Fast Distributed Algorithms for (Weakly) Connected Dominating Sets and Linear-Size Skeletons

Devdatt Dubhashi* Alessandro Mei† Alessandro Panconesi‡
Jaikumar Radhakrishnan§ Aravind Srinivasan¶

Abstract

Motivated by routing issues in ad hoc networks, we present polylogarithmic-time distributed algorithms for two problems. Given a network, we first show how to compute connected and weakly connected dominating sets whose size is at most $O(\log\Delta)$ times optimal, Δ being the maximum degree of the input network. This is best-possible if $\mathrm{NP} \not\subseteq \mathrm{DTIME}[n^{O(\log\log n)}]$ and if the processors are limited to polynomial-time computation. We then show how to construct dominating sets which satisfy the above properties, as well as the "low stretch" property that any two adjacent nodes in the network have their dominators at a distance of at most $O(\log n)$ in the network. (Given a dominating set S, a dominator of a vertex u is any $v \in S$ such that the distance between u and v is at most one.) We also show our time bounds to be essentially optimal.

Key words and phrases. Ad hoc networks, dominating sets, distributed algorithms.

1 Introduction

We present fast distributed algorithms for computing connected and weakly connected dominating sets with good "low stretch" properties, in a given distributed network. Our motivation comes from the study of ad hoc wireless networks. A crucial way in which these differ from current cellular networks is that they do not have a separate routing infrastructure such as a system of base-stations; the mobiles have to conduct their own communication through routing. In these networks it is necessary to set up a so-called *backbone*, i.e., a set of vertices and links among them that is in charge of routing. In the specialised literature there is a general consensus that the backbone should be a *dominating set*, i.e., each vertex is either in the backbone or next to some vertex in it. To quote Rajaraman [14], *"The most basic clustering that has been studied in the context of ad hoc networks is based on dominating sets"*. Moreover, the following additional features are considered to be appealing: (a) the backbone should be "small" and (b) it should be connected or weakly connected.

A dominating set D is *connected* if the subgraph induced by it is connected, while D is *weakly connected* if the graph induced by the stars of vertices of D is connected. An equivalent definition is the following: connect every two vertices of D that are at distance 1 or 2. If the resulting graph (with vertex set D) is connected, then D is weakly connected. While connectivity appears to be a natural requirement, several authors have argued that the right notion to apply in the wireless context is weak connectivity (see for instance [5]). There is also a general consensus that, at least as a reasonable first approximation for many situations of interest, wireless networks can be modeled as message-passing, synchronous networks: vertices are processors, edges are communication links and the network is synchronous. Communication proceeds in synchronous *rounds*: in each round, every vertex sends messages to its neighbors, receives messages from its neighbors, and does some local computation. In this model, the running time is the number of communication rounds. This is the standard model assumed in this paper. What does it mean to "compute" a subset S of the vertices, such as a (weakly) connected dominating set, in this model? What we aim for is an efficient distributed algorithm, at the end of which each vertex knows whether it is a member of S or not. Henceforth, we denote by n the number of vertices of the network and by Δ its maximum degree.

There are several papers describing distributed algorithms for computing (weakly) connected dominating

*Computing Science, Chalmers University of Technology, SE-412 96 Göteborg, Sweden. Email: dubhashi@cs.chalmers.se.

†Informatica, La Sapienza, via Salaria 113, 00198 Roma, Italy. E-mail: mei@dsi.uniroma1.it.

‡Informatica, La Sapienza, via Salaria 113, 00198 Roma, Italy. E-mail: ale@dsi.uniroma1.it.

§School of Technology and Computer Science, TIFR, Homi Bhabha Road, Mumbai 400 005, India. E-mail: jaikumar@tifr.res.in. Part of this work was done while visiting Informatica, La Sapienza, via Salaria 113, 00198 Roma.

¶Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742. E-mail: srin@cs.umd.edu. Supported in part by NSF Award CCR-0208005.

sets in our model. These algorithms fall into two categories. Some algorithms might be fast, but there is no guarantee that the dominating set be small. The algorithm of [17] falls into this category. Other algorithms are guaranteed to compute a "best possible" dominating set with respect to size, but the number of rounds can be polynomial in the number of vertices of the network (see [16] for an analysis of several known dominating set algorithms). By "best possible" we mean an $O(\log \Delta)$-approximation; this is the best one can hope for, assuming that each vertex performs a polynomial-time-bounded computation during every communication round and that NP $\not\subseteq$ $\mathrm{DTIME}[n^{O(\log \log n)}]$ [6, 12, 15]. (Logarithms in this paper are to the base two, unless specified otherwise.) Algorithms in this category are typically distributed implementations of sequential approximation algorithms such as the ones in [7]. There are also quite nice distributed algorithms for computing "best possible" dominating sets in polylogarithmically many rounds ([8] and, with some modifications, [13]). But these dominating sets are, in general, neither connected nor weakly connected.

In this paper we present several randomised distributed algorithms that compute "best possible" connected and weakly connected dominating set in polylogarithmically (in n) many rounds. As above, by "best possible" we mean dominating sets of size $O(\log \Delta) \cdot Opt$. In fact, our guarantee is in terms of the smallest, not necessarily (weakly) connected, dominating set; since the smallest dominating set has cardinality at most that of a smallest (weakly) connected dominating set, this suffices.

Our results are based on a technique that might be of independent interest. We show that given a graph G that is connected, G can be sparsified by deleting all but a linear number of edges. Crucially, the resulting graph stays connected. This is interesting in view of two facts. First, as shown by Proposition 2.1, there is no deterministic or randomised algorithm running in $o(n)$ rounds that is capable of removing just one edge and preserving connectivity. In particular, this implies that there are no $o(n)$-round protocols for computing spanning or Steiner trees. Such trees are the tool of choice for connecting up a dominating set D while at the same time introducing only linearly many (in the size of D) new vertices. Second, the result is also interesting because real networks sometimes are bound to have a superlinear number of edges. Indeed, the following model has been proposed as a cost-effective way to set up sensor networks. The sensor networks are distributed essentially at random over the area of interest that can be assumed to be the unit square; each chip is a radio transmitter that sets up a connection to the k nearest chips. It is known that if one wants the network to be connected then k must be $\Omega(\log n)$ [18]. Therefore the resulting network will have a superlinear number of edges and the same might very well hold for the graphs induced by dominating sets in the graph. Furthermore, Proposition 2.4 shows the running time of $O(\log n)$ for our sparsification, to be best-possible.

In addition to domination, (weak) connectivity and small size, there is yet another feature of the backbone that would be attractive. Since the vertices in the dominating set take care of routing, it is desirable that dominators of "nearby" vertices are "relatively nearby". In particular, we show that our above results can be strengthened as follows. In Section 3, we present a polylogarithmic-time randomised distributed algorithm to construct a connected dominating set S of size at most $O(\log \Delta)$ times optimal, with the following additional "low stretch" property: for any pair of nodes u and v in G, the distance (in the subgraph induced by S) between their respective dominators is at most $O(\log n)$ times the distance between u and v in G. To our knowledge, this is the first result of this kind in this distributed setting.

Thus we present fast distributed algorithms for constructing dominating sets with good size, connectivity, and stretch properties; our running times are also shown to be essentially best-possible. We remark that in the case of the unit-disk graph, which is sometimes considered to be a good model of ad hoc networks, our algorithms deliver essentially a 12-approximation rather than an $O(\log n)$-approximation (to be precise, the approximated solution is of size at most $12 \cdot Opt + 3$). Finally, one of our algorithms (Algorithm 2 in § 2.2) admits a straightforward asynchronous implementation.

2 Distributed algorithms for connected dominating sets

In this section, we describe two randomised distributed algorithms that compute connected dominating sets in graphs. These algorithms run in time polylogarithmic in the size of the graph (we assume that all processors know the size of the graph, or at least a reasonable upper bound on it), and produce a solution that is within a factor $O(\log \Delta)$ of the optimum. In fact, both these algorithms have a common first phase, during which an $O(\log \Delta)$-approximated dominating set is computed. (In other words, a dominating set whose cardinality is at most $O(\log \Delta)$ times the minimum cardinality of a dominating set, is computed.) This can be achieved by using existing randomised distributed algorithms such as the one of [8] or a suitable modification of that in [13]. The running time of both algorithms is polylogarithmic.

Once an approximation for the dominating set is found, it remains only to "connect it up". Sequentially this is easy: a dominating set of size ℓ can be connected up as a tree by adding at most $2(\ell - 1)$ more vertices. This gives a connected dominating set of size within a factor of $O(\log \Delta)$ of the optimum dominating set. Guha and Khuller [7] showed that one can connect the dominating set somewhat more efficiently, for example, by using an approximation to the Steiner tree to connect up.

However we are interested in fast distributed algorithms, and in this setting, the following impossibility result applies; this shows that a (Steiner) tree-based approach will not lead to efficient algorithms.

PROPOSITION 2.1. *There is no deterministic distributed algorithm that in $o(n)$ time is able to remove one edge and preserve connectivity. For any constant $\epsilon > 0$, there is no randomised algorithm that runs in $o(n)$ time for this task, with a success probability of at least $1/2 + \epsilon$.*

Clearly, it suffices to show the lower bound for randomized algorithms. Consider the following argument.

Input: The simple cycle C on the vertex set $[n] = \{0, 1, \ldots, n-1\}$ with edge set $\{(i, (i+1) \bmod n) : i \in [n]\}$, or the path P_i obtained by deleting the edge $(i, (i+1) \bmod n)$ from the cycle C.

Claim: *Suppose there is a randomised distributed algorithm that runs in time t, and deletes some edges from the input graph, such that*

(i) On input C, with probability at least $1/2 + \epsilon$, the final graph has at least one edge missing;

(ii) On input P_i (for each i), with probability at least $1/2 + \epsilon$, the final graph is the same as P_i.

Then, $t \geq \frac{2n\epsilon}{1+2\epsilon} - 3/2$.

Note that there is a trivial randomised algorithm (with no communication) for $\epsilon = 0$: the edge $(0, 1)$ drops out with probability $\frac{1}{2}$ (except in P_0, where this edge does not exist and nothing is done). Our claim says that if we want to improve this to any positive constant ϵ, we need linear time. For the rest of the proof, when we mention vertices $i-1, i+1, i+2$ etc., the addition is meant modulo n.

Proof. Pick $i \in [n]$ randomly, and let I be the set of edges in C that are a distance at least $t+2$ away from the edge $(i, i+1)$ (for example, edges $(i+1, i+2)$ and $(i-1, i)$ are at distance 1 from $(i, i+1)$). Then, $|I| = n - 2t - 3$. Now, run the algorithm on C and observe the probability of the event $\mathcal{E}(I) \equiv$ "some edge in I is deleted by the algorithm". Then, one can use part (i) of the claim and an averaging argument to show that

$$\Pr[\mathcal{E}(I)] \geq (1/2 + \epsilon) \cdot \frac{n - 2t - 3}{n},$$

where the probability is over the internal coin tosses of the algorithm and the random choice of i. So we can fix $\hat{\imath}, \hat{I}$ so that

$$\Pr[\mathcal{E}(\hat{I})] \geq (1/2 + \epsilon) \cdot \frac{n - 2t - 3}{n}.$$

Now, one can verify easily that whenever the deleted edge is in $\hat{I}$, neither endpoint of that edge has received any communication emanating from vertices $\hat{\imath}$ or $\hat{\imath} + 1$. So, the probability of $\mathcal{E}(\hat{I})$ when the input graph is $P_{\hat{\imath}}$ is also at least $(1/2 + \epsilon) \cdot \frac{n-2t-3}{n}$. But, whenever this happens in $P_{\hat{\imath}}$, there is an error. Hence, $(1/2 + \epsilon) \cdot \frac{n-2t-3}{n} \leq 1/2 - \epsilon$. Our claim (as well as Proposition 2.1) follows by rearranging this inequality.

Thus, it is impossible to connect up a given dominating set as a tree in time significantly better than the diameter of the graph, and since we want polylogarithmic running time we cannot afford this. A way around this is to connect up all vertices in a dominating set D by means of a connected graph with a linear number of edges (in the size of D). In this section, we present two distributed algorithms, with polylogarithmic running time, for producing a connected spanning subgraph with a linear number (in the number of vertices) of edges. We then show that our $O(\log n)$ running time for this is best-possible.

The first algorithm is randomised. It may (very rarely) fail to have a linear number of edges, but the resulting graph is always connected and spanning. As a consequence, the solution produced by our randomised algorithm for the connected dominating set problem is *always* a connected dominating set, but with negligibly small probability, its size may not be within our targeted $O(\log \Delta)$ factor of the optimal solution.

The second algorithm is a deterministic version of the randomised algorithm and never fails.

Note: In the sequel we give algorithms for computing connected dominating sets. The straightforward modifications for computing weakly connected dominating sets are omitted from this extended abstract.

We start with a useful definition and a simple proposition.

DEFINITION 2.1. **(Powers of Graphs)** *Given a graph $G = (V, E)$ and a positive integer d, the graph G^d has vertex-set V, and two vertices u and v are connected by* *an edge in G^d iff they are distinct and are connected by a path of length at most d in G. Also, given a set $V' \subseteq V$, let $G^d[V']$ denote the subgraph of G^d induced by V'.*

Proposition 2.2. *Suppose a distributed algorithm $\mathcal{A}(H)$ for a synchronous network $H = (V, E)$ runs in at most $T(|V|)$ rounds, where $T(\cdot)$ is a non-decreasing function. Then, given a network $G = (V, E)$, a set $V' \subseteq V$, and a positive integer d, we can run $\mathcal{A}(G^d[V'])$ in our network G in at most $d \cdot T(|V|)$ rounds. (This is so, since we can simulate one round in $G^d[V']$ by d rounds in our network G.)*

From now on, let S denote the $O(\log \Delta)$–approximated dominating set produced by one of the algorithms in [8, 13]. Let G' denote $G^3[S]$ (i.e., the graph with vertex-set S, where two distinct vertices are connected by an edge iff their distance in G is at most 3). In different language, the following result has also been proved earlier in [7]:

Proposition 2.3. *If G is connected then G' is also connected.*

Proof. Suppose G' is not connected. Let v and w be vertices in different components of G' whose distance in G, $d_G(v, w)$, is shortest. Clearly, $d(v, w) \geq 4$, for otherwise, $\{v, w\}$ would be an edge of G', and v and w would not be in different components. Let $v = v_0, v_1, v_2, v_3, \ldots, v_d = w$ be a shortest path from v to w in G. Let u be the vertex of S that is closest in G to v_2; since S is a dominating set, we have $d_G(u, v_2) \leq 1$. So, $d_G(u, v) \leq 3$, and hence u and v are in the same component of G'. But then, u and w must be in different components, and $d(u, w) \leq d(v, w) - 1$, contradicting the choice of (v, w). Hence, G' must be connected.

Our goal now is to find a spanning subgraph H of G' of linear size, say with $2|V(G')|$ edges. The edges of H correspond to paths of length at most three in G. So, if we add to the dominating set S, at most two intermediate vertices for each edge of H, we get a connected dominating set of size at most $3|S|$, giving us a solution within a factor $O(\log \Delta)$ of the optimum. It is straightforward to see that once H has been found, the rest of the work can be done in a constant number of deterministic steps performed in a distributed fashion. So, it remains only to show a randomised distributed algorithm for constructing a connected spanning subgraph H of G' with at most $2|V(G')|$ edges. The following lemma is central to all our algorithms: to ensure that H is sparse, it is enough to ensure that H has no small cycles.

Lemma 2.1. (see, e.g., [9, Lemma 15.3.1]) *A graph on n vertices with girth g has at most $n^{1+\frac{2}{g-1}}$ edges.*

This theorem suggests the following strategy. Consider the graph G'. Keep deleting edges that appear in cycles of length less than $1 + 2 \log n$ until no such cycles remain. In the end, we will be left with at most $2n$ edges. While implementing this strategy using a distributed algorithm, we must ensure that the graph remains connected when we delete edges in parallel. We now present two algorithms achieving this. For notational convenience, we present these two algorithms as removing all cycles of length less than $1 + 2 \log n$ from an n-vertex network $G = (V, E)$; note that in our application, these algorithms will be run on G'. In view of Proposition 2.2, there is a blow-up of a constant factor (three) in running the algorithms on G'.

2.1 Algorithm 1.

1. *We start with the graph G and delete edges in order to remove all short cycles.* For $g = 3, 4, \ldots, \lfloor 1 + 2 \log n \rfloor$, destroy cycles of length g from G by deleting edges.

   1.1. Repeat the following steps $10 \log n$ times.

      a. Mark each edge with probability $\frac{1}{g}$ independently.

      b. If the edge e is the only marked edge in some cycle of length g, then delete e.

      c. Remove the marks on all the surviving edges.

Correctness: We show that after the first execution of the outer loop (corresponding to $g = 3$), with high probability, there are no triangles in the remaining graph. In general, after execution of the outer loop corresponding to $g = i$, with high probability, there are no cycles of length i in the remaining graph. To show this, we need two observations.

Claim 2.1. *The probability that a cycle C of length ℓ survives at the end of the iteration of the outer loop corresponding to $g = \ell$ is at most $\frac{1}{n^5}$.*

Proof. The probability that the cycle C of length ℓ is removed in one iteration of the inner loop is at least

$$\sum_{e \in C} \Pr[e \text{ is the only marked edge of } C];$$

i.e., at least

$$\ell \cdot \frac{1}{\ell} \left(1 - \frac{1}{\ell}\right)^{\ell - 1} \geq \frac{1}{3}.$$

Thus, the probability that the cycle survives after one iteration of the inner loop is at most $\frac{2}{3}$; the probability that it survives after $10 \log n$ iterations is at most $\frac{1}{n^5}$.

Claim 2.2. (Folklore) *The number of cycles in G of length girth(G) is at most $\binom{n}{3}$ if girth(G) $\neq 4$. (For girth(G) $= 4$, there are clearly at most n^4 such cycles.)*

Proof. Pick three equally spaced vertices on a cycle of length g. This set uniquely determines the cycle.

The rest of the argument is straightforward. After the first iteration of the outer loop the probability that any cycle of length 3 remains is at most $\frac{1}{n^2}$. Assuming that we did succeed in destroying all cycles of length 3 in the first iteration of the outer loop, the probability that some cycle of length 4 remains after the next iteration is at most $\frac{1}{n}$. Proceeding in this manner, we see that the probability that some cycle of length less than $1 + 2 \log n$ survives at the end is at most $\frac{1+2\log n}{n}$.

2.2 Algorithm 2. Consider the following deterministic, distributed algorithm for removing all cycles of length at most $1 + 2 \log n$, while preserving connectivity. It is assumed that each edge has a unique ID (if not a simple distributed way of achieving this is to make each edge choose a random real in $[0, 1]$ as its ID).

> Algorithm 2: Each edge, in parallel, drops out if it is the edge with the smallest ID in a cycle of length less than $1 + 2 \log n$.

The algorithm admits a straightforward, $O(\log n)$-time distributed implementation and it clearly breaks all cycles of the required length. We now prove that it preserves connectivity.

Lemma 2.2. *Algorithm 2 preserves connectivity.*

Proof. Consider the following sequential process. Order the edges according to their ID number. Consider the edges one-by-one from this list and delete them if they are currently in a cycle of length less than $1 + 2 \log n$. Clearly, when an edge is deleted it is in a cycle, so its deletion cannot disconnect the graph. Thus, in the end we will be left with a connected graph of girth at least $1 + 2 \log n$.

The claim follows by observing that this sequential process and Algorithm 2 remove exactly the same set of edges. To see this, observe that when an edge e is discarded by the sequential process, there is a cycle in which it is the highest ID edge.

Thus we obtain the following.

Theorem 2.1. *There is a distributed algorithm that, given an n-vertex connected graph G, computes in time $O(\log n)$ a connected subgraph H of G with a linear number of edges.*

This result can be shown to be time-optimal, as described in the following proposition.

Proposition 2.4. *For constants C and $\epsilon > 0$, suppose there is a randomised distributed algorithm with the following property. Given any connected graph G on n vertices, it produces a subgraph whose expected number of edges is at most Cn. Also, with probability one it terminates within $T(n)$ steps; with probability at least ϵ, it returns a connected subgraph. Then, we must have $T(n) = \Omega(\log n)$.*

Proof. We give a very brief proof sketch. Suppose for a contradiction, there exist constants C, ϵ, as well as an algorithm with $T(n) = o(\log n)$, which satisfy the conditions of the proposition. Take a graph G on n vertices with girth $10 \cdot T(n)$ and a superlinear number of edges (the existence of such a graph can be shown by standard probabilistic arguments). We can show that when G is fed to the algorithm, some edge $e = (a, b)$ must have a deletion probability of $1 - o(1)$. In the graph $G - e$, let A be the set of vertices at distance at most $2 \cdot T(n)$ from a, and B the set of vertices at distance at most $2 \cdot T(n)$ from b. Consider a minimal cut (set of edges) C of $G - e$ that separates A and B (note that A and B are disjoint). Run the algorithm on the connected graph $G - C$. With probability $1 - o(1)$, the algorithm will delete e and thus disconnect $G - C$, because in time $T(n)$, the edge e does not know what has happened to edges at distance $2T(n)$ or more. This leads to a contradiction.

Remark on asynchronous implementation. Let H be the graph obtained after running Algorithm 2 on the input graph G. Since the set of edges deleted by Algorithm 2 only depends on G and the distribution of ID's, the edges can be deleted in any order to obtain the same H. Therefore, Algorithm 2 admits a straightforward *asynchronous* implementation. This property might be useful from a practical point of view.

3 Dominating sets with low stretch

The algorithms in the previous section relied on the observation that if a graph has no cycles of length less than $1 + 2 \log n$, then it must have at most $2n$ edges. From this observation, one can infer the following stronger statement, which is a well-known property of spanners.

Proposition 3.1. (see, e.g., [2]) *For every graph G on n vertices, there is a subgraph H with $V(H) = V(G)$ and* *with a linear number of edges, such that for every two vertices $v, w \in V(G)$, $d_H(v,w) \le d_G(v,w) \cdot (1 + 2\log n)$.*

Proof. Start from the empty graph H. Consider the edges of G in some order. If the edge closes a cycle of length less than $1 + 2\log n$ in H, then discard it, otherwise add it to H. Clearly, the resulting graph H satisfies $d_H(v,w) \le d_G(v,w) \cdot (1 + 2\log n)$. Since H has no cycle of length less than $1 + 2\log n$, it has at most $2n$ edges.

This proposition and its sequential algorithmic proof lead us to the following question.

Question: Is there an efficient distributed algorithm for generating a (weakly) connected dominating set whose induced subgraph H has the "spanning" property guaranteed by the above proposition? (Note: Algorithms 1 and 2 are not guaranteed to produce such a solution.)

Answer: Yes! Algorithm 3 below does precisely this.

We will make use of the following graph decomposition theorem of Linial and Saks [10] (see also [3]).

THEOREM 3.1. **[10]** *There is a randomised distributed algorithm that with high probability, in $O(\log^2 n)$ time, partitions the vertices of a graph G into $O(\log n)$ colour classes, such that in the subgraph induced by any colour class, every two vertices in the same connected component have distance $O(\log n)$ in G. (The shortest path need not be confined to the connected component. Also, both the running time and correctness of this algorithm hold with high probability.)*

Our algorithm:

1. Apply the Linial-Saks decomposition algorithm to the graph G^d, where $d \triangleq \lfloor \frac{1}{2}(1 + 2\log n) \rfloor$. Let $C_1, C_2, \ldots, C_k$ be the colour classes, where $k = O(\log n)$. For brevity, we call the connected components of the graph induced by the colour classes *blocks*.
2. Let H be the empty graph with vertex set $V(G)$.
3. Cycle through the colour classes sequentially. In the ith iteration, consider the edges of G incident on vertices in C_i, and add them to H so that no cycles of length less than $1 + 2\log n$ are formed, and yet every edge that is not included closes a cycle of length less than $1 + 2\log n$ in H. Stated formally,
   a. In parallel for each block B of colour class C_i, consider the set E_B of all edges of $E(G)$ with at least one end-point in B.
   b. Compute a maximal subset $F_B \subseteq E_B$ such that there are no cycles of length less than $1 + 2\log n$ in $E(H) \cup F_B$. This can be done as follows. Assume that the decomposition of Step 1 satisfies the guarantees of Theorem 3.1 (which is the case with high probability, by Theorem 3.1). Then, every pair of vertices in B is at a distance of at most $O(d \cdot \log n) = O(\log^2 n)$. So, this step can be performed in $O(\log^2 n)$ time, for example, by collecting the required information at one node in each block, computing F_B locally, and transmitting this information back to all nodes in B.
   c. Set $E(H) \leftarrow E(H) \cup F_B$ and $E(G) \leftarrow E(G) - E_B$.

Correctness: Since F_B is chosen to be maximal in Step 3(b), when an edge e of G is excluded from $E(G)$, there is a cycle of length less than $1 + 2\log n$ in H that e closes. Thus, for every edge of G, its end points are at distance less than $1 + 2\log n$ in H. Now, to show that H has a linear number of edges, we claim that in the end H has no cycles of length less than $1 + 2\log n$. For, suppose a cycle was created for the first time in the iteration corresponding to colour class C_i. Let H_{i-1} be the value of H at the beginning of the iteration. Since $E(H_{i-1}) \cup F_B$ does not have any such cycle (by the definition of F_B), the cycle must have edges from F_{B_1} and F_{B_2}, for *distinct* blocks B_1 and B_2 of C_i. But this means that there are vertices in B_1 and B_2 that are within distance $\lfloor \frac{1}{2}(1 + 2\log n) \rfloor$ of each other in G. This is not possible because each block is a connected component of C_i in G^d, where $d = \lfloor \frac{1}{2}(1 + 2\log n) \rfloor$.

Running time: The Linial-Saks algorithm takes time $O(\log^2 n)$; thus, since $d = \Theta(\log n)$, we see by Proposition 2.2 that Step 1 can be run in $O(\log^3 n)$ time. The remaining steps are deterministic, and there is no further scope for error. There are, with high probability, only $O(\log n)$ colour classes. Each class is processed in $O(\log^2 n)$ time with high probability, as seen above. Thus, the overall running time of our algorithm is $O(\log^3 n)$.

THEOREM 3.2. *Suppose we are given a distributed network G. There is a randomised distributed algorithm that in polylogarithmic time computes a spanning subgraph H of the graph G, such that*

(a) H has no cycles of length $1 + 2\log n$, and consequently, H has a linear number of edges; and

(b) for every edge of G, there is a path in H of length at most $1 + 2\log n$.

COROLLARY 3.1. *Suppose we are given a distributed network $G = (V, E)$. There is a randomised distributed algorithm that in polylogarithmic time computes a subset V' of V, such that*

(i) *V' is a connected dominating set, and has size $O(\log \Delta)$ times the size of a minimum-sized dominating set;*

(ii) *for every pair of adjacent vertices in G, there is a path connecting them with all internal vertices lying in V', with the path-length being $O(\log n)$. (Thus, for any pair of vertices v, w in G, there is a path with all internal vertices lying in V', with the path-length being at most $O((\log n) \cdot d_G(v, w))$.)*

Proof. As in Section 2, let S denote an $O(\log \Delta)$–approximated dominating set, and let G' denote $G^3[S]$. Run the algorithm guaranteed by Theorem 3.2 on the graph G'; by Proposition 2.2 and Theorem 3.2, this requires only polylogarithmic time, with high probability. The resulting spanning subgraph H of G' has at most $2|V(G')|$ edges. As in Section 2, the edges of H correspond to paths of length at most three in G; we add to the dominating set S, at most two intermediate vertices for each edge of H. This yields a connected dominating set of size at most $3|S|$. What about the "spanner" property (ii)? If (u, v) is an edge in G, then it is easy to check that the dominators $u' \in S$ and $v' \in S$ of u and v respectively, are adjacent in G'. By property (b) of Theorem 3.2, we know that there is a path of length $O(\log n)$ connecting u' and v' in H. Finally, since each edge in H corresponds to a path of length at most 3 in G, property (ii) follows.

4 Constant-factor approximations for unit-disk graphs

Unit-disk graphs can in some cases be taken as good models for radio networks, including ad hoc networks. Given a set of points in the plane, the graph is obtained by having a vertex for each point and by connecting any two points at Euclidean distance one or less. In [16] the following result is proven: if I is a maximal independent set (MIS) in a unit-disk graph G, then $|I|$ is at most one more than four times the smallest dominating set size. Since an MIS is also a dominating set, we can apply the same sparsification techniques of the preceding sections to augment I to a connected dominating set of size at most $3 \cdot |I|$, which gives (essentially) a 12-approximation. It is well-known that an MIS can be computed in $O(\log n)$ communication rounds in our distributed model using randomization [1, 11].

5 Open Problems and Summary of results

It would be nice to develop a more direct and computationally simpler method to obtain connected dominating sets with good stretch properties. One way to achieve this would be to solve the following problem. Let $G = (A, B, E)$ be a bipartite network. A subset $A' \subseteq A$ is a *cover* if $N(A')$, the set of neighbours of A', is the whole of B. We want a fast distributed algorithm that finds a *minimal* such A'. In essence, this is the unweighted set cover problem where the input instance is represented as a bipartite graph, and where one is to find a minimal, as opposed to a small, cover. Besides being an interesting symmetry-breaking problem *per se*, finding a fast distributed algorithm would give as a by-product a way to compute a sparse, connected subgraph with low stretch. This follows from the following discussion.

DEFINITION 5.1. *Given an input graph G, a cycle is small if its length is a at most $c \log n$, for some arbitrary, but fixed, parameter c. Let $\mathcal{C}$ be the collection of small cycles (these are those that we want to break). A set A of edges is a cover if $G[V, E - A]$ has no $\mathcal{C}$-cycles.*

CLAIM 5.1. *Let A be a minimal cover with respect to $\mathcal{C}$. Then, for each $a \in A$ there is a $c_a \in \mathcal{C}$ such that c_a is broken by a and only by a.*

Proof. If $a \in A$ does not have a corresponding c_a, it can be removed from A. This would violate the minimality assumption.

CLAIM 5.2. *Let A be a minimal cover with respect to the set of small cycles $\mathcal{C}$. Then, $G[V, E - A]$ has stretch $O(\log n)$.*

Proof. Given any uv-path p of G, every edge e of p either is present in $G[V, E - A]$ or, if removed, can be bypassed by means of the corresponding $c_e \in \mathcal{C}$.

The problem of finding a minimal set of edges to break all small cycles reduces to that of computing a minimal cover in a bipartite graph, according to the definition above. The two sides of the bipartition are edges on the B-side and small cycles on the A-side. A cycle and an edge are connected in the bipartite graph if the edge belongs to the cycle. An algorithm for finding a minimal set of B-vertices covering all the A-vertices can be simulated in logarithmic time in the bipartite edge-cycle graph. Thus, a fast distributed solution to the minimal (as opposed to small) set cover problem would yield an efficient algorithm for computing sparse, spanning subgraphs with low stretch.

We now summarize the results contained in this paper.

Result 1 (based on Algorithms 1 and 2): There is a randomised distributed algorithm for computing a $O(\log \Delta)$ approximation to the minimal connected dominating set in time $O((\log n)^2)$.

Result 2 (based on Algorithm 3): There is a randomised distributed algorithm, that given a graph G on n vertices, computes a connected subgraph H of G, with the following properties:

Domination. H is dominating: every vertex in $G - H$ has a neighbour in H.

Sparsity. The number of vertices in H is within a factor $O(\log \Delta)$ of the minimum size of a dominating set in G.

Stretch (or dilation). For every pair of vertices (v, w) in G, there is a path using H (that is, with all internal vertices belonging to H) of length at most $O((\log n) \cdot d_G(v, w))$.

(Note: the approximation factor is $O(\log \Delta)$ for the size of the dominating set, but $O(\log n)$ for the stretch.)

Acknowledgments. We thank Samir Khuller and the SODA 2003 referees for their helpful comments.

References

[1] N. Alon, L. Babai, and A. Itai. A Fast and Simple Randomized Parallel Algorithm for the Maximal Independent Set Problem. J. Algorithms, 7:567–583, 1986.

[2] I. Althöfer, G. Das, D. P. Dobkin, D. Joseph, and J. Soares. On sparse spanners of weighted graphs. Discrete Comput. Geom., 9:81–100, 1993.

[3] B. Awerbuch and D. Peleg. Sparse Partitions. 31st IEEE Symp. on Foundations of Computer Science, pages 503–513, 1990.

[4] Khaled Alzoubi, Peng-Jun Wan, and Ophir Frieder. Message-Optimal Connected-Dominating-Set Construction for Routing in Mobile Ad Hoc Networks. Proceedings of Mobihoc 2002.

[5] Yuanzhu Chen and Arthur Liestman. Approximating Minimum Size Weakly-Connected Dominating Sets for Clustering Mobile Ad Hoc Networks. Proceedings of Mobihoc 2002.

[6] U. Feige. A Threshold of $\ln n$ for Approximating the Set Cover. Journal of the ACM, 45(4), 634–652, July 1998.

[7] S. Guha and S. Khuller. Approximation algorithms for connected dominating sets. Algorithmica, 20(4):374–387, 1998.

[8] L. Jia, R. Rajaraman and T. Suel. An Efficient Distributed Algorithm for Constructing Small Dominating Sets. Proceedings of the Annual ACM Symposium on Principles of Distributed Computing, pages 33–42, 2001.

[9] J. Matoušek. Lectures on discrete geometry. Graduate Texts in Mathematics, Vol. 212, Springer, 2002.

[10] N. Linial and M. Saks. Low diameter graph decompositions. Combinatorica, 13:441–454, 1993.

[11] M. Luby. A Simple Parallel Algorithm for the Maximal Independent Set Problem SIAM J. on Computing, November 1986, Vol. 15, No. 4, pp. 1036-1053.

[12] C. Lund and M. Yannakakis. On the Hardness of Approximating Minimization Problems. Journal of the ACM, 41(5), 960–981, 1994.

[13] S. Rajagopalan and V.V. Vazirani. Primal-Dual RNC Approximation Algorithms for (multi)Set (multi)Cover and Covering Integer Programs. SIAM J. on Computing 28, 2 (1998) 525-540.

[14] R. Rajaraman. Topology control and routing in ad hoc networks: a survey. SIGACT News, Vol. 33, Number 2, pages 60–73, 2002.

[15] L. Trevisan. Non-approximability Results for Optimization Problems on Bounded Degree Instances. Proceedings of the 33rd ACM STOC, 453–461, 2001.

[16] Peng-Jun Wan, Khaled M. Alzoubi and Ophir Frieder. Distributed construction of connected dominating set in wireless ad hoc networks. Proceedings of Infocom 2002.

[17] Jie Wu and Hailan Li. A Dominating-Set-Based Routing Scheme in Ad Hoc Wireless Networks, special issue on *Wireless Networks* of the Telecommunication Systems Journal, Vol. 3, 63–84, 2001.

[18] F. Xue and P. R. Kumar. The number of neighbors needed for connectivity of wireless networks. Manuscript, 2002. Available from *http://black1.csl.uiuc.edu/~prkumar/postscript_files.html*.