. We choose q as described above and we set $r = 2k-4$ such that $g(D(r,q)) \geq 2k + 1$. Let $(p) = (p_1, \dots)$ and $[l] = [l_1, \dots]$ be two nodes of $D(r,q)$. (p) and $[l]$ are connected by an edge in G_k if and only if they are connected in $D(r,q)$ and there is an edge between nodes $u \in V_1(G_k')$ and $v \in V_2(G_k')$ for which $c(u) = p_1$ and $c(v) = l_1$. Finally, nodes without incident edges are removed from G_k.

LEMMA 2. *The graph G_k constructed as described above is a cluster tree with the same degrees δ_i as in G_k'. G_k has at most $2mq^{2k-5}$ nodes and girth at least $2k + 1$.*

PROOF. The girth directly follows from the construction; removing edges cannot create cycles.

For the degrees between clusters, consider two neighboring clusters $C_i' \subset V_1(G_k')$ and $C_j' \subset V_2(G_k')$ in G_k'. In G_k, each node is replaced by q^{2k-5} new nodes. The clusters C_i and C_j consist of all nodes (p) and $[l]$ which have their first coordinates equal to the labels of the nodes in C_i' and C_j', respectively. Let each node in C_i' have δ_α neighbors in C_j', and let each node in C_j' have δ_β neighbors in C_i'. By Lemma 1, nodes in C_i have δ_α neighbors in C_j and nodes in C_j have δ_β neighbors in C_i, too. □

Remark. In [Lazebnik et al. 1995], it has been shown that $D(r,q)$ is disconnected and consists of at least $q^{\lfloor \frac{r+2}{4} \rfloor}$ isomorphic components which the authors call $CD(r,q)$. Clearly, those components are valid cluster trees as well and we could use one of them for the analysis. As the asymptotic results remain unaffected by this observation, we continue to use G_k as constructed above.

3.3 Equality of Views

In this subsection, we prove that two adjacent nodes in clusters C_0 and C_1 have the same *view*, i.e. within distance k, they see exactly the same topology $\mathcal{T}_{v,k}$. Consider a node $v \in G_k$. Given that v's view is a tree, we can derive its *view-tree* by recursively following all neighbors of v. The proof is largely based on the observation that corresponding subtrees occur in both node's view-tree.

Let C_i and C_j be adjacent clusters in CT_k connected by $\ell(C_i, C_j) = (\delta_l, \delta_{l+1})$, i.e. each node in C_i has δ_l neighbors in C_j, and each node in C_j has δ_{l+1} neighbors in C_i. When traversing a node's view-tree, we say that we *enter* cluster C_j (resp., C_i) over *link* δ_l (resp., δ_{l+1}) from cluster C_i (resp., C_j).

Let T_u be the tree topology seen by some node u of degree d and let $T_1, \ldots, T_d$ be the topologies of the subtrees of the d neighbors of u. We use the following notation to describe the topology T_u based on $T_1, \ldots, T_d$:

$$T_u := [T_1 \uplus T_2 \uplus \ldots \uplus T_d] = \left[\biguplus_{i=1}^{d} T_i \right].$$

Further, we define the following abbreviation:

$$d \cdot T := \underbrace{T \uplus T \uplus \ldots \uplus T}_{d \text{ times}}.$$

If $\mathcal{T}_1, \ldots, \mathcal{T}_d$ are sets of trees, $[\mathcal{T}_1 \uplus \ldots \uplus \mathcal{T}_d]$ denotes the set of all view-trees $[T_1, \uplus \ldots \uplus T_d]$ for which $T_i \in \mathcal{T}_i$ for all i. We will also mix single trees and set of trees, e.g., $[\mathcal{T}_1 \uplus T]$ is the set of all view-trees $[T_1 \uplus T]$ with $T_1 \in \mathcal{T}_1$.

DEFINITION 2. *The following nomenclature refers to subtrees in the view-tree of a node in G_k.*

—M_i *is the subtree seen upon entering cluster C_0 over a link δ_i.*

—$\mathcal{B}_{i,d,\lambda}^{\uparrow}$ *denotes the set of subtrees that are seen upon entering a cluster $C \in \mathcal{C} \backslash \{C_0\}$ on level λ over a link δ_i from level $\lambda - 1$, where C has depth d.*

—$\mathcal{B}_{i,d,\lambda}^{\downarrow}$ *denotes the set of subtrees that are seen upon entering a cluster $C \in \mathcal{C} \backslash \{C_0\}$ on level λ over a link δ_i from level $\lambda + 1$, where C has depth d.*

In the following, we will frequently abuse notation and write $\mathcal{B}_{i,d,\lambda}$ when we mean some tree in $\mathcal{B}_{i,d,\lambda}$. The following example should clarify the various definitions. Additionally, you may refer to the example of G_3 in Figure 7.2.

EXAMPLE 1. *Consider G_1. Let V_{C_0} and V_{C_1} denote the view-trees of nodes in C_0 and C_1, respectively:*

$$
\begin{aligned}
V_{C_0} &\in \left[\delta_0 \cdot \mathcal{B}_{0,1,1}^{\uparrow} \uplus \delta_1 \cdot \mathcal{B}_{1,0,1}^{\uparrow}\right] & V_{C_1} &\in \left[\delta_0 \cdot \mathcal{B}_{0,0,2}^{\uparrow} \uplus \delta_1 \cdot M_1\right] \\
\mathcal{B}_{0,1,1}^{\uparrow} &\subseteq \left[\delta_0 \cdot \mathcal{B}_{0,0,2}^{\uparrow} \uplus (\delta_1 - 1) \cdot M_1\right] & \mathcal{B}_{0,0,2}^{\uparrow} &\subseteq \left[(\delta_1 - 1) \cdot \mathcal{B}_{1,1,1}^{\downarrow}\right] \\
\mathcal{B}_{1,0,1}^{\uparrow} &\subseteq \left[(\delta_2 - 1) \cdot M_2\right] & M_1 &\in \left[(\delta_0 - 1) \cdot \mathcal{B}_{0,1,1}^{\uparrow} \uplus \delta_1 \cdot \mathcal{B}_{1,0,1}^{\uparrow}\right] \\
M_2 &\in \left[\delta_0 \cdot \mathcal{B}_{0,1,1}^{\uparrow} \uplus (\delta_1 - 1) \cdot \mathcal{B}_{1,0,1}^{\uparrow}\right] & &\ldots
\end{aligned}
$$

We start the proof by giving a set of rules which describe the subtrees seen at a given point in the view-tree. We call these rules *derivation rules* because they allow us to *derive* the view-tree of a node by mechanically applying the matching rule for a given subtree.
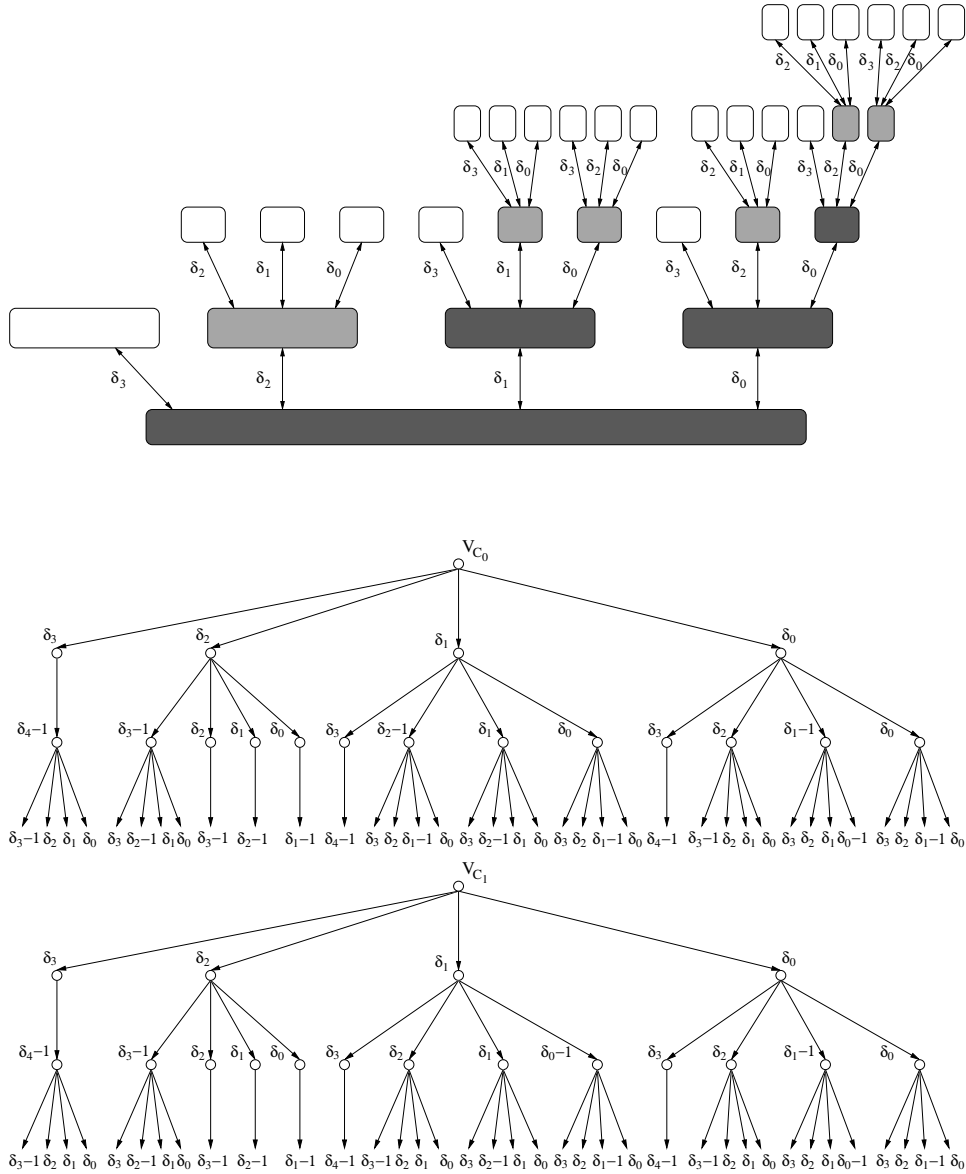
Fig. 2. The Cluster Tree CT_3 and the corresponding view-trees of nodes in C_0 and C_1. The cluster trees CT_1 and CT_2 are shaded dark and light, respectively. The labels of the arcs of the cluster tree represent the number of higher-level cluster. The labels of the reverse links are omitted. In the view-trees, an arc labeled with δ_i stands for δ_i edges, all connecting to identical subtrees.

LEMMA 3. *The following derivation rules hold in G_k:*

$$M_i \in \left[(\delta_{i-1} - 1) \cdot \mathcal{B}_{i-1,k-i+1,1}^{\uparrow} \uplus \biguplus_{\substack{j}} \delta_j \cdot \mathcal{B}_{j,k-j,1}^{\uparrow} \right]$$

$$\mathcal{B}_{i,d,1}^{\uparrow} \subseteq \left[\mathcal{F}_{\{i+1\},d,1} \uplus \mathcal{D}_{d,1} \uplus (\delta_{i+1} - 1) \cdot M_{i+1} \right]$$

$$\mathcal{B}_{i,k-i,1}^{\downarrow} \subseteq \left[\mathcal{F}_{\{i-1,i+1\},k-i,1} \uplus \mathcal{D}_{k-i,1} \uplus \delta_{i+1} \cdot M_{i+1} \uplus (\delta_{i-1} - 1) \cdot \mathcal{B}_{i-1,k-i-1,2}^{\uparrow} \right]$$

$$\mathcal{B}_{i-2,k-i,2}^{\uparrow} \subseteq \left[\mathcal{F}_{\{i-1\},k-i,2} \uplus \mathcal{D}_{k-i,2} \uplus (\delta_{i-1} - 1) \cdot \mathcal{B}_{i-1,k-i+1,1}^{\downarrow} \right] \quad (i \geq 2),$$

where $\mathcal{F}$ and $\mathcal{D}$ are defined as

$$\mathcal{F}_{W,d,\lambda} := \biguplus_{j \in \{0,\ldots,k-d+1\} \setminus W} \delta_j \cdot \mathcal{B}_{j,d-1,\lambda+1}^{\uparrow}$$

$$\mathcal{D}_{d,\lambda} := \biguplus_{j=k-d+2}^{k} \delta_j \cdot \mathcal{B}_{j,k-j,\lambda+1}^{\uparrow}.$$

PROOF. We first show the derivation rule for M_i. By Definition 2, M_i is the subtree seen upon entering the cluster C_0 over a link δ_i. Let us therefore first derive a rule for the view-tree V_{C_0} of C_0 in G_k. We show by induction on k that $V_{C_0} \in \left[\biguplus_{j \in \{0,\ldots,k\}} \delta_j \cdot \mathcal{B}_{j,k-j,1}^{\uparrow} \right]$. It can be seen in Example 1 that the rule holds for $k = 1$. For the induction step, we build CT_{k+1} from CT_k as defined in Definition 1. C_0 is an inner cluster and therefore, one new cluster with view trees of the form $\mathcal{B}_{k+1,0,1}^{\uparrow}$ is added. The depth of all other subtrees increases by 1 and thus the rule for V_{C_0} follows. If we enter C_0 over link δ_i, there will be only $\delta_{i-1} - 1$ edges left to return to the cluster from which we had entered C_0. Consequently, M_i is the same as V_{C_0} but with only $\delta_{i-1} - 1$ subtrees of the form $\mathcal{B}_{i-1,k-i+1,1}^{\uparrow}$.

The remaining rules follow along similar lines. Let C_i be a cluster with entry-link δ_i which was first created in CT_r, $r < k$. Note that in CT_k, the depth of C_i is $d = k-r$ because each subtree increases its depth by one in each "round". According to the second building rule of Definition 1, r new neighboring clusters (subtrees) are created in CT_{r+1}. More precisely, a new cluster is created for all entry-links $\delta_0 \ldots \delta_r$, except δ_i. We call these subtrees *fixed-depth* subtrees F. If the subtree with root C_i has depth d in CT_k, the fixed-depth subtrees have depth $d-1$. In each $CT_{r'}$, $r' \in \{r + 2, \ldots, k\}$, C_i is an inner-cluster and hence, one new neighboring cluster with entry-link $\delta_{r'}$ is created. We call these subtrees *diminishing-depth* subtrees D. In CT_k, each of these subtrees has grown to depth $k - r'$.

We now turn our attention to the differences between the three rules. They stem from the exceptional treatment of level 1, as well as the predicates $\uparrow$ and $\downarrow$. In $\mathcal{B}_{i,d,1}^{\uparrow}$, the link δ_{i+1} returns to C_0, but contains only $\delta_{i+1} - 1$ edges in the view-tree.

In $\mathcal{B}_{i,k-i,1}^{\downarrow}$, we have to consider two special cases. The first one is the link to C_0. For a cluster on level 1 with depth d and entry-link (from C_0) δ_j, the equality $k = d + j$ holds and therefore, the link to C_0 is δ_{i+1} and thus, M_{i+1} follows. Secondly, because in $\mathcal{B}_{i,k-i,1}^{\downarrow}$ we come from a cluster C' on level 2 over a δ_i link, there are only $\delta_{i-1} - 1$ links back to C' and therefore there are only $\delta_{i-1} - 1$ subtrees of the form $\mathcal{B}_{i-1,k-i-1,2}^{\uparrow}$. (Note that since we entered the current cluster from a higher level, the link leading back to where we came from is δ_{i-1}, instead of δ_{i+1}). The depths of the subtrees in $\mathcal{B}_{i,k-i,1}^{\downarrow}$ follow from the above observation that the view $\mathcal{B}_{i,k-i,1}^{\downarrow}$ corresponds to the subtree of a node in the cluster on level 1 that is entered on link δ_{i+1} from C_0.

Finally in $\mathcal{B}_{i-2,k-i,2}^{\uparrow}$, we again have to treat the returning link δ_{i-1} to the cluster on level 1 specially. Consider a view-tree $\mathcal{B}_{j,d,x}^{\uparrow}$ of depth d. All level $x + 1$ subtrees that are reached by links $\delta_{j'}$ with $j' \le k - d + 1$ are fixed-depth subtrees of depth $d - 1$. Because $\mathcal{B}_{i-2,k-i,2}^{\uparrow}$ is reached through link δ_{i-1} from the cluster on level 1 and because $i - 1 \le k - (k - i) = i$, $\mathcal{B}_{i-2,k-i,2}^{\uparrow}$ is a fixed-depth subtree of its level 1

parent. Thus, we get that the depth of the $\delta_{i-1}-1$ subtrees $\mathcal{B}^{\downarrow}_{i-1,k-i+1,1}$ is $k-i+1$. $\qquad \square$

Note that we do not give general derivation rules for all $\mathcal{B}^{\uparrow}_{i,d,x}$ and $\mathcal{B}^{\downarrow}_{i,d,x}$ because they are not needed in the following proofs. Next, we define the notion of r-equality. Intuitively, if two view-trees are r-equal, they have the same topology within distance r.

DEFINITION 3. *Let* $\mathcal{V} \subseteq \left[\biguplus_{i=1}^{d} \mathcal{T}_i \right]$ *and* $\mathcal{V}' \subseteq \left[\biguplus_{i=1}^{d} \mathcal{T}'_i \right]$ *be sets of view-trees. Then,* $\mathcal{V}$ *and* $\mathcal{V}'$ *are* r-equal *if there is a permutation* π *on* $\{1,\dots,d\}$ *such that* $\mathcal{T}_i$ *and* $\mathcal{T}'_{\pi(i)}$ *are* $(r-1)$-equal for all $i \in \{1,\dots,d\}$:*

$$\mathcal{V} \overset{r}{=} \mathcal{V}' \iff \mathcal{T}_i \overset{r-1}{=} \mathcal{T}'_{\pi(i)}, \; \forall i \in \{1,\dots,d\}.$$

Further, all (sets of) subtrees are 0-equal, *i.e.,* $\mathcal{T} \overset{0}{=} \mathcal{T}'$ *for all* $\mathcal{T}$ *and* $\mathcal{T}'$.

Using the notion of r-equality, we can now define what we actually have to prove. We will show that in G_k, $V_{C_0} \overset{k}{=} V_{C_1}$ holds. Since the girth of G_k is at least $2k+1$, this is equivalent to showing that each node in C_0 sees exactly the same topology within distance k as its neighbor in C_1. We first establish several helper lemmas. For two collections of sub-tree sets $\beta = \mathcal{T}_1 \uplus \dots \uplus \mathcal{T}_d$ and $\beta' = \mathcal{T}'_1 \uplus \dots \uplus \mathcal{T}'_d$, we say that $\beta \overset{r}{=} \beta'$ if $\mathcal{T}_i \overset{r}{=} \mathcal{T}'_i$ for all $i \in [d]$.

LEMMA 4. *Let* $\beta = \biguplus_{i=1}^{t} \mathcal{T}_i$ *and* $\beta' = \biguplus_{i=1}^{t} \mathcal{T}'_i$ *be collections of sub-tree sets and let*

$$V_v \in \left[\beta \uplus \biguplus_{i \in I} \delta_i \cdot \mathcal{B}^{\uparrow}_{i,d_i,x_i} \right] \quad and \quad V_{v'} \in \left[\beta' \uplus \biguplus_{i \in I} \delta_i \cdot \mathcal{B}^{\uparrow}_{i,d'_i,x'_i} \right]$$

for a set of integers I *and integers* d_i, d'_i, x_i, *and* x'_i *for* $i \in I$. *Let* $r \geq 0$ *be an integer. If for all* $i \in I$, $d_i = d'_i$ *or* $r \leq 1 + \min\{d_i, d'_i\}$, *it holds that*

$$\mathcal{V}_{v_1} \overset{r}{=} \mathcal{V}_{v_2} \iff \beta \overset{r-1}{=} \beta'.$$

PROOF. Assume that the roots of the subtree of V_v and $V_{v'}$ are in clusters C and C', respectively. W.l.o.g., we assume that $d' \leq d$. Note that we have $d' \geq 1 + \min_{i \in I} \min\{d_i, d'_i\}$ and thus $r \leq d'$. In the construction process of G_k, C and C' have been created in steps $k-d$ and $k-d'$, respectively.

By Definition 1, all subtrees with depth $d^* < d'$ have grown identically in both views V_v and $V_{v'}$. The remaining subtrees of $V_{v'}$ were all created in step $k-d'+1$ and have depth $d'-1$. The corresponding subtrees in V_v have at least the same depth and the same structure up to that depth. Hence paths of length at most d' which start at the roots of V_v and V_v, go into one of the subtrees in $\mathcal{B}^{\uparrow}_{i_0,d_{i_0},x_{i_0}}$ and $\mathcal{B}^{\uparrow}_{i_0,d'_{i_0},x'_{i_0}}$ for $i_0 \in I$ and do not return to clusters C and C' must be identical. Further, consider all paths which, after $s \leq d'$ hops, return to C and C over link

δ_{i_0+1}. After these s hops, they return to the original cluster and see views

$$V'_v \in \left[\beta \uplus (\delta_{i_0} - 1) \cdot \mathcal{B}^{\uparrow}_{i_0,d_{i_0},x_{i_0}} \uplus \biguplus_{i \in I \setminus \{i_0\}} \delta_i \cdot \mathcal{B}^{\uparrow}_{i,d_i,x_i} \right] \quad \text{and}$$

$$V'_{v'} \in \left[\beta' \uplus (\delta_{i_0} - 1) \cdot \mathcal{B}^{\uparrow}_{i_0,d'_{i_0},x'_{i_0}} \uplus \biguplus_{i \in I \setminus \{i_0\}} \delta_i \cdot \mathcal{B}^{\uparrow}_{i,d'_i,x'_i} \right],$$

differing from V_v and $V_{v'}$ only in having $\delta_{i_0} - 1$ instead of δ_{i_0} subtrees in $\mathcal{B}^{\uparrow}_{i_0,d_{i_0},x_{i_0}}$ and $\mathcal{B}^{\uparrow}_{i_0,d'_{i_0},x'_{i_0}}$, respectively. This does not affect β and β' and therefore,

$$V_{v_1} \stackrel{r}{=} V_{v_2} \impliedby V'_{v_1} \stackrel{r-s}{=} V'_{v_2} \wedge \beta \stackrel{r-1}{=} \beta' , \; s > 1.$$

Note that $s \leq d'$ implies $s \leq r$. Thus, the same argument can be repeated until $r - s = 0$ and because $V'_v \stackrel{0}{=} V'_{v'}$, we can conclude that $V_v \stackrel{r}{=} V_v$ and thus the lemma follows. □
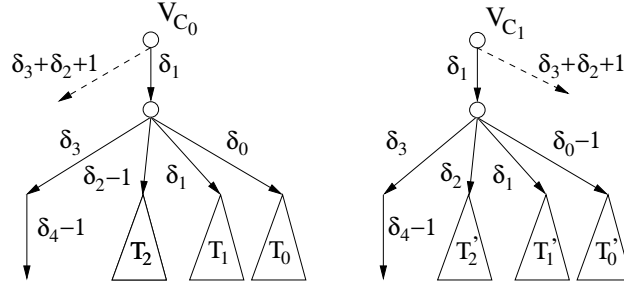

Fig. 3.   The view-trees V_{C_0} and V_{C_1} in G_3 seen upon using link δ_1.

Figure 3 shows a part of the view-trees of nodes in C_0 and C_1 in G_3. The figure shows that the subtrees with links δ_0 and δ_2 cannot be matched directly to one another because of the different placement of the -1. It turns out that this inherent difference appears in every step of our theorem. However, the following lemma shows that the subtrees T_0 and T_2 (T'_0 and T'_2) are equal up to the required distance and hence, nodes are unable to distinguish them. It is this crucial property of our cluster tree, which allows us to "move" the "-1" between links δ_i and δ_{i+2} and enables us to derive the main theorem.

LEMMA 5. *For all $i \in \{2, \ldots, k\}$, we have*

$$M_i \stackrel{k-i-1}{=} \mathcal{B}^{\uparrow}_{i-2,k-i,2}.$$

PROOF. By Lemma 3, we have

$$M_i \in \left[\biguplus_{j \in \{0,\dots,k\}\setminus\{i-1\}} \delta_j \cdot \mathcal{B}^\uparrow_{j,k-j,1} \uplus (\delta_{i-1} - 1) \cdot \mathcal{B}^\uparrow_{i-1,k-i+1,1} \right]$$

$$\mathcal{B}^\uparrow_{i-2,k-i,2} \subseteq \left[\biguplus_{\substack{j \in \{0,\dots,i+1\} \\ \setminus\{i-1\}}} \delta_j \cdot \mathcal{B}^\uparrow_{j,k-i-1,3} \uplus \biguplus_{j \in \{i+2,\dots,k\}} \delta_j \cdot \mathcal{B}^\uparrow_{j,k-j,3} \uplus (\delta_{i-1} - 1) \cdot \mathcal{B}^\downarrow_{i-1,k-i+1,1} \right]$$

Consider the subtrees of the form $\mathcal{B}^\uparrow_{j,d,x}$ for $j \neq i - 1$ in both cases. For $j \leq i + 1$, the depths of the subtrees are $k - j$ and $k - i - 1 \leq k - j$, respectively. For $j > i+1$, the depths are $k - j$ in both cases. We can therefore apply Lemma 4 to get that

$$M_i \stackrel{k-i-1}{=} \mathcal{B}^\uparrow_{i-2,k-i,2} \Longleftarrow \mathcal{B}^\uparrow_{i-1,k-i+1,1} \stackrel{k-i-2}{=} \mathcal{B}^\downarrow_{i-1,k-i+1,1}. \tag{4}$$

To show that the right-hand side of (4) holds, we plug $\mathcal{B}^\uparrow_{i-1,k-i+1,1}$ and $\mathcal{B}^\downarrow_{i-1,k-i+1,1}$ into Lemma 3 and use the derivation rules:

$$\mathcal{B}^\uparrow_{i-1,k-i+1,1} \subseteq \left[\mathcal{F}_{\{i\},k-i+1,1} \uplus \mathcal{D}_{k-i+1,1} \uplus (\delta_i - 1) \cdot M_i \right]$$

$$= \left[\mathcal{F}_{\{i-2,i\},k-i+1,1} \uplus \mathcal{D}_{k-i+1,1} \uplus (\delta_i - 1) \cdot M_i \uplus \delta_{i-2} \cdot \mathcal{B}^\uparrow_{i-2,k-i,2} \right]$$

$$\mathcal{B}^\downarrow_{i-1,k-i+1,1} \subseteq \left[\mathcal{F}_{\{i-2,i\},k-i+1,1} \uplus \mathcal{D}_{k-i+1,1} \uplus \delta_i \cdot M_i \uplus (\delta_{i-2} - 1) \cdot \mathcal{B}^\uparrow_{i-2,k-i,2} \right]$$

The two expressions are equal except for the placement of the "-1". Therefore, we get $\mathcal{B}^\uparrow_{i-1,k-i+1,1} \stackrel{k-i-2}{=} \mathcal{B}^\downarrow_{i-1,k-i+1,1}$ if $M_i \stackrel{k-i-3}{=} \mathcal{B}^\uparrow_{i-2,k-i,2}$. Hence, we have shown that

$$M_i \stackrel{k-i-1}{=} \mathcal{B}^\uparrow_{i-2,k-i,2} \Longleftarrow \mathcal{B}^\uparrow_{i-1,k-i+1,1} \stackrel{k-i-2}{=} \mathcal{B}^\downarrow_{i-1,k-i+1,1} \Longleftarrow M_i \stackrel{k-i-3}{=} \mathcal{B}^\uparrow_{i-2,k-i,2}.$$

This process can be continued using exactly the same rules until the requirement becomes that either

$$\mathcal{B}^\uparrow_{i-1,k-i+1,1} \stackrel{0}{=} \mathcal{B}^\downarrow_{i-1,k-i+1,1} \quad \text{or} \quad M_i \stackrel{0}{=} \mathcal{B}^\uparrow_{i-2,k-i,2},$$

which is always true. $\qquad \square$

Finally, we are ready to prove the main theorem.

THEOREM 6. *Consider graph G_k. Let V_{C_0} and V_{C_1} be the view-trees of two adjacent nodes in clusters C_0 and C_1, respectively. Then, $V_{C_0} \stackrel{k}{=} V_{C_1}$.*

PROOF. By the construction of G_k, the view-trees of V_{C_0} and V_{C_1} can be written as

$$V_{V_0} \in \left[\biguplus_{j=0}^{k} \delta_j \cdot \mathcal{B}^\uparrow_{j,k-j,1} \right] \quad \text{and}$$

$$V_{V_1} \in \left[\delta_1 \cdot M_1 \uplus \biguplus_{j \in \{0,\dots,k\}\setminus\{1\}} \delta_j \cdot \mathcal{B}^\uparrow_{j,k-j,2} \right].$$

It follows that $V_{C_0} \overset{k}{=} V_{C_1} \Longleftarrow \mathcal{B}_{1,k-1,1}^\uparrow \overset{k-1}{=} M_1$ by Lemma 4. To prove that $\mathcal{B}_{1,k-1,1}^\uparrow \overset{k-1}{=} M_1$, we show that

$$\mathcal{B}_{k-s,s,1}^\uparrow \overset{s}{=} M_{k-s} \quad \text{for all} \quad s \in \{0, \ldots, k-1\}. \tag{5}$$

We show Equation (5) by induction on s. The statement is trivially true for $s = 0$ because any two trees are 0-equal. For the induction step, consider the derivation rules for $\mathcal{B}_{k-s,s,1}^\uparrow$ and M_{k-s}:

$$\mathcal{B}_{k-s,s,1}^\uparrow \subseteq \left[\biguplus_{j=0}^{k-s} \delta_j \cdot \mathcal{B}_{j,s-1,2}^\uparrow \uplus \biguplus_{j=k-s+2}^{k} \delta_j \cdot \mathcal{B}_{j,k-j,2}^\uparrow \uplus (\delta_{k-s+1} - 1) \cdot M_{k-s+1} \right]$$

$$M_{k-s} \in \left[\biguplus_{j \in \{0,\ldots,k\} \setminus \{k-s-1\}} \delta_j \cdot \mathcal{B}_{j,k-j,1}^\uparrow \uplus (\delta_{k-s-1} - 1) \cdot \mathcal{B}_{k-s-1,s+1,1}^\uparrow \right]$$

Consider the subtrees of the form $\mathcal{B}_{j,d,x}^\uparrow$ for $j \notin \{k-s-1, k-s+1\}$ in both expressions. For $j \leq k-s$, the depths of the subtrees are $s-1 < k-j$ and $k-j$, respectively. For $j > k-s+1$, the depth is $k-j$ in both cases. Hence, we can apply Lemma 4 to get

$$\mathcal{B}_{k-s,s,1}^\uparrow \overset{s}{=} M_{k-s} \Longleftarrow \tag{6}$$
$$a \cdot \mathcal{B}_{k-s-1,s+1,2}^\uparrow \uplus (b-1) \cdot M_{k-s+1} \overset{s-1}{=} (a-1) \cdot \mathcal{B}_{k-s-1,s+1,1}^\uparrow \uplus b \cdot \mathcal{B}_{k-s+1,s-1,1}^\uparrow$$

for $a = \delta_{k-s+1}$ and $b = \delta_{k-s-1}$. By Lemma 5, we have $\mathcal{B}_{k-s-1,s+1,2}^\uparrow \overset{s-1}{=} M_{k-s+1}$ and thus the right-hand side of (6) is true by the induction hypothesis. This concludes the induction to show (5) and thus also the proof of the theorem. □

Remark. As a side-effect, the proof of Theorem 6 highlights the fundamental significance of the *critical path* $P = (\delta_1, \delta_2, \ldots, \delta_k)$ in CT_k. After following path P, the view of a node $v \in C_0$ ends up in the leaf-cluster neighboring C_0 and sees δ_{i+1} neighbors. Following the same path, a node $v' \in C_1$ ends up in C_0 and sees $\sum_{j=0}^{i} \delta_j - 1$ neighbors. There is no way to match these views. This inherent inequality is the underlying reason for the way G_k is defined: It must be ensured that the critical path is at least k hops long.

3.4 Analysis

In this subsection, we derive the lower bounds on the approximation ratio of k-local MVC algorithms. Let OPT be an optimal solution for MVC and let ALG be the solution computed by any algorithm. The main observation is that adjacent nodes in the clusters C_0 and C_1 have the same view and therefore, every algorithm treats nodes in both of the two clusters the same way. Consequently, ALG contains a significant portion of the nodes of C_0, whereas the optimal solution covers the edges between C_0 and C_1 entirely by nodes in C_1.

LEMMA 7. *Let ALG be the solution of any distributed (randomized) vertex cover algorithm which runs for at most k rounds. When applied to G_k as constructed in Subsection 3.2 in the worst case (in expectation), ALG contains at least half of the nodes of C_0.*

PROOF. Let $v_0 \in C_0$ and $v_1 \in C_1$ be two arbitrary, adjacent nodes from C_0 and C_1. We first prove the lemma for deterministic algorithms. The decision whether a given node v enters the vertex cover depends solely on the topology $\mathcal{T}_{v,k}$ and the labeling $\mathcal{L}(\mathcal{T}_{v,k})$. Assume that the labeling of the graph is chosen uniformly at random. Further, let $p_0^{\mathcal{A}}$ and $p_1^{\mathcal{A}}$ denote the probabilities that v_0 and v_1, respectively, end up in the vertex cover when a deterministic algorithm $\mathcal{A}$ operates on the randomly chosen labeling. By Theorem 6, v_0 and v_1 see the same topologies, that is, $\mathcal{T}_{v_0,k} = \mathcal{T}_{v_1,k}$. With our choice of labels, v_0 and v_1 also see the same distribution on the labelings $\mathcal{L}(\mathcal{T}_{v_0,k})$ and $\mathcal{L}(\mathcal{T}_{v_1,k})$. Therefore it follows that $p_0^{\mathcal{A}} = p_1^{\mathcal{A}}$.

We have chosen v_0 and v_1 such that they are neighbors in G_k. In order to obtain a feasible vertex cover, at least one of the two nodes has to be in it. This implies $p_0^{\mathcal{A}} + p_1^{\mathcal{A}} \geq 1$ and therefore $p_0^{\mathcal{A}} = p_1^{\mathcal{A}} \geq 1/2$. In other words, for all nodes in C_0, the probability to end up in the vertex cover is at least $1/2$. Thus, by the linearity of expectation, at least half of the nodes of C_0 are chosen by algorithm $\mathcal{A}$. Therefore, for every deterministic algorithm $\mathcal{A}$, there is at least one labeling for which at least half of the nodes of C_0 are in the vertex cover.[7]

The argument for randomized algorithms is now straight-forward using Yao's minimax principle. The expected number of nodes chosen by a randomized algorithm cannot be smaller than the expected number of nodes chosen by an optimal deterministic algorithm for an arbitrarily chosen distribution on the labels. □

Lemma 7 gives a lower bound on the number of nodes chosen by any k-local MVC algorithm. In particular, we have that $E[|ALG|] \geq |C_0|/2 = n_0/2$. We do not know OPT, but since the nodes of cluster C_0 are not necessary to obtain a feasible vertex cover, the optimal solution is bounded by $|OPT| \leq n - n_0$. In the following, we define

$$\delta_i := 2^{i(i-1)/2}\delta^i \quad , \; \forall i \in \{0, \ldots, k+1\} \tag{7}$$

for some value δ. Hence, $\delta_0 = 1$ and for all $i \in \{0, \ldots, k\}$, we have $\delta_{i+1}/\delta_i = 2^i\delta$.

LEMMA 8. *If $\delta > 2$, the number of nodes n of G_k is*

$$n \; \leq \; n_0 \left(1 + \frac{2}{\delta - 2}\right).$$

PROOF. Consider a cluster C of size $|C|$ on some level ℓ and some neighbor cluster C' on level $\ell + 1$. For some $i \in \{0, \ldots, k\}$, all nodes in C have δ_i neighbors in cluster C' and all nodes in C' have δ_{i+1} neighbors in cluster C. We therefore have $|C|/|C'| = \delta_{i+1}/\delta_i$ for some $i \geq 0$. Further, for every i, C can have at most 1 such neighboring cluster on level $\ell + 1$. The total number of nodes in level $\ell + 1$ clusters that are neighbors of C can therefore be bounded as

$$|C| \cdot \sum_{i=0}^{k} \frac{\delta_i}{\delta_{i+1}} = |C| \cdot \sum_{i=0}^{k} \frac{1}{2^i\delta} \leq |C| \cdot \sum_{i=0}^{\infty} \frac{1}{2^i\delta} = \frac{2}{\delta} \cdot |C|.$$

---

[7] In fact, since at most $|C_0|$ such nodes can be in the vertex cover, for at least $1/3$ of the labelings, the number exceeds $|C_0|/2$.

Hence, the number of nodes decreases by at least a factor of $\delta/2$ on each level. For $\delta > 2$, the total number of nodes can thus be bounded by

$$\sum_{i=0}^{\infty} n_0 \cdot \left(\frac{2}{\delta}\right)^i = n_0 \cdot \frac{\delta}{\delta - 2}.$$

$\square$

It remains to determine the relationship between δ and n_0 such that G_k can be realized as described in Subsection 3.2. There, the construction of G_k with large girth is based on a smaller instance G'_k where girth does not matter. Using (7) (i.e. $\delta_i := 2^{i(i-1)/2}\delta^i$), we can now tie up this loose end and describe how to obtain G'_k. Let C_i and C_j be two adjacent clusters with $\ell(C_a, C_b) = (\delta_i, \delta_{i+1})$. We require that $|C_a|/|C_b| = \delta_{i+1}/\delta_i$. Hence, C_i and C_j can simply be connected by as many complete bipartite graphs $K_{\delta_i, \delta_{i+1}}$ as necessary.

To compute the necessary cluster sizes to do this, let c_ℓ be the size of the smallest cluster on level ℓ. We have $c_0 = n'_0$ and $c_{\ell-1}/c_\ell = \delta_{k+1}/\delta_k = 2^k\delta$. Hence, the size of the smallest cluster decreases by a factor $2^k\delta$ when going from some level to the next one. The smallest cluster $C_{\min}$ of G'_k is on level $k + 1$. Because each node in the neighboring cluster of $C_{\min}$ on level k has δ_k neighbors in $C_{\min}$, $C_{\min}$ needs to have size at least δ_k. We thus choose $C_{\min}$ of size $c_{k+1} = |C_{\min}| = \delta_k$. From $c_{\ell-1}/c_\ell = 2^k\delta$, we thus get

$$n'_0 = c_0 = c_{k+1} \cdot 2^{k(k+1)}\delta^k = \delta_k \cdot 2^{k(k+1)}\delta^k = 2^{k(3k+1)/2}\delta^{2k} \leq (2^k\delta)^{2k}.$$

If we assume that $\delta > 4$, we have $n \leq 2n_0$, by Lemma 8. Applying the construction of Subsection 3.2, we get $n \leq n' \cdot \langle n'\rangle^{2k-5}$, where $\langle n'\rangle$ denotes the smallest prime power larger than or equal to n'. We have $\langle n'\rangle < 2n'$. Putting everything together, we obtain

$$n \;\leq\; (4n'_0)^{2k-4} \;\leq\; 2^{4k^3 + 4k}\delta^{4k^2}. \tag{8}$$

THEOREM 9. *For every constant $\varepsilon > 0$, there are graphs G, such that in k communication rounds, every distributed algorithm for the minimum vertex cover problem on G has approximation ratios at least*

$$\Omega\left(n^{\frac{1/4-\varepsilon}{k^2}}\right) \quad and \quad \Omega\left(\Delta^{\frac{1-\varepsilon}{k+1}}\right),$$

where n and Δ denote the number of nodes and the highest degree in G, respectively.

PROOF. For $\delta \geq 4$, based on Lemmas 7 and 8, the approximation ratio of any k-rounds algorithm is at least $\Omega(\delta)$. We thus need to bound δ as a function of the number of nodes n and the largest degree Δ of G_k.

In the following, assume that we choose $\delta \geq a^k$ for a large enough constant $a > 1$. Using Inequality (8), we then get that the number of nodes is at most $n \leq 2^{4k^3 + 4k}a^{4k^3}$. For any constant $\varepsilon' > 0$, this is less that $a^{4k^3/(1-4\varepsilon')} = \delta^{(1/4-\varepsilon)/k^2}$ if the constant a is chosen large enough. Hence, the first lower bound follows. The largest degree of G_k is $\Delta = \delta_{k+1} = 2^{k(k+1)/2}\delta^{k+1} \leq (2a)^{k(k+1)}$. Choosing a large enough, this is at least $a^{(k+1)/(1-\varepsilon)}$, implying also the second lower bound. $\square$

THEOREM 10. *In order to obtain a constant approximation ratio, every distributed algorithm for the MVC problem requires at least $\Omega\left(\sqrt{\log n}\right)$ and $\Omega\left(\log \Delta\right)$ communication rounds.*
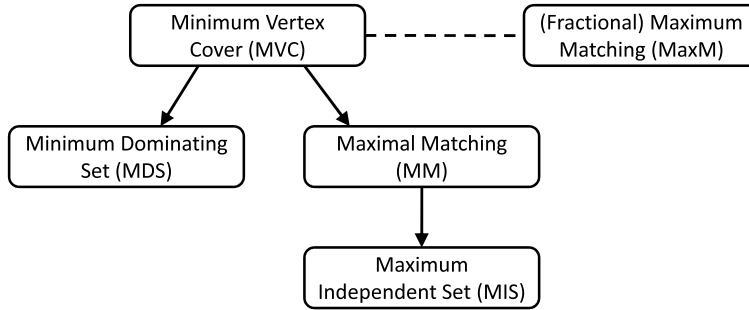
Fig. 4. Locality Preserving Reductions. The dotted line between MVC and MaxM implies that we do not know of a direct locality preserving reduction between the covering and packing problem, but the respective lower bound constructions are based on a common cluster tree structure.

PROOF. Follows directly from Theorem 9. □

4. LOCALITY-PRESERVING REDUCTIONS

Using the MVC lower bound, we can now derive lower bounds for several of the other classical graph problems defined in Section 1.2. Interestingly, the *hardness of distributed approximation* lower bound on the MVC problem also gives raise to local computability lower bounds for two of the most fundamental exact problems in distributed computing: MIS and MM.

Specifically, we use the notion of *locality preserving reductions* to show that a number of other problems can be reduce to MVC with regard to their local computability/approximability. This implies that, like MVC, these problems fall into the pseudo-local class of problems. Figure 4 shows the hierarchy of locality preserving reductions derived in this section.

4.1 Lower Bounds for Minimum Dominating Set

In a non-distributed setting, MDS in equivalent to the general minimum set cover problem, whereas MVC is a special case of set cover which can be approximated much better. It is therefore not surprising that also in a distributed environment, MDS is harder than MVC. In the following, we formalize this intuition giving a locality-preserving reduction from MVC to MDS.

THEOREM 11. *For any $\varepsilon > 0$, there are graphs G, such that in k communication rounds, every (possibly randomized) distributed algorithm for the minimum dominating set problem on G has approximation ratios at least*

$$\Omega\left(n^{\frac{1/4-\varepsilon}{k^2}}\right) \quad and \quad \Omega\left(\Delta^{\frac{1-\varepsilon}{k+1}}\right),$$

where n and Δ denote the number of nodes and the highest degree in G, respectively.

PROOF. To obtain a lower bound for MDS, we consider the line graph $L(G_k)$ of G_k. The nodes of a line graph $L(G)$ of G are the edges of G. Two nodes in $L(G)$ are connected by an edge whenever the two corresponding edges in G are incident

to the same node. Assuming that initially each node knows all its incident edges, a k-round computation on the line graph of G can be simulated in k round on G, i.e., in particular G_k and $L(G_k)$ have the same locality properties.

A dominating set of the line graph of a graph $G = (V, E)$ is a subset $E' \subseteq E$ of the edges such that for every $\{u, v\} \in E$, there is an edge $\{u', v'\} \in E'$ such that $\{u, v\} \cap \{u', v'\} \neq \emptyset$. Hence, for every edge dominating set E', the node set $S = \bigcup_{\{u,v\} \in E'} \{u, v\}$ is a vertex cover of G of size $|S| \leq 2|E'|$. In the other direction, given a vertex cover S of G, we obtain a dominating set of E' of $L(G)$ of the same size $|E'| = |S|$ simply by adding some edge $\{u, v\}$ to E' for every node $u \in S$. Therefore, up to a factor of at most 2 in the approximation ratio, the two problems are equivalent and thus the claim of the theorem follows.   □

Remark. Using the same locality-preserving reduction as from MVC to MDS, it can also be shown that solving the fractional version of MDS is at least as hard as the fractional version of MVC. Since Theorem 9 also holds for fractional MVC (the integrality gap of MVC is at most 2), Theorem 11 and Corollary 12 can equally be stated for fractional MDS, that is, for the standard linear programming relaxation of MDS.

COROLLARY 12. *In order to obtain a constant approximation ratio for minimum dominating set or fractional minimum dominating set, there are graphs on which every distributed algorithm requires time*

$$\Omega\left(\sqrt{\log n}\right) \quad and \quad \Omega\left(\log \Delta\right).$$

For polylogarithmic approximation ratios, we obtain the following lower bounds:

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

PROOF. The corollary follows directly from Theorem 11.   □

Remark. The MDS problem on the line graph of G is also known as the minimum edge dominating set problem of G (an edge dominating set is a set of edges that 'covers' all edges). Hence, the above reduction shows that also the minimum edge dominating set problem is hard to approximate locally.

4.2   Lower Bounds for Maximum Matching

While MVC and MDS are standard covering problems, the lower bound can also be extended to *packing problems*. Unfortunately, we are not aware of a simple locality-preserving reduction from MVC to a packing problem, but we can derive the result by appropriately adjusting the cluster graph from Section 3.2. In fact, we prove the result for the *fractional maximum matching problem* in which edges may be selected fractionally, and the sum of these fractional values incident at a single node must not exceed 1. Let $E(v)$ denotes the set of edges incident to node v.

The basic idea of the lower bound follows along the lines of the MVC lower bound in Section 3. The view of an edge is defined to be the union of its incident nodes' views. In other words, two edges (u, v) and (u', v') have the same view if $\mathcal{V}_{u,k} \cup \mathcal{V}_{v,k} = \mathcal{V}_{u',k} \cup \mathcal{V}_{v',k}$. The idea is to construct a graph H_k which contains a
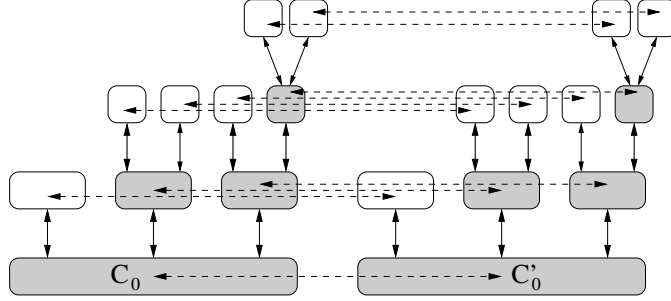
Fig. 5. The structure of lower-bound graph H_k.

large set $E' \subset E$ of edges with equal view up to distance k. This implies that, in expectation, the fractional values y_e assigned to the edges in E' must be equal. H_k is constructed in such a way, however, that there are edges in E' who are incident to many other edges in E', whereas the majority of edges in E' are incident to only a few such edges. Clearly, an optimal solution consists of mainly the latter edges, leading to a matching with large cardinality. On the other hand, every distributed k-local algorithm assigns equal fractional values y_e to all edges in E' in expectation. In order to keep the feasibility at the nodes incident to many edges in E', this fractional value must be rather small, which leads to the sub-optimality ultimately captured in the theorem.

The construction of H_k uses the lower-bound graph G_k of the MVC lower bound as follows. Let G_k and G'_k be two identical copies of the MVC lower-bound graph defined in Section 3. The graph H_k takes the two copies and connects each node in G_k to its counterpart in G'_k as illustrated in Figure 5. Formally, let $\phi : V(G_k) \rightarrow V(G'_k)$ be an isomorphism mapping nodes of G_k to G'_k. Graph H_k consists of G_k and G'_k, as well as additional edges between nodes $v \in G_k$ and $w \in G'_k$ if and only if $w = \phi(v)$. C'_k denotes the set of nodes in G'_k corresponding to cluster C_i in G_k. Furthermore, we use the abbreviations $S_0 := C_0 \cup C'_0$ and $S_1 := C_1 \cup C'_1$.

By the construction of H_k and the structural properties proven in Theorem 6, the following lemma follows immediately.

LEMMA 13. *Let v and w be two arbitrary nodes in $S_0 \cup S_1$ of H_k. It holds that v and w see the same topology up to distance k.*

Lemma 13 implies that no distributed k-local algorithm can distinguish between edges connecting two nodes in $S_0 \cup S_1$. In particular, this means that edges between C_0 and C_1 cannot be distinguished from edges between C_0 and C'_0. In the sequel, let OPT be the value of the optimal solution for fractional maximum matching and let ALG be the value of the solution computed by any algorithm.

LEMMA 14. *When applied to H_k, any distributed, possibly randomized algorithm which runs for at most k rounds computes, in expectation, a solution of at most $ALG \leq |S_0|/(2\delta) + (|V| - |S_0|)$.*

PROOF. First, consider deterministic algorithms. The decision of which value y_e is assigned to edge $e = (v, v)$ depends only on the view the topologies $\mathcal{T}_{u,k}$

and $\mathcal{T}_{v,k}$ and the labelings $\mathcal{L}(\mathcal{T}_{u,k})$ and $\mathcal{L}(\mathcal{T}_{v,k})$, which u and v can collect during the k communication rounds. Assume that the labeling of H_k is chosen uniformly at random. In this case, the labeling $\mathcal{L}(\mathcal{T}_{u,k})$ for any node $u \in V$ is also chosen uniformly at random.

All edges connecting nodes in S_0 and S_1 see the same topology. If the node's labels are distributed uniformly at random, it follows that the *distribution of the views* (and therefore the distribution of the y_e) is the same for all edges connecting nodes in S_0 and S_1. We denote the random variables describing the distribution of the y_e by Y_e. Every node $u \in S_1$ has $\delta_1 = \delta$ neighbors in S_0. Therefore, for edges e between nodes in S_0 and S_1, it follows by linearity of expectation that $E[Y_e] \leq 1/\delta$ because otherwise, there exists at least one labeling for which the computed solution is not feasible. On the other hand, consider an edge e' having both end-points in S_0. By Lemma 13, these edges have the same view as edges e between S_0 and S_1. Hence, for y'_e of e', it must equally hold that $E[Y'_e] \leq 1/\delta$. Because there are $|S_0|/2$ such edges, the expected total value contributed to the objective function by edges between two nodes in S_0 is at most $|S_0|/(2\delta)$.

Next, consider all edges which do not connect two nodes in S_0. Every such edge has at least one end-point in $V \setminus S_0$. In order to obtain a feasible solution, the total value of all edges incident to a set of nodes V', can be at most $|V'| = |V \setminus S_0|$. This can be seen by considering the dual problem, a kind of minimum vertex cover where some edges only have one incident node. Taking all nodes of V' (assigning 1 to the respective variables) yields a feasible solution for this vertex cover problem. This concludes the proof for deterministic algorithms.

For probabilistic algorithms, we can apply an identical argument based on Yao's minimax principle as in the MVC lower bound (cf. Lemma 7).   □

Lemma 14 yields an upper bound on the objective value achieved by any k-local fractional maximum matching algorithm. On the other hand, it is clear that choosing all edges connecting corresponding nodes of G_k and G'_k is feasible and hence, $OPT \geq n/2 \geq |S_0|/2$. Let α denote the approximation ratio achieved by any k-local distributed algorithm, and assume—as in the MVC proof—that $k+1 \leq \delta/2$. Using the relationship between n, $|S_0|$, δ, and k proven in Lemma 8 and combining it with the bound on ALG gives raise to the following theorem.

THEOREM 15. *For every $\varepsilon > 0$, there are graphs G, such that in k communication rounds, every (possibly randomized) distributed algorithm for the (fractional) maximum matching problem on G has approximation ratios at least*

$$\Omega\left(n^{\frac{1/4-\varepsilon}{k^2}}\right) \quad and \quad \Omega\left(\Delta^{\frac{1-\varepsilon}{k+1}}\right),$$

where n and Δ denote the number of nodes and the highest degree in G, respectively.

PROOF. By Lemma 14 and by Lemma 8, on H_k, the approximation ratio of any, possibly randomized, (fractional) maximum matching algorithm is $\Omega(\delta)$. Because asymptotically, the relations between δ and the largest degree Δ and the number of nodes n is the same in the MVC lower bound graph G_k and in H_k, the lower bounds follow in the same way as the lower bounds in Theorem 9.   □

COROLLARY 16. *In order to obtain a constant approximation ratio, every distributed algorithm for the (fractional) maximum matching problem requires at least*

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

communication rounds.

4.3 Lower Bounds for Maximal Matching

A maximal matching M of a graph G is a maximal set of edges which do not share common end-points. Hence, a maximal matching is a set of non-adjacent edges M of G such that all edges in $E(G) \setminus M$ have a common end-point with an edge in M. The best known lower bound for the distributed computation of a maximal matching is $\Omega(\log^* n)$ which holds for rings [Linial 1992].

THEOREM 17. *There are graphs G on which every distributed, possibly randomized algorithm in expectation requires time*

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

to compute a maximal matching. This bound holds even if message size is unlimited and nodes have unique identifiers.

PROOF. It is well known that the set of all end-points of the edges of a maximal matching form a 2-approximation for MVC. This simple 2-approximation algorithm is commonly attributed to Gavril and Yannakakis. For deterministic algorithms, the lower bound for the construction of a maximal matching in Theorem 17 therefore directly follows from Theorem 10.

Generalizing this result to randomized algorithms, however, still requires some work. The problem is that Theorem 9 lower bounds the achievable approximation ratio by distributed algorithms whose time complexity is exactly k. That is, it does not provide a lower bound for randomized algorithms whose time complexity is at most k in expectation or with a certain probability. As stated in the theorem, however, we consider distributed algorithms that always compute a feasible solution, i.e., only the time complexity depends on randomness. In other words, Theorem 9 yields a bound on Monte Carlo type algorithms, whereas in the case of maximal matching, we are primarily interested in Las Vegas type algorithms.

In order to generalize the theorem to randomized algorithms, we give a transformation from an arbitrary distributed maximal matching algorithm $\mathcal{A}_M$ with expected time complexity T into a distributed vertex cover algorithm $\mathcal{A}_{\mathrm{VC}}$ with fixed time complexity $2T + 1$ and expected approximation ratio 11.

We first define an algorithm $\mathcal{A}'_{\mathrm{VC}}$. In a first phase, $\mathcal{A}'_{\mathrm{VC}}$ simulates $\mathcal{A}_M$ for exactly $2T$ rounds. Let $E_M \subseteq E$ be the set of edges selected after these rounds. In the second phase, every node v checks whether it has at most one incident edge in E_{VC}. If a node has more than one incident edge in E_{VC}, it removes all these edges from E_{VC}. Hence, E_{VC} forms a feasible matching, although not necessarily a maximal one.

It follows from Markov's inequality that when running $\mathcal{A}_M$ for $2T$ rounds, the probability for obtaining a feasible maximal matching is at least $1/2$. Therefore, algorithm $\mathcal{A}'_{\mathrm{VC}}$ outputs a matching that is maximal with probability at least $1/2$.

Let $V_{\mathrm{VC}} \subseteq V$ denote the set of all nodes incident to an edge in E_{VC}. If E_{VC} is a maximal matching, V_{VC} is a feasible vertex cover (with probability at least $1/2$). In any case, the construction of $\mathcal{A}'_{\mathrm{VC}}$ guarantees that $|V_{\mathrm{VC}}|$ is at most twice the size of an optimal vertex cover.

Algorithm $\mathcal{A}_{\mathrm{VC}}$ executes $c \cdot \ln \Delta$ independent runs of $\mathcal{A}'_{\mathrm{VC}}$ in parallel for a sufficiently large constant c. Let $V_{\mathrm{VC},i}$ be the node set V_{VC} constructed by the ith of the $c \cdot \ln \Delta$ runs of Algorithm $\mathcal{A}_{\mathrm{VC}}$. For each node $u \in V$, we define

$$x_u := 6 \cdot \frac{|\{i : u \in V_{\mathrm{VC},i}\}|}{c \cdot \ln \Delta}.$$

Algorithm $\mathcal{A}_{\mathrm{VC}}$ computes a vertex cover S as follows. All nodes with $x_u \geq 1$ join the initial set S. In one additional round, nodes that have an uncovered edge also join S to guarantee that S is a vertex cover.

Let OPT_{VC} be the size of an optimal vertex cover. Because for each i, $|V_{\mathrm{VC},i}| \leq 2OPT_{\mathrm{VC}}$, we get $\sum_{u \in V} x_u \leq 12 \cdot OPT_{\mathrm{VC}}$. For every edge $\{u, v\}$, in each run of $\mathcal{A}'_{\mathrm{VC}}$ that ends with a vertex cover, the set $\{i : u \in V_{\mathrm{VC},i}\}$ contains at least one of the two nodes $\{u, v\}$. Hence, if at least $1/3$ of the runs of $\mathcal{A}'_{\mathrm{VC}}$ produces a vertex cover, we have $x_u + x_v \geq 2$ for every edge $\{u, v\}$. Thus, in this case, taking all nodes u for which $x_u \geq 1$ gives a valid vertex cover of size at most $\sum_{u \in V} x_u$. Let X be the number of runs of $\mathcal{A}'_{\mathrm{VC}}$ that result in a vertex cover. Because the runs are independent and since each of them gives a vertex cover with probability at least $1/2$, we can bound the number of successful runs using a Chernoff bound:

$$\Pr\left[X < \frac{c \ln \Delta}{3}\right] = \Pr\left[X < \left(1 - \frac{1}{3}\right) \cdot \frac{c \ln \Delta}{2}\right] \leq e^{-\frac{c}{36} \ln \Delta} = \frac{1}{\Delta^{c/36}}.$$

For $c \geq 36$, the probability that the nodes u with $x_u \geq 1$ do not form a vertex cover is at most $1/\Delta$. Thus, with probability at least $1 - 1/\Delta$, the algorithm computes a vertex cover of size at most $10 OPT_{\mathrm{VC}}$. With probability at most $1/\Delta$, the vertex cover has size at most $n \leq \Delta OPT_{\mathrm{VC}}$. The expected size of the computed vertex cover therefore is at most $11 OPT_{\mathrm{VC}}$. The theorem now follows from Theorem 10. □

4.4   Lower Bounds for Maximal Independent Set (MIS)

As in the case of a maximal matching, the best currently known lower bound on the distributed complexity of an MIS has been Linial's $\Omega(\log^* n)$ lower bound. Using a locality-preserving reduction from MM to MIS, we can strengthen this lower bound on general graphs as formalized in the following theorem.

THEOREM 18. *There are graphs G on which every distributed, possibly randomized algorithm in expectation requires time*

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

to compute a maximal independent set (MIS). This bound holds even if message size is unlimited and nodes have unique identifiers.

PROOF. For the MIS problem, we again consider the line graph $L(G_k)$ of G_k, i.e., the graph induced by the edges of G_k. The MM problem on a graph G is equivalent to the MIS problem on $L(G)$. Further, if the real network graph is G, k

communication rounds on $L(G)$ can be simulated in $k+\mathrm{O}(1)$ communication rounds on G. Therefore, the times t to compute an MIS on $L(G_k)$ and t' to compute an MM on G_k can only differ by a constant, $t \geq t' - O(1)$. Let n' and Δ' denote the number of nodes and the maximum degree of G_k, respectively. The number of nodes n of $L(G_k)$ is less than $n'^2/2$, the maximum degree Δ of G_k is less than $2\Delta'$. Because n' only appears as $\log n'$, the power of 2 does not hurt and the theorem holds ($\log n = \Theta(\log n')$). □

4.5 Connected Dominating Set Lower Bound

In this section, we extend our lower bound to the minimum connected dominating set problem (MCDS). First first start with a simple technical lemma that relates the respective sizes of an optimal dominating set and an optimal connected dominating set in a graph.

LEMMA 19. *Let $G = (V, E)$ be a connected graph and let DS_{OPT} and CDS_{OPT} be the sizes of optimal dominating and connected dominating sets of G. It holds that $CDS_{OPT} < 3 \cdot DS_{OPT}$. Moreover, every dominating set D of G can be turned into a connected dominating set $D' \supseteq D$ of size $|D'| < 3|D|$.*

PROOF. Given G and D, we define a graph $G_D = (V_D, E_D)$ as follows. $V_D = D$ and there is an edge $(u, v) \in E_D$ between $u, v \in D$ if and only if $d_G(u, v) \leq 3$. We first show that G_D is connected. For the sake of contradiction assume that G_D is not connected. Then there is a cut (S, T) with $S \subseteq D, T = D \setminus S$, and $S, T \neq \emptyset$ such that

$$\forall u \in S, \forall v \in T \ : \ d_G(u, v) \geq 4. \tag{9}$$

Let $u \in S$ and $v \in T$ be such that

$$d_G(u, v) = \min_{u \in S, v \in T}(d_G(u, v)). \tag{10}$$

By Inequality (9), there is a node $w \in V$ with $d_G(u, w) \geq 2$ and $d_G(v, w) \geq 2$ on each shortest path connecting u and v. Because of Equation (10), we have that

$$\forall u \in S, \forall v \in T \ : \ d_G(u, w) \geq 2 \wedge d_G(v, w) \geq 2.$$

However this is a contradiction to the assumption that $D = S \cup T$ is a dominating set of G.

We can now construct a connected dominating set D' as follows. We first compute a spanning tree of G_D. For each edge (u, v) of the spanning tree, we add at most two nodes such that u and v become connected. Because the number of edges of the spanning tree is $|D| - 1$, this results in a connected dominating set of size at most $3|D| - 2$. □

Using this lemma, we can now derive the lower bound on the local approximability of the MCDS problem.

THEOREM 20. *Consider a (possibly randomized) k-round algorithm for the MCDS problem. There are graphs for which every such algorithm computes a connected dominating set S of size at least*
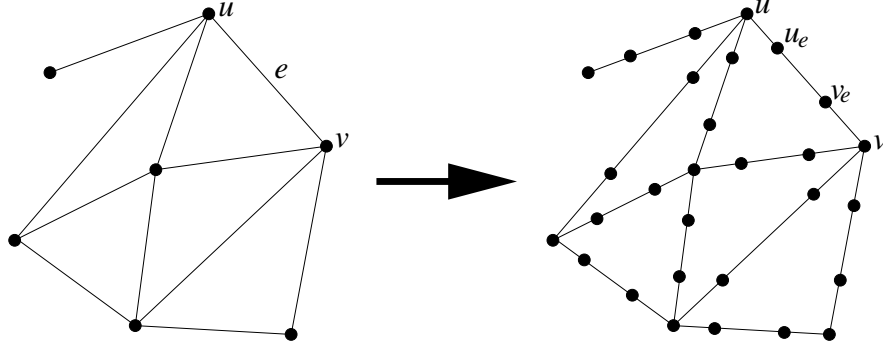
$$|S| \ \geq \ n^{\Omega(1/k)} \cdot \mathrm{CDS}_{\mathrm{OPT}},$$

Fig. 6. Graph transformation used for the distributed minimum connected dominating set lower bound

where $\mathrm{CDS}_{\mathrm{OPT}}$ *denotes the size of an optimal connected dominating set.*

PROOF. It follows from a well-known theorem (see e.g. [Bollobas 1978]) that there exist graphs $G = (V, E)$ with girth $g(G) \geq (2k + 1)/3$ and number of edges $|E| = n^{1+\Omega(1/k)}$. From any such graph G, we construct a graph G' as follows. For every edge $e = (u, v) \in E$, we generate additional nodes u_e and v_e. In G', there is an edge between u and u_e, between u_e and v_e, and between v and v_e. Note that there is no edge between u and v anymore. The described transformation is illustrated in Figure 6. We denote the set of all new nodes by W and the number of nodes of G' by $N = |V \cup W|$.

By the definition of G', the nodes in V form a dominating set of G'. Hence, an optimal connected dominating set on G' has size less than $3|V|$ by Lemma 19. Note that the construction described in Lemma 19 actually computes a spanning tree T on G and adds all nodes $u_e, v_e \in W$ to the dominating set for which (u, v) is an edge of T. To bound the number of nodes in the connected dominating set of a distributed algorithm, we have a closer look at the locality properties of G'. Because $g(G) \geq (2k + 1)/3$, the girth of G' is $g(G') \geq 2k + 1$. This means that in k communication rounds it is not possible to detect a cycle of G'. Hence, no node can locally distinguish G' from a tree. However, since on a tree all edges are needed to keep a connected graph, a k-round algorithm removing an edge from G' cannot guarantee that the resulting topology remains connected. This means that the connected dominating set of every k-round algorithm must contain all nodes $V \cup W$ of G'. The approximation ratio of every distributed k-round MCDS algorithm on G' is therefore bounded by

$$\frac{|V \cup W|}{3|V|} \;=\; \frac{n^{1+\Omega(1/k)}}{3n} \;=\; n^{\Omega(1/k)} \;=\; N^{\left(1-\frac{1}{k+\Omega(1)}\right)\Omega(1/k)} \;=\; N^{\Omega(1/k)}.$$

$\square$

5.   LOCAL COMPUTATION: UPPER BOUNDS

This section is devoted to distributed algorithms with similar time-approximation guarantees as given by the lower bounds in Section 3 for the problems introduced in Section 1.2. In Section 5.1. we start with a simple algorithm that specifically targets

the minimum vertex cover problem and asymptotically achieves the trade-off given by the $\Omega(\Delta^{\frac{1-\varepsilon}{k+1}})$ lower bound in Theorem 9. We then describe a generic distributed algorithm to approximate covering and packing linear programs in Section 5.2. In Sections 5.3 and 5.4, we show how an LP solution can be turned into a solution for vertex cover, dominating set, matching, or a related problems by randomized rounding and how a dominating set can be extended to a connected dominating set. Finally, we conclude this section by providing a derandomization result for the distributed solution of fractional problems and with a general discussion on the role of randomization and fractional relaxations in the context of local computations in Section 5.5.

5.1   Distributed Vertex Cover Algorithm

The MVC problem appears to be an ideal starting point for studying distributed approximation algorithms. In particular, MVC does not involve the aspect of *symmetry breaking* which is so crucial in more complex combinatorial problems. The fractional relaxation of the MVC problem asks for a value $x_i \geq 0$ for every node $v_i \in V$ such that the sum of all x_i is minimized and such that for every edge $\{v_i, v_j\}$, $x_i + x_j \geq 1$. A factional solution can be turned into an integer solution by rounding up all nodes with a fractional value at least $1/2$. This increases the approximation ratio by at most a factor of 2. Moreover, any maximal matching is a 2-approximation for MVC and hence, the randomized parallel algorithm for maximal matching by Israeli et al. provides a 2-approximation in time $O(\log n)$ with high probability [Israeli and Itai 1986]. This indicates that the amount of locality required in order to achieve a constant approximation for MVC is bounded by $O(\log n)$. In this section, we present a simple distributed algorithm that places an upper bound on the achievable trade-off between time-complexity and approximation ratio for the minimum vertex cover problem.

Specifically, the algorithm comes with a parameter k, which can be any integer larger than 0. The algorithm's time complexity—and hence its locality—is $O(k)$ and its approximation ratio depends inversely on k. The larger k, the better the achieved global approximation.

Algorithm 1 simultaneously approximates both MVC and its dual problem, the fractional maximum matching (FMM) problem. Let E_i denote the set of incident edges of node v_i. In the FMM problem, each edge $e_j \in E$ is assigned a value y_j such that the sum of all y_j is maximized and such that for every node $v_i \in V$, $\sum_{e_j \in E_i} y_j \leq 1$. The idea of Algorithm 1 is to compute a feasible solution for minimum vertex cover (MVC) and while doing so, distribute dual values y_j among the incident edges of each node. Each node v_i that joins the vertex cover S sets its x_i to 1 and subsequently, the sum of the dual values y_j of incident edges $e_j \in E_i$ is increased by 1 as well. Hence, at the end of each iteration of the main loop, the invariant $\sum_{v_i \in V} x_i = \sum_{e_j \in E} y_j$ holds. We will show that for all nodes v_i, $\sum_{e_j \in E_i} y_j \leq \alpha$ for $\alpha = 3 + \Delta^{1/k}$ and that consequently, dividing all y_j by α yields a feasible solution for FMM. By LP duality, α is an upper bound on the approximation ratio for FMM and MVC. We call an edge *covered* if at least one of its endpoints has joined the vertex cover. The set of uncovered edges incident to a node v_i is denoted by $\tilde{E}_i$, and we define node v_i's *dynamic degree* to be $\tilde{\delta}_i := |\tilde{E}_i|$.

```
1  x_i ← 0;  forall e_j ∈ E_i do y_j ← 0;
2  for ℓ = k − 1, k − 2, . . . , 0 do
3  │    δ̃_i ← |{uncovered edges e ∈ E_i}| = |Ẽ_i|;
4  │    δ̃_i^(1) ← max_{i′∈Γ(v_i)} δ̃_{i′};
5  │    if δ̃_i ≥ (δ̃_i^(1))^{ℓ/(ℓ+1)} then
6  │    │    x_i ← 1;
7  │    │    forall e_j ∈ E_i do y_j ← y_j + 1/δ̃_i
8  │    end
9  │    Y_i ← ∑_{e_j∈E_i} y_j;
10 │    if x_i = 0 and Y_i ≥ 1 then
11 │    │    x_i ← 1;
12 │    │    forall e_j ∈ E_i do y_j ← y_j + 1/Y_i
13 │    end
14 end
15 Y_i ← ∑_{e_j∈E_i} y_j;
16 forall e_j = (v_i, v_{i′}) ∈ E_i do y_j ← y_j/ max{Y_i, Y_{i′}}
```

Algorithm 1: Vertex Cover and Fractional Matching: Code for node $v_i \in V$

The maximum dynamic degree $\tilde{\delta}_{i'}$ among all neighbors $v_{i'}$ of v_i is denoted by $\tilde{\delta}_i^{(1)}$.

In the algorithm, a node joins the vertex cover if it has a large dynamic degree—i.e., many uncovered incident edges—relative to its neighbors. In this sense, it is a faithful distributed implementation of the natural sequential greedy algorithm. Because the running time is limited to k communication rounds, however, the greedy selection step must inherently be parallelized, even at the cost of sub-optimal decisions.

The following lemma bounds the resulting decrease of the maximal dynamic degree in the network.

LEMMA 21. *At the beginning of each iteration, it holds that $\tilde{\delta}_i \leq \Delta^{(\ell+1)/k}$ for every $v_i \in V$.*

PROOF. The proof is by induction over the main loop's iterations. For $\ell = k-1$, the lemma follows from the definition of Δ. For subsequent iterations, we show that all nodes having $\tilde{\delta}_i \geq \Delta^{\ell/k}$ set $x_i := 1$ in Line 6. In the algorithm, all nodes with $\tilde{\delta}_i \geq (\tilde{\delta}_i^{(1)})^{\ell/(\ell+1)}$ set $x_i := 1$. Hence, we have to show that for all v_i, $(\tilde{\delta}_i^{(1)})^{\ell/(\ell+1)} \leq \Delta^{\ell/k}$. By the induction hypothesis, we know that $\tilde{\delta}_i \leq \Delta^{(\ell+1)/k}$ at the beginning of the loop. Since $\tilde{\delta}_i^{(1)}$ represents the dynamic degree $\tilde{\delta}_{i'}$ of some node $v_{i'} \in \Gamma(v_i)$, it holds that $\tilde{\delta}_i^{(1)} \leq \Delta^{(\ell+1)/k}$ for every such v_i and the claim follows because $(\tilde{\delta}_i^{(1)})^{\ell/(\ell+1)} \leq \Delta^{\frac{\ell+1}{k} \cdot \frac{\ell}{\ell+1}}$. □

The next lemma bounds the sum of dual y values in E_i for an arbitrary node $v_i \in V$. For that purpose, we define $Y_i := \sum_{e_j \in E_i} y_j$.

LEMMA 22. *At the end of the algorithm, for all nodes $v_i \in V$,*

$$Y_i \;=\; \sum_{e_j \in E_i} y_j \;\leq\; 3 + \Delta^{1/k}.$$

PROOF. Let Φ_h denote the iteration in which $\ell = h$. We distinguish three cases, depending on whether (or in which line) a node v_i joins the vertex cover. First, consider a node v_i which does not join the vertex cover. Until Φ_0, it holds that $Y_i < 1$ since otherwise, v_i would have set $x_i := 1$ in Line 11 of a previous iteration. In Φ_0, it must hold that $\tilde{\delta}_i = 0$ because all nodes with $\tilde{\delta}_i \geq 1$ set $x_i := 1$ in the last iteration. That is, all adjacent nodes $v_{i'}$ of v_i have set $x_{i'} := 1$ before the last iteration and Y_i does not change anymore. Hence, $Y_i < 1$ for nodes which do not belong to the vertex cover constructed by the algorithm.

Next, consider a node v_i that joins the vertex cover in Line 6 of an arbitrary iteration Φ_ℓ. With the same argument as above, we know that $Y_i < 1$ at the beginning of Φ_ℓ. When v_i sets $x_i := 1$, Y_i increases by one. In the same iteration, however, neighboring nodes $v_{i'} \in \Gamma(v_i)$ may also join the vertex cover and thereby further increase Y_i. By the condition in Line 5, those nodes have a dynamic degree at least $\tilde{\delta}_{i'} \geq (\tilde{\delta}_{i'}^{(1)})^{\ell/(\ell+1)} \geq \tilde{\delta}_i^{\ell/(\ell+1)}$. Further, it holds by Lemma 21 that $\tilde{\delta}_i \leq \Delta^{(\ell+1)/k}$ and therefore

$$\tilde{\delta}_i \cdot \frac{1}{\tilde{\delta}_{i'}} \;\leq\; \frac{\tilde{\delta}_i}{\tilde{\delta}_i^{\ell/(\ell+1)}} \;=\; \tilde{\delta}_i^{1/(\ell+1)} \;\leq\; \Delta^{1/k}.$$

Thus, edges that are simultaneously covered by neighboring nodes may entail an additional increase of Y_i by $\Delta^{1/k}$. Together with v_i's own cost of 1 when joining the vertex cover, the total increase of Y_i in Line 7 of Φ_ℓ is then at most $1 + \Delta^{1/k}$. In Line 7, dual values are distributed among uncovered edges only. Therefore, the only way Y_i can increase in subsequent iterations is when neighboring nodes $x_{i'}$ set $x_{i'} := 1$ in Line 11. The sum of the y_j of all those edges covered only by v_i (note that only these edges are eligible to be increased in this way) is at most 1. In Line 12, these y_j can be at most doubled. Putting everything together, we have $Y_i \leq 3 + \Delta^{1/k}$ for nodes joining the vertex cover in Line 6.

Finally, we study nodes v_i that join the vertex cover in Line 11 of some iteration Φ_ℓ. Again, it holds that $Y_i < 1$ at the outset of Φ_ℓ. Further, using an analogous argument as above, Y_i is increased by at most $\Delta^{1/k}$ due to neighboring nodes joining the vertex cover in Line 6 of Φ_ℓ. Through the joining of v_i, Y_i further increases by no more than 1. Because the y_j are increased proportionally, no further increase of Y_i is possible. Thus, in this case we have $Y_i \leq 2 + \Delta^{1/k}$. □

Based on the bound obtained in Lemma 22, the main theorem follows from LP duality.

THEOREM 23. *In k rounds of communication, Algorithm 1 achieves an approximation ratio of $O(\Delta^{1/k})$. The algorithm is deterministic and requires $O(\log \Delta)$ and $O(\log \Delta / \log \log \Delta)$ rounds for a constant and polylogarithmic approximation, respectively.*

PROOF. We first prove that the algorithm computes feasible solutions for MVC and fractional maximum matching. For MVC, this is clear because in the last

iteration, all nodes having $\tilde{\delta}_i \geq 1$ set $x_i := 1$. The dual y-values form a feasible solution because in Line 16, the y_j of each edge e_j is divided by the larger of the Y_i of the two incident nodes corresponding to e_j, and hence, all constraints of the fractional matching problem are guaranteed to be satisfied. The algorithm's running time is $O(k)$, because every iteration can be implemented with a constant number of communication rounds. As for the approximation ratio, it follows from Lemma 22 that each y_j is divided by at most $\alpha = 3 + \Delta^{1/k}$ and therefore, the objective functions of the primal and the dual problem differ by at most a factor α. By LP duality, α is a bound on the approximation ratio for both problems. Finally, setting $k_1 = \beta \log \Delta$ and $k_2 = \beta \log \Delta / \log \log \Delta$ for an appropriate constant β leads to a constant and polylogarithmic approximation ratio, respectively.   □

Hence, the time-approximation trade-off of Algorithm 1 asymptotically matches the lower bound of Theorem 9. The reason why Algorithm 1 does not achieve a constant or polylogarithmic approximation ratio in a constant number of communication rounds is that it needs to "discretize" the greedy step in order to achieve the necessary parallelism. Whereas the sequential greedy algorithm would select a single node with maximum dynamic degree in each step, a k-local distributed algorithm must inherently take many such decisions in parallel. This discrepancy between Algorithm 1 and the simple sequential greedy algorithm can be seen even in simple networks. Consider for instance the network induced by the complete bipartite graph $K_{m,\sqrt{m}}$. When running the algorithm with parameter $k = 2$, it holds for every node v_i that $\tilde{\delta}_i \geq (\tilde{\delta}_i^{(1)})^{\ell/(\ell+1)}$ in the first iteration ($\ell = 1$) of the loop. Hence, every node will join the vertex cover, resulting in a cover of cardinality $m + \sqrt{m}$. The optimal solution being $\sqrt{m}$, the resulting approximation factor is $\sqrt{m} + 1 = \Delta^{1/2} + 1$.

5.2 Distributed Algorithm for Covering and Packing Linear Programs

We will now describe a generic distributed algorithm to solve covering and packing LPs in the network setting described in Section 1.2. The algorithm is based on a randomized technique to cover a graph with clusters of small diameter described in [Linial and Saks 1993]. The property of covering and packing LPs allows to solve local sub-LPs for all clusters and to combine the local solutions into an approximate global one.

Assume that we are given a primal-dual pair of covering and packing LPs of the canonical form (P) and (D) and the corresponding network graph $G_{\mathrm{LP}} = (V_p \dot\cup V_d, E)$ as defined in Section 1.2. We first describe how the local sub-LPs look like. Let $Y = \{y_1, \ldots, y_{n_d}\}$ be the set of variables of (D). Each local primal-dual sub-LP pair is defined by a subset $S \subseteq V_d$ of the dual nodes V_d and thus by a subset $Y_S \subseteq Y$ of the dual variables. There is a one-to-one correspondence between the inequalities of (P) and the variables of (D). Let P_S be the LP that is obtained from (P) by restricting to the inequalities corresponding to the dual variables in Y_S. The primal variables involved in P_S are exactly the ones held by primal nodes V_p that are neighbors of some node in S. The local LP D_S is the dual LP of P_S. The variables of We first prove crucial basic properties of such a pair of local sub-LPs.

LEMMA 24. *Assume that we are given a pair of LPs P_S and D_S that are constructed from (P) and (D) as described above. If (P) and (D) are both feasible, P_S*

*and D_S are both feasible. Further, any solution to D_S (with dual variables in $Y \setminus Y_S$
set to 0) is a feasible solution of (D).*

PROOF. Clearly P_S is feasible as every feasible solution for (P) directly gives a
feasible solution for P_S (by just ignoring all variables that do not occur in P_S).
Because P_S is a minimization problem and since (P) and (D) are covering and
packing LPs, all coefficients in the objective function (the vector $\underline{c}$) are non-negative,
P_S is also bounded (its objective function is always at least 0). Hence, also the dual
LP D_S must be feasible.

Assume that we are given a feasible solution for D_S which is extended to a solu-
tion for (D) by setting variables in $Y \setminus Y_S$ to 0. Inequalities in (D) that correspond
to columns of variables occuring in P_S are satisfied by the feasibility of the solution
for D_S. In all other inequalities of (D), all variables are set to 0 and thus, feasibility
follows from the fact that $\underline{c} \geq 0$.   □

Note that by construction, a feasible solution for P_S gives a solution for (P)
which satisfies all the inequalities corresponding to variables Y_S and for which the
left-hand sides of all other inequalities are at least 0 because all coefficients and
variables are non-negative. We next show how to obtain local sub-LPs P_S and D_S
that can be solved efficiently.

In [Linial and Saks 1993], Linial and Saks presented a randomized distributed
algorithm for a weak-diameter network decomposition. We use their algorithm to
decompose the linear program into sub-programs which can be solved locally in
the LOCAL model. Assume that we are given a network graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with
$n = |\mathcal{V}|$ nodes. The basic building block of the algorithm in [Linial and Saks 1993]
is a randomized algorithm $\mathcal{LS}(p, R)$ which computes a subset $\mathcal{S} \subseteq \mathcal{V}$ such that each
node $u \in \mathcal{S}$ has a leader $\ell(u) \in \mathcal{V}$ and the following properties hold for arbitrary
parameters $p \in [0, 1]$ and $R \geq 1$:

(1) $\forall u \in \mathcal{S} : d_{\mathcal{G}}(u, \ell(u)) \leq R$, where $d_{\mathcal{G}}(u, v)$ is the shortest path distance between
    two nodes $u, v \in \mathcal{V}$.
(2) $\forall u, v \in \mathcal{S} : \ell(u) \neq \ell(v) \implies (u, v) \notin \mathcal{E}$.
(3) $\mathcal{S}$ can be computed in O(R) rounds.
(4) $\forall u \in \mathcal{V} : \Pr[u \in \mathcal{S}] \geq p(1 - p^R)^{n-1}$.

Hence, Algorithm $\mathcal{LS}(p, R)$ computes a set of clusters of nodes such that nodes
belonging to different clusters are at distance at least 2 and such that every node u
that belongs to some cluster is at distance at most R from its cluster center $\ell(u)$.
Note that Algorithm $\mathcal{LS}(p, R)$ does bound the distance between nodes of the same
cluster in the graph induced by the nodes of the cluster. It merely bounds their
distance in $\mathcal{G}$. The maximal $\mathcal{G}$-distance between any two nodes of a cluster is called
the *weak diameter* of the cluster.

Based on the graph G_{LP}, we define the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ on which we invoke
Algorithm $\mathcal{LS}(p, R)$:

$$\mathcal{V} := V_d, \quad \mathcal{E} := \left\{ \{u, v\} \in \binom{V_s}{2} \,\middle|\, d(u, v) \leq 4 \right\}, \tag{11}$$

where $d(u, v)$ denotes the distance between u and v in G_{LP}. Hence, the nodes of $\mathcal{G}$
are all nodes corresponding to dual variables in G_{LP}. As discussed, there is a one-

to-one correspondence between nodes in V_d and inequalities in the linear program (P). Two nodes $u, v \in V_d$ are connected by an edge in $\mathcal{E}$ iff the corresponding inequalities contain variables that occur together in some inequality. We apply Algorithm $\mathcal{LS}(p, R)$ several times on graph $\mathcal{G}$ to obtain different locally solvable sub-LPs that can then be combined into an approximate solution for (P) and (D). The details are given by Algorithm 2.

---

1 Run ℓ independent instances of $\mathcal{LS}(p, R)$ on $\mathcal{G}$ **in parallel**:

2   yields node sets $\mathcal{S}_1, \ldots, \mathcal{S}_\ell \subseteq \mathcal{V} = V_d$;

3 Solve local LPs $P_{\mathcal{S}_1}, D_{\mathcal{S}_1}, \ldots, P_{\mathcal{S}_\ell}, D_{\mathcal{S}_\ell}$;

4 Interpret as solutions for (P) and (D):

   $x_{1,1}, \ldots, x_{1,n_p}, y_{1,1}, \ldots, y_{1,n_d}, \ldots x_{\ell,1}, \ldots, x_{\ell,n_p}, y_{\ell,1}, \ldots, y_{\ell,n_d}$;

5 **forall** $i \in \{1, \ldots, n_p\}$ **do** $x_i \leftarrow \sum_{t=1}^{\ell} x_{t,i}$;

6 **forall** $i \in \{1, \ldots, n_d\}$ **do** $y_i \leftarrow \sum_{t=1}^{\ell} y_{t,i}$;

7 **forall** $i \in \{1, \ldots, n_p\}$ **do** $x_i \leftarrow x_i / \min_{v_j^d \in \Gamma_{v_i^p}} (A\underline{x})_j / b_j$;

8 **forall** $i \in \{1, \ldots, n_d\}$ **do** $y_i \leftarrow y_i / \ell$;

9 **return** $\underline{x}$ and $\underline{y}$

Algorithm 2: Algorithm for Covering and Packing linear programs with parameters: ℓ, p, and R

We first analyze the time complexity of Algorithm 2.

LEMMA 25. *Algorithm 2 can be executed in* $O(R)$ *rounds. It computes feasible solutions for (P) and (D).*

PROOF. Algorithm 2 consists of the following main steps. First, ℓ independent instances of Algorithm $\mathcal{LS}(p, R)$ are executed. Then, for each collection of clusters resulting from these executions, a local LP is solved and the local LPs are combined to solutions of (P) and (D). Finally, each resulting dual variable is divided by ℓ and each primal variable is divided by an amount that keeps the primal solution feasible.

As the ℓ instances of Algorithm $\mathcal{LS}(p, R)$ are independent, they can be executed in parallel and thus the time complexity for the first step is $O(R)$. Note that since neighbors in $\mathcal{G}$ are at distance at most 4, each round on $\mathcal{G}$ can be simulated in 4 rounds on G_{LP}.

For the second step, consider the set of nodes $\mathcal{S}_i \subseteq V_d$ computed by the ith instance of $\mathcal{LS}(p, R)$. Two nodes that are in different connected components of the sub-graph $\mathcal{G}[\mathcal{S}_i]$ of $\mathcal{G}$ induced by $\mathcal{S}_i$ are at distance at least 5. Hence, the corresponding dual variables and also the primal variables corresponding to their G_{LP}-neighbors in V_p cannot occur together in an inequality of (D) and (P), respectively. Hence, the local sub-LP induced by $\mathcal{S}_i$ can be solved by individually solving the sub-LPs induced by every connected component of $\mathcal{G}[\mathcal{S}_i]$. As every connected component of $\mathcal{G}[\mathcal{S}_i]$ has a leader node that is at distance at most R from all nodes of the connected component, all information from the sub-LP corresponding to $\mathcal{G}[\mathcal{S}_i]$ can be sent to this leader and the sub-LP can be solved there locally. Hence, the second step of the algorithm can also be executed in $O(R)$ rounds.

For the third step, note that the values by which the primal variables x_i are divided can be computed locally (by only exchanging information with direct neighbors in G_{LP}). Finally, the computed dual solution is feasible because it is the average of the dual solutions of all sub-LP and because each dual sub-LP is feasible for (D) by Lemma 24. Line 7 of Algorithm 2 guarantees that the computed primal solution is a feasible solution for (P). □

THEOREM 26. *Let $\varepsilon \in (0, 1)$, $\alpha > 1$, and $\beta > 0$ be parameters. We choose $p = n_d^{-\alpha/R}$ and define $q := p \cdot (1 - n_d \cdot p^R)$. If we choose $\ell \geq \frac{2(1+\beta)}{\varepsilon^2 q} \ln n_d$, Algorithm 2 computes $\frac{1}{q(1-\varepsilon)}$ approximations for (P) and (D) in $\mathrm{O}(R)$ time with probability at least $1 - 1/n_d^\beta$.*

PROOF. The time complexity follows directly from Lemma 25. Because by Lemma 25, the computed solutions for (P) and (D) are both feasible, the approximation ratio of the algorithm is bounded by the ratio of the objective funtions of the solutions for (P) and (D). Both solutions are computed as the sum of the solutions of all local sub-LPs in Lines 5 and 6 of the algorithm that are then divided by appropriate factors in Lines 7 and 8. By LP duality (of the sub-LPs), we have $\underline{c}^T \underline{x} = \underline{b}^T \underline{y}$ after Line 6. Hence, the approximation ratio is upper bounded be the ratio between the factor ℓ by which the dual variables are divided and the minimal value by which the primal variables are divided. The approximation is therefore upper bounded by

$$\frac{\ell}{\min_{v_i^p \in V_p} \min_{v_j^d \in \Gamma_{v_i^p}} (A\underline{x})_j / b_j} = \frac{\ell}{\min_{v_j^d \in V_d} (A\underline{x})_j / b_j} \tag{12}$$

for $\underline{x}$ after Line 6. To obtain a lower bound on the value of the above equation, assume that for every $v_j^d \in V_d$, the number of local sub-LPs $P_{\mathcal{S}_t}$ for which $(A\underline{x_t})_j \geq b_j$ is at least $\ell' \leq \ell$. Hence, ℓ' is a lower bound on the number of times each inequality of (P) is satisfied, combined over all sub-LPs. Because $b_j \geq 0$ for all j and because all coefficients of A and the variables x are non-negative, we then have $(A\underline{x})_j / b_j \geq \ell'$ for all j. By Equation (12). it then following that the computed solutions for (P) and (D) are at most by a factor ℓ/ℓ' worse than the optimal solutions.

We get a bound on the minimum number of times each inequality of (P) is satisfied by a local sub-LP by using the properties of Algorithm $\mathcal{LS}(p, R)$ and a chernoff bound. From [Linial and Saks 1993], we have that for each $t \in \{1, \ldots, \ell\}$ and $v_i^d \in V_d$, the probability that $v_i^d \in \mathcal{S}_t$ is at least

$$p(1-p^R)^{n_d-1} \overset{p^R}{=} \frac{1}{n_d^{\alpha/R}} \cdot \left(1 - \frac{1}{n_d^\alpha}\right)^{n_d-1} \overset{(\alpha>1)}{\geq} \frac{1}{n_d^{\alpha/R}} \cdot \left(1 - \frac{1}{n_d^{\alpha-1}}\right) = q.$$

Therefore, for every $v_j^d \in V_d$, the probability P_j that j^{th} inequality of (P) is satisfied less than $(1-\varepsilon)q\ell$ times is at most

$$P_j < e^{-\frac{\varepsilon^2}{2} q\ell} \leq e^{-(1+\beta)\ln n_d} = \frac{1}{n_d} \cdot \frac{1}{n^\beta}. \tag{13}$$

The theorem now follows by a union bound over all n_d inequalities of (P). □

COROLLARY 27. *In k rounds, Algorithm 2 with high probability computes an* $O(n^{c/k})$*-approximation for covering and packing LPs for some constant $c > 0$. An* $(1 + \varepsilon)$*-approximation can be computed in time* $O(\log(n)/\varepsilon)$.

5.3   Randomized Rounding

We next show how to use the algorithm of the last section to solve the MDS problem or another combinatorial covering or packing problem. Hence, we show how to turn a fractional covering or packing solution into an integer one by a distributed rounding algorithm. In particular, we give an algorithm for integer covering and packing problems of the forms

$$
\begin{array}{llll}
\min & \underline{c}^{\mathrm{T}}\underline{x}' & \qquad & \min & \underline{b}^{\mathrm{T}}\underline{y}' \\
\text{s. t.} & A \cdot \underline{x}' \geq \underline{b} & \qquad (\mathrm{P}_I) & \text{s. t.} & A^{\mathrm{T}} \cdot \underline{y}' \leq \underline{c} \qquad (\mathrm{D}_I) \\
& x'_i \in \mathbb{N}. & \qquad & & y'_i \in \mathbb{N}.
\end{array}
$$

with matrix elements $a_{ij} \in \{0, 1\}$. LPs (P) and (D) are the fractional relaxations of (P_I) and (D_I). Not that we denote the solution vectors for the integer program by $\underline{x}'$ and $\underline{y}'$ whereas the solution vectors for the corresponding LPs are called $\underline{x}$ and $\underline{y}$.

We start with covering problems (problems of the form of (P)). Because the a_{ij} and the x_i are restricted to integer values, w.l.o.g. we can round up all b_j to the next integer value. After solving/approximating the LP, each primal node v_i^p executes Algorithm 3. The value of the parameter λ will be determined later.

---

1 **if** $x_i \geq 1(\lambda \ln \Delta_p)$ **then**
2 $\quad \big|\quad x'_i \leftarrow \lceil x_i \rceil$
3 **else**
4 $\quad \big|\quad p_i \leftarrow x_i \cdot \lambda \ln \Delta_p$;
5 $\quad \big|\quad x'_i \leftarrow 1$ with probability p_i and $x'_i \leftarrow 0$ otherwise
6 **end**

Algorithm 3: Distributed Randomized Rouding: Covering Problems

The expected value of the objective function is $\mathrm{E}[\underline{c}^{\mathrm{T}}\underline{x}'] \leq \lambda \ln \Delta_p \cdot \underline{c}^{\mathrm{T}}\underline{x}$. Yet regardless of how we choose λ, there remains a non-zero probability that the obtained integer solution is not feasible. To overcome this, we have to increase some of the x'_i. Assume that the jth constraint is not satisfied. Let $\underline{a}_j$ be the row vector representing the jth row of the matrix A and let $b'_j := b_j - \underline{a}_i x'$ be the missing weight to make the jth row feasible. Further, let $i_{j_{\min}}$ be the index of the minimum c_i for which $a_{ji} = 1$. We set $x'_{i_{j_{\min}}} := x'_{i_{j_{\min}}} + b'_j$. Applied to all non-satisfied primal constraints, this gives a feasible solution for the considered integer covering problem.

THEOREM 28. *Consider an integer covering problem (P_I) with $a_{ij} = \{0, 1\}$ and $b_j \in \mathbb{N}$. Furthermore, let $\underline{x}$ be an α-approximate solution for the LP relaxation (P) of (P_I). The above described algorithm computes an $O(\alpha \log \Delta_p)$-approximation $\underline{x}'$ for (P_I) in a constant number of rounds.*

PROOF. As stated above, the expected approximation ratio of the first part of

the algorithm is $\lambda \ln \Delta_p$. In order to bound the additional weight of the second part, where $x'_{i_{j_{\min}}}$ is increased by b'_j, we define dual variables $\tilde{y}_j := b'_j c_{i_{j_{\min}}} / b_j$. For each unsatisfied primal constraint, the increase $c_{i_{j_{\min}}} b'_j$ of the primal objective function is equal to the increase $b_j \tilde{y}_j$ of the dual objective function. If the jth constraint is not satisfied, we have $b'_j \geq 1$. Therefore, $\mathrm{E}[\tilde{y}_j] \leq q_j c_{i_{j_{\min}}}$, where q_j is the probability that the jth primal inequality is not fulfilled.

In order to get an upper bound on the probability q_j, we have to look at the sum of the x'_i before the randomized rounding step in Line 5 of the algorithm. Let $\beta_j := b_i - \underline{a_i} \underline{x}'$ be the missing weight in row j before Line 5. Because the x-values correspond to a feasible solution for the LP, the sum of the p_i involved in row j is at least $\beta_j \lambda \ln \Delta_p$. For the following analysis, we assume that $\ln \Delta_p \geq 1$. If $\ln \Delta_p < 1$, applying only the last step of the described algorithm gives a simple distributed 2-approximation for the considered integer program. Using a Chernoff bound, we can bound q_j as

$$q_j < e^{-\frac{1}{2}\beta_j \lambda \ln \Delta_p (1 - \frac{1}{\lambda \ln \Delta_p})^2} \leq \left(\frac{1}{\Delta_p} \right)^{\frac{1}{2}\lambda(1-\frac{1}{\lambda})^2} \leq \frac{1}{\Delta_p}.$$

In the second inequality, we use that $\beta_j \geq 1$. For the last inequality, we have to choose λ such that $\lambda(1 - 1/\lambda)^2/2 \geq 1$ (i.e., $\lambda \geq 2 + \sqrt{3}$). Thus, the expected value of $\tilde{y}_j$ is $\mathrm{E}[\tilde{y}_j] \leq c_{i_{j_{\min}}}/\Delta_p$. Hence, by definition of $c_{i_{j_{\min}}}$, in expectation the $\tilde{y}$-values form a feasible solution for (D). Therefore, the expected increase of the objective function $\underline{c}^{\mathrm{T}} \underline{x}'$ in the last step after the randomized rounding is upper-bounded by the objective function of an optimal solution for (P). $\qquad \square$

Combining Algorithms 2 and 3, we obtain an $\mathrm{O}(\log \Delta)$-approximation for MDS in $\mathrm{O}(\log n)$ rounds.

We now turn our attention to integer packing problems. We have an integer program of the form of (D_I) where all $a_{ij} \in \{0, 1\}$ and where $\underline{y}' \in \mathbb{N}^n$. We can w.l.o.g. assume that the c_j are integers because rounding down each c_j to the next integer has no influence on the feasible region. Each dual node v_i^d applies Algorithm 4.

---

1 **if** $y_i \geq 1$ **then**
2 $\quad$ $y'_i \leftarrow \lfloor y_i \rfloor$
3 **else**
4 $\quad$ $p_i \leftarrow 1/(2e\Delta_d)$;
5 $\quad$ $y'_i \leftarrow 1$ with probability p_i and $y'_i \leftarrow 0$ otherwise
6 **end**
7 **if** $y'_i \in$ 'non-satisfied constraint' **then**
8 $\quad$ $y'_i \leftarrow \lfloor y_i \rfloor$
9 **end**

Algorithm 4: Distributed Randomized Rouding: Packing Problems

Clearly, the yields a feasible solution for the problem. The approximation ratio of the algorithm is given by the next theorem.

THEOREM 29. *Let (D_I) be an integer covering problem with $a_{ij} = \{0, 1\}$ and $c_j \in \mathbb{N}$. Furthermore, let $\underline{y}$ be an α-approximate solution for the LP relaxation of*

(D_I). Algorithm 4 computes an $\mathrm{O}(\alpha\Delta_d)$-approximation $\underline{y}'$ for (D_I) in a constant number of rounds.

PROOF. After Line 6, the expected value of the objective function is $\underline{b}^{\mathrm{T}}\underline{y}' \geq \underline{b}^{\mathrm{T}}\underline{y}/(2e\Delta_d)$. We will now show that a non-zero y'_i stays non-zero with constant probability in Line 8. Let q_j be the probability that the jth constraint of the integer program is not satisfied given that y'_i has been set to 1 in Line 5. For convenience, we define $Y'_j := \sum_i a_{ij} y'_i$. If $c_j \geq 2$, we apply a Chernoff bound to obtain

$$q_j \;=\; \Pr[Y'_j > c_j \,\big|\, y'_i = 1] \;\leq\; \Pr[Y'_j > c_j - 1]$$
$$<\; \left(\frac{e^{e\Delta_d - 1}}{(e\Delta_c)^{e\Delta_d}}\right)^{c_j/(2e\Delta_d)} \;<\; \frac{1}{\Delta_d}.$$

If $c_j = 1$, we get

$$q_j \;\leq\; 1 - \Pr[Y'_j = 0] \;=\; 1 - \prod_{v_i^d \in \Gamma(v_j^p)} (1 - p_i)$$
$$\leq\; 1 - \left(1 - \frac{1}{2e\Delta_d}\right) \;=\; \frac{1}{2e\Delta_d}.$$

The probability that all dual constraints containing y'_i are satisfied is lower-bounded by the product of the probabilities for each constraint [Srinivasan 1995]. Therefore, under the natural assumption that $\Delta_d \geq 2$:

$$\Pr[y'_i = 1 \text{ after Line 8}] \geq \left(1 - \frac{1}{\Delta_d}\right)^{\Delta_d} \geq \frac{1}{4}.$$

Thus the expected value of the objective function of the integer program (D_I) is

$$\mathrm{E}[\underline{b}^{\mathrm{T}}\underline{y}'] \;\geq\; 8e\Delta_d \cdot \underline{b}^{\mathrm{T}}\underline{y}.$$

$\square$

5.4 Connecting a Dominating Set

An important applications of dominating sets in networks is to obtain clusterings in ad hoc or sensor networks. In particular, clustering helps to improve information dissemination and routing algorithms in such networks. However, for this purpose, one usually needs clusters to be connected to each other and thus a connected dominating set as underlying structure. Lemma 19 in Section 4.5 shows that every dominating set D can be extended to a connected dominating set D' of size $|D'| < 3|D|$. In the following, we described a simple distributed strategy to convert any dominating set into a connected dominating set that is only slightly larger. A similar strategy is also used in [Dubhashi et al. 2003].

Assume that we are given a dominating set D of the network graph G. As in the proof of Lemma 19, we define a graph G_D as follows. The node set of G_D is D and there is an edge between $u, v \in D$ iff their distance in G is at most 3. We have seen that G_D is connected and thus, any spanning tree of G_D induces a connected dominating set of size $\mathrm{O}(D)$. Unfortunately, for a local, distributed algorithm, it is not possible to compute a spanning tree of G_D. Nevertheless, a similar approach

also works for distributed algorithms. Instead of computing a spanning tree of G_D, it is sufficient to compute any sparse spanning subgraph of G_D. If the number of edges of the subgraph of G_D is linear in the number of nodes $|D|$ of G_D, we obtain a connected dominating set S' which is only by a constant factor larger than D.

We therefore need to solve the following problem. Given a graph $G = (V, E)$ with $|V| = n$, we want to compute a spanning subgraph G' of G with a minimal number of edges. For an arbitrary $k \geq 1$, the following Algorithm 5 shows how to compute such a spanning subgraph in k rounds. For the algorithm, we assume that all edges $e = (u, v)$ of G have a unique weight w_e and that there is a total order on all edge weights. If there are no natural edge weights, a weight for (u, v) can for example be constructed by taking the ordered pair of the IDs of the nodes u and v. Two weights can be compared using lexicographic order.

---

1 $G' \leftarrow G$;
2 forall $u \in V$ **do** u **collects** complete k-neighborhood;
3 forall $e \in E$ **do**
4   **if** weight w_e of e is largest in any cycle of length $\leq 2k$ **then**
5   remove e from G
6   **end**
7 end

Algorithm 5: Computing a sparse connected subgraph

The following lemma shows that Algorithm 5 indeed computes a sparse connected subgraph G' of G.

LEMMA 30. *For every n-node connected graph $G = (V, E)$ and every k, Algorithm 5 computes a spanning subgraph $G' = (V, E')$ of G for which the number of edges is bounded by $|E'| \leq n^{1+O(1/k)}$.*

PROOF. We first prove that the produced G' is connected. For the sake of contradiction, assume that G' is not connected. Then, there must be a cut (S, T) with $S \subseteq V$, $T = V \setminus S$, and $S, T \neq \emptyset$ such that $S \times T \cap E' = \emptyset$. However, since G is connected, there must be an edge $e \in S \times T \cap E$ crossing the given cut. Let e be the edge with minimal weight among all edges crossing the cut. Edge e can only be removed by Algorithm 5 if it has the largest weight of all edges in some cycle. However, all cycles containing e also contain another edge e' crossing the (S, T)-cut. By definition of e, $w_{e'} > w_e$ and therefore, e is not deleted by the algorithm.

Let us now look at the number of edges of G'. Because in every cycle of length at most $2k$ at least one edge is removed by Algorithm 5, G' has girth $g(G') \geq 2k + 1$. It is well-known that therefore, G' has at most $|V|^{1+O(1/k)}$ edges (see e.g. [Bollobas 1978]). □

We can therefore formulate a k-round MCDS algorithm consisting of the following three phases. First, a fractional dominating set is computed using Algorithm 2. Second, we use the randomized rounding scheme given by Algorithm 3 to obtain a dominating set D. Finally, Algorithm 5 is applied to G_D. For each edge (u, v) of the produced spanning subgraph of G_D, we add the nodes (at most 2) of a shortest path connecting u and v in G to D. Note that a k-round algorithm on G_D needs at most $3k$ rounds when executed on G. The achieved approximation ratio is given

by the following theorem.

THEOREM 31. *In* $\mathrm{O}(k)$ *rounds, the above described MCDS algorithm computes a connected dominating set of expected size*

$$\mathrm{O}\Big(\mathrm{CDS_{OPT}} \cdot n^{\mathrm{O}(1/k)} \cdot \log \Delta\Big).$$

PROOF. Given the dominating set D, by Lemma 30, the number of nodes of the connected dominating set D' can be bounded by

$$|D'| \ \leq \ 3|D|^{1+\mathrm{O}(1/k)} \leq \ 3|D| n^{\mathrm{O}(1/k)}$$

and therefore

$$\mathrm{E}[|D'|] \ \leq \ 3\,\mathrm{E}[|D|] n^{\mathrm{O}(1/k)}. \tag{14}$$

Using Theorems 26 and 28, it follows that the expected size of the dominating set D is

$$\mathrm{E}[|D|] \ \in \ \mathrm{O}\left(\mathrm{DS_{OPT}} n^{\mathrm{O}(1/k)} \log \Delta\right).$$

Plugging this into Inequality (14) completes the proof. □

5.5   Role of Randomization and Distributed Derandomization

Randomization plays a crucial role in distributed algorithms. For many problems such as computing a MIS, there are simple and efficient randomized algorithms. For the same problems, the best deterministic algorithms are much more complicated and usually significantly slower. The most important use of randomization in distributed algorithms is breaking symmetries. We have seen that in certain cases, LP relaxation can be used to "avoid" symmetry breaking. The question is whether the use of randomness can also be avoided in such cases? In the following, we show that this indeed is the case, i.e., we show that in the LOCAL model *any distributed randomized algorithm for solving a linear program can be derandomized.*

Assume that we are given a randomized distributed k-round algorithm $\mathcal{A}$ which computes a solution for an arbitrary linear program P. We assume that $\mathcal{A}$ explicitly solves P such that w.l.o.g. we can assume that each variable x_i of P is associated with a node v which computes x_i. We also assume that $\mathcal{A}$ always terminates with a feasible solution. The following theorem shows that $\mathcal{A}$ can be derandomized.

THEOREM 32. *Algorithm $\mathcal{A}$ can be transformed into a deterministic k-round algorithm $\mathcal{A}'$ for solving P. The objective value of the solution produced by $\mathcal{A}'$ is equal to the expected objective value of the solution computed by $\mathcal{A}$.*

PROOF. We first show that for the node computing the value of variable x_i, it is possible to deterministically compute the expected value $\mathrm{E}[x_i]$. We have seen that in the LOCAL model every deterministic k-round algorithm can be formulated as follows. First, every node collects all information up to distance k. Then, each node computes its output based on this information. The same technique can also be applied for randomized algorithms. First, every node computes all its random bits. Collecting the k-neighborhood then also includes collecting the random bits of all nodes in the k-neighborhood. However, instead of computing x_i as a function of the collected information (including the random bits), we can also compute $\mathrm{E}[x_i]$ without even knowing the random bits.

In algorithm $\mathcal{A}'$, the value of each variable is now set to the computed expected value. By linearity of expectation, the objective value of $\mathcal{A}'$'s solution is equal to the expected objective value of the solution of $\mathcal{A}$. It remains to prove that the computed solution is feasible. For the sake of contradiction, assume that this is not the case. Then, there must be an inequality of P which is not satisfied. By linearity of expectation, this implies that this inequality is not satisfied in expectation for the randomized algorithm $\mathcal{A}$. Therefore, there is a non-zero probability that $\mathcal{A}$ does not fulfill the given inequality, a contradiction to the assumption that $\mathcal{A}$ always computes a feasible solution.   □

Theorem 32 implies that the algorithm of Section 5.2 could be derandomized to deterministically compute an $(1+\varepsilon)$-approximation for (P) and (D) in $O(\log(n)/\varepsilon)$ rounds. It also means that in principle every distributed dominating set algorithm (e.g. [Jia et al. 2001; Rajagopalan and Vazirani 1998] could be turned into a deterministic fractional dominating set algorithm with the same approximation ratio. Hence, when solving integer linear programs in the LOCAL model, randomization is only needed to break symmetries. Note that this is really a property of the LOCAL model and only true as long as there is no bound on message sizes and local computations. The technique described in Theorem 32 can drastically increase message sizes and local computations of a randomized distributed algorithm.

6.   CONCLUSIONS & FUTURE WORK

Lower Bounds: Distributed systems is an area in computer science with a strong lower bound culture. This is no coincidence as lower bounds can be proved using *indistinguishability* arguments, i.e. that some nodes in the system cannot distinguish two configurations, and therefore must make "wrong" decisions.

Indistinguishability arguments have also been used in locality. In his seminal paper, Linial proved an $\Omega(\log^* n)$ lower bound for coloring the *drosophila melanogaster* of distributed computing, the ring topology [Linial 1992]. However, one cannot prove local inapproximability bounds on the ring or other highly symmetric topologies, as they allow for straight-forward purely local, constant approximation solutions. Take for instance the minimum vertex cover problem (MVC): In any δ-regular graph, the algorithm which includes all nodes in the vertex cover is already a 2-approximation. Each node will cover at most δ edges, the graph has $n\delta/2$ edges, and therefore at least $n/2$ nodes need to be in a vertex cover.

On the other extreme, asymmetric graphs often enjoy constant-time algorithms, too. In a tree, choosing all inner nodes yields a 2-approximation MVC. Similar trade-offs exist for node degrees. If the maximum node degree is low (constant), we can tolerate to choose all nodes, and have by definition a good (constant) approximation. If there are nodes with high degree, the diameter of the graph will be small, and a few communication rounds suffice to inform all nodes of the entire graph. Thus, our lower bound construction in Section 3 requires the construction of a "fractal", self-recursive graph that is neither too symmetric nor too asymmetric, and has a variety of node degrees! To the best of our knowledge, not many graphs with these "non-properties" are known in computer science, where symmetry and regularity are often the key to a solution.

Upper Bounds: It is interesting to compare the lower and upper bounds for

the various problems. The MVC algorithm presented in Section 5.1 achieves an $O(\Delta^{1/k})$ approximation in k communication rounds, and hence, the lower and upper bounds achieved in Theorems 9 and 23 are *tight*. In particular, any distributed algorithm requires at least $\Omega(\log \Delta)$-hop and $\Omega(\log \Delta/\log \log \Delta)$-hop neighborhood information in order to achieve a constant and polylogarithmic approximation ratio to the MVC problem, respectively, which is exactly what our algorithm achieves.

Our bounds are not equally tight when expressed as a function of n, rather than Δ. In particular, the remaining gap between our upper and lower bounds can be as large as $\Theta(\sqrt{\log n/\log \log n})$ for polylogarithmic and $\Theta(\sqrt{\log n})$ for constant approximations, respectively. The additional square-root in the lower bounds when formulated as a function of n follows inevitably from the high-girth construction of G_k: In order to derive a lower-bound graph as described in Sections 3.1 and 3.2, there must be many "bad" nodes that have the same view as a few neighboring "good" nodes. If each bad node has a degree of δ_{bad} (in G_k, this degree is $\delta_{bad} \in \Theta(n^{1/k})$) and if we want to have girth at least k, the graph must contain at least $n \geq \delta_{bad}^k$ nodes. Taking all good nodes and applying Algorithm 1 of Section 5.1 to the set of bad nodes, we obtain an approximation ratio of $\alpha \in O(\delta_{bad}^{1/k})$ in k communication rounds. Combining this with the bound on the number of nodes in the graph, it follows that there is no hope for a better lower bound than $\Omega(n^{1/k^2})$ with this technique. From this it follows that if we want to improve the lower bound (i.e., by getting rid of its square-root), we either need an entirely different proof technique, or we must handle graphs with low girth in which nodes do not see trees in their k-hop neighborhood, which would necessitate arguing about views containing cycles.

Future Work: We believe that the study of local computation and local approximation is relevant far beyond distributed computing, and there remain numerous directions for future research. Clearly, it is interesting to study the locality of other network coordination problems that appear to be pseudo-local, including for example the *maximum domatic partition problem* [Feige et al. 2003], the *maximum unique coverage problem* [Demaine et al. 2006], or—of course—various coloring problems [Kuhn and Wattenhofer 2006].

Beyond these specific open problems, the most intriguing *distant goal* of this line of research is to divide distributed problems into *complexity classes* according to the problems' local nature. The existence of *locality-preserving reductions* and the fact that several of the problems discussed in this paper exhibit similar characteristics with regard to local computability/approximability raises the hope for something like a *locality hierarchy* of combinatorial optimization problems. It would be particularly interesting to establish ties between such a distributed hierarchy of complexity classes and the classic complexity classes originating in the Turing model of computation [Lenzen et al. 2008].

Besides classifying computational problems, studying local computation may also help in gaining a more profound understanding of the relative strengths of the underlying *network graph models* themselves. It was shown in [Schneider and Wattenhofer 2008], for example, that a MIS can be computed in unit disk graphs (as well as generalizations thereof) in time $O(\log^* n)$, which—in view of Linial's lower bound on the ring—is asymptotically optimal. Hence, in terms of local computabil-

ity, the vast family of unit disk graphs are equally hard as a simple ring network. On the other hand, our lower bounds prove that general graphs are strictly harder, thus separating these network topologies.

REFERENCES

AFEK, Y., KUTTEN, S., AND YUNG, M. 1990. Memory-efficient self stabilizing protocols for general networks. In *WDAG*, J. van Leeuwen and N. Santoro, Eds. Lecture Notes in Computer Science, vol. 486. Springer, 15–28.

ALON, N., BABAI, L., AND ITAI, A. 1986. A Fast and Simple Randomized Parallel Algorithm for the Maximal Independent Set Problem. *Journal of Algorithms 7,* 4, 567–583.

AWERBUCH, B. AND SIPSER, M. 1988. Dynamic networks are as fast as static networks. 206–219.

AWERBUCH, B. AND VARGHESE, G. 1991. Distributed program checking: a paradigm for building self-stabilizing distributed protocols. 258–267.

BARTAL, Y., BYERS, J. W., AND RAZ, D. 1997. Global Optimization Using Local Information with Applications to Flow Control. In *Proc. of the 38th Symposium on Foundations of Computer Science (FOCS)*. 303–312.

BOLLOBAS, B. 1978. *Extremal Graph Theory*. Academic Press.

COLE, R. AND VISHKIN, U. 1986. Deterministic Coin Tossing with Applications to Optimal Parallel List Ranking. *Information and Control 70,* 1, 32–53.

CZYGRINOW, A., HAŃĆKOWIAK, M., AND WAWRZYNIAK, W. 2008. Fast distributed approximations in planar graphs. Vol. 5218. 78–92.

DEMAINE, E. D., FEIGE, U., HAJIAGHAYI, M. T., AND SALAVATIPOUR, M. R. 2006. Combination Can Be Hard: Approximability of the Unique Coverage Problem. In *Proc. of the 17th ACM-SIAM Symposium on Discrete Algorithm (SODA)*. 162–171.

DIJKSTRA, E. W. 1973. Self-stabilization in spite of distributed control. Manuscript EWD391.

DIJKSTRA, E. W. 1974. Self-stabilizing systems in spite of distributed control. *Commun. ACM 17,* 11, 643–644.

DUBHASHI, D., MEI, A., PANCONESI, A., RADHAKRISHNAN, J., AND SRINIVASAN, A. 2003. Fast Distributed Algorithms for (Weakly) Connected Dominating Sets and Linear-Size Skeletons. In *Proc. of the ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 717–724.

ELKIN, M. 2004a. An Unconditional Lower Bound on the Hardness of Approximation of Distributed Minimum Spanning Tree Problem. In *Proc. of the 36th ACM Symposium on Theory of Computing (STOC)*. 331–340.

ELKIN, M. 2004b. Distributed Approximation - A Survey. *ACM SIGACT News - Distributed Computing Column 35,* 4.

FEIGE, U., HALLDÓRSSON, M. M., KORTSARZ, G., AND SRINIVASAN, A. 2003. Approximating the Domatic Number. *SIAM Journal on Computing 32,* 1, 172–195.

FICH, F. AND RUPPERT, E. 2003. Hundreds of Impossibility Results for Distributed Computing. *Distributed Computing 16,* 2-3, 121–163.

FISCHER, M. J., LYNCH, N. A., AND PATERSON, M. S. 1985. Impossibility of Distributed Consensus With One Faulty Process. *Journal of the ACM 32,* 2, 374–382.

FLEISCHER, L. 2000. Approximating Fractional Multicommodity Flow Independent of the Number of Commodities. *SIAM Journal on Discrete Mathematics 13,* 4, 505–520.

GOLDSTEIN, S. C., CAMPBELL, J. D., AND MOWRY, T. C. 2005. Programmable matter. *Computer 38,* 6, 99–101.

ISRAELI, A. AND ITAI, A. 1986. A Fast and Simple Randomized Parallel Algorithm for Maximal Matching. *Information Processing Letters 22,* 77–80.

JIA, L., RAJARAMAN, R., AND SUEL, R. 2001. An Efficient Distributed Algorithm for Constructing Small Dominating Sets. In *Proc. of the 20th ACM Symposium on Principles of Distributed Computing (PODC)*. 33–42.

KUHN, F., MOSCIBRODA, T., AND WATTENHOFER, R. 2004. What Cannot be Computed Locally! In *Proc. of the 23rd ACM Symposium on the Principles of Distributed Computing (PODC)*. 300–309.

KUHN, F., MOSCIBRODA, T., AND WATTENHOFER, R. 2006. The Price of Being Near-Sighted. In *Proc. of the 17th ACM-SIAM Symposium on Discrete Algorithms (SODA)*.

KUHN, F. AND WATTENHOFER, R. 2003. Constant-Time Distributed Dominating Set Approximation. In *Proc. of the 22nd Annual ACM Symp. on Principles of Distributed Computing (PODC)*. 25–32.

KUHN, F. AND WATTENHOFER, R. 2006. On the Complexity of Distributed Graph Coloring. In *Proc. of the 25th ACM Symposium on Principles of Distributed Computing (PODC)*.

LAMPORT, L., SHOSTAK, R., AND PEASE, M. 1982. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst. 4,* 3, 382–401.

LAZEBNIK, F. AND USTIMENKO, V. A. 1995. Explicit Construction of Graphs with an Arbitrary Large Girth and of Large Size. *Discrete Applied Mathematics 60,* 1-3, 275–284.

LAZEBNIK, F., USTIMENKO, V. A., AND WOLDAR, A. J. 1995. A New Series of Dense Graphs of High Girth. *Bulletin of the American Mathematical Society (N.S.) 32,* 1, 73–79.

LENZEN, C., OSWALD, Y. A., AND WATTENHOFER, R. 2008. What can be approximated locally? In *20th ACM Symposium on Parallelism in Algorithms and Architecture (SPAA), Munich, Germany*.

LENZEN, C., SUOMELA, J., AND WATTENHOFER, R. 2009. Local Algorithms: Self-Stabilization on Speed. In *11th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS), Lyon, France*.

LENZEN, C. AND WATTENHOFER, R. 2008. Leveraging Linial's locality limit. Vol. 5218. 394–407.

LINIAL, N. 1992. Locality in Distributed Graph Algorithms. *SIAM Journal on Computing 21,* 1, 193–201.

LINIAL, N. AND SAKS, M. 1993. Low Diameter Graph Decompositions. *Combinatorica 13,* 4, 441–454.

LUBY, M. 1986. A Simple Parallel Algorithm for the Maximal Independent Set Problem. *SIAM Journal on Computing 15*, 1036–1053.

MALEWICZ, G., AUSTERN, M. H., BIK, A. J., DEHNERT, J. C., HORN, I., LEISER, N., AND CZAJKOWSKI, G. 2010. Pregel: A System for Large-Scale Graph Processing. In *Proceedings of the International Conference on Management of Data (SIGMOD)*.

NAOR, M. AND STOCKMEYER, L. 1995. What Can Be Computed Locally? *SIAM Journal on Computing 24,* 6, 1259–1277.

NGUYEN, H. N. AND ONAK, K. 2008. Constant-time approximation algorithms via local improvements. 327–336.

PAPADIMITRIOU, C. H. AND YANNAKAKIS, M. 1991. On the Value of Information in Distributed Decision Making. In *Proc. of the 10th ACM Symposium on Principles of Distributed Computing (PODC)*. 61–64.

PAPADIMITRIOU, C. H. AND YANNAKAKIS, M. 1993. Linear Programming Without the Matrix. In *Proc. of the 25th ACM Symposium on Theory of Computing (STOC)*. 121–129.

PARNAS, M. AND RON, D. 2007. Approximating the minimum vertex cover in sublinear time and a connection to distributed algorithms. *Theor. Comput. Sci. 381,* 1-3, 183–196.

PELEG, D. 2000. *Distributed Computing: A Locality-Sensitive Approach.* SIAM Monographs on Discrete Mathematics and Applications.

PLOTKIN, S., SHMOYS, D., AND TARDOS, E. 1995. Fast Approximation Algorithms for Fractional Packing and Covering Problems. *Mathematics of Operations Research 20*, 257–301.

RAJAGOPALAN, S. AND VAZIRANI, V. 1998. Primal-Dual RNC Approximation Algorithms for Set Cover and Covering Integer Programs. *SIAM Journal on Computing 28*, 525–540.

SCHNEIDER, J. AND WATTENHOFER, R. 2008. A Log-Star Distributed Maximal Independent Set Algorithm for Growth-Bounded Graphs. In *27th ACM Symposium on Principles of Distributed Computing (PODC), Toronto, Canada*.

SCHNEIDER, J. AND WATTENHOFER, R. 2009. Bounds On Contention Management Algorithms. In *20th International Symposium on Algorithms and Computation (ISAAC), Honolulu, USA*.

SRINIVASAN, A. 1995. Improved approximations of packing and covering problems. In *Proc. of the 27th ACM Symposium on Theory of Computing (STOC)*. 268–276.

STERLING, A. 2009. Memory consistency conditions for self-assembly programming. *CoRR abs/0909.2704.*

SUOMELA, J. 2009. Survey of local algorithms. Manuscript.

WATTENHOFER, M. AND WATTENHOFER, R. 2004. Distributed Weighted Matching. In *Proc. of the 18th Annual Conference on Distributed Computing (DISC).* 335–348.

YOUNG, N. 2001. Sequential and Parallel Algorithms for Mixed Packing and Covering. In *Proc. of the 42nd IEEE Symposium on Foundations of Computer Science (FOCS).* 538–546.