A Primal-Dual Bicriteria Distributed Algorithm for Capacitated Vertex Cover *

F. Grandoni† J. Könemann‡ A. Panconesi § M. Sozio¶

January 16, 2008

Abstract

In this paper we consider the capacitated vertex cover problem which is the variant of vertex cover where each node is allowed to cover a limited number of edges. We present an efficient, deterministic, distributed approximation algorithm for the problem. Our algorithm computes a $(2+\epsilon)$-approximate solution which violates the capacity constraints by a factor of $(4+\epsilon)$ in a polylogarithmic number of communication rounds. On the other hand, we also show that every efficient distributed approximation algorithm for this problem must violate the capacity constraints.

Our result is achieved in two steps. We first develop a 2-approximate, sequential primal-dual algorithm that violates the capacity constraints by a factor of 2. Subsequently, we present a distributed version of this algorithm. We demonstrate that the sequential algorithm has an inherent need for synchronization which forces any naive distributed implementation to use a linear number of communication rounds. The challenge in this step is therefore to achieve a reduction of the communication complexity to a polylogarithmic number of rounds without worsening the approximation guarantee too much.

Keywords: Vertex Cover, Approximation Algorithms, Distributed Algorithms, Primal-Dual Algorithms

* A preliminary version of this paper appeared in the Proceedings of the 24th Symposium on Principles of Distributed Computing, Las Vegas, Nevada, USA, 2005

† Dipartimento di Informatica, Sistemi e Produzione, Università di Roma Tor Vergata, Via del Politecnico 1, 00133, Roma, Italy. Email: grandoni@disp.uniroma2.it.

‡ Department of Combinatorics and Optimization, University of Waterloo, 200 University Avenue West, Waterloo, ON N2L 3G1, Canada. Email: jochen@uwaterloo.ca. This work was done while being on leave at the Dipartimento di Informatica, Sapienza Università di Roma.

§ Dipartimento di Informatica, Sapienza Università di Roma, Via Salaria 113, 00198, Roma, Italy. Email ale@di.uniroma1.it.

¶ Department of Databases and Information Systems, Max-Planck-Institut für Informatik, Stuhlsatzenhausweg 85, 66123 Saarbrücken, Germany. Email: msozio@mpi-inf.mpg.de.

1 Introduction

The *capacitated vertex cover* problem (capVC) is the variant of vertex cover in which there is a limit on the number of edges that a vertex can cover. A precise formulation of the problem is as follows. We are given an n-vertex undirected graph $G = (V, E)$, non-negative weights $\mathtt{wt}_v$ and vertex capacities $B_v \geq 1$ for all vertices $v \in V$. A solution to a given capVC instance consists of a subset $C \subseteq V$ and an assignment $\pi : E \to C$ of edges to vertices such that

1. $\pi(e) \in \{u, v\} \cap C$ for all edges $e = (u, v) \in E$, and
2. $|\pi^{-1}(v)| \leq B_v$ for all $v \in C$.

The first set of constraints says that every edge must be covered by some vertex in the cover C. The second condition limits the number of edges that can be assigned to any cover vertex v to B_v. The goal is to find a feasible solution that has minimum total weight

$$\mathtt{wt}(C) := \sum_{v \in C} \mathtt{wt}_v.$$

We emphasize the difference between the above *hard-capacity* version of capVC and its *soft-capacity* counterpart (capVC$_s$): in capVC$_s$, each vertex $v \in V$ may be included $x_v \geq 0$ times in a cover. Vertex v then contributes $x_v \mathtt{wt}_v$ to the weight of the cover, and the maximum number of edges that can be assigned to v is $x_v B_v$.

In this paper we present *bicriteria* sequential and distributed approximation algorithms for the (hard) capacitated vertex cover problem. Given a feasible instance of the problem of optimal weight opt, our sequential primal-dual algorithm computes a vertex cover C of weight $\sum_{v \in C} \mathtt{wt}_v \leq 2\mathtt{opt}$, which assigns at most $2B_v$ edges to each cover vertex $v \in C$. Note that, differently from the hard-capacity case, capacities might be violated. However, the amount of the violation is bounded, which is not the case for capVC$_s$. We also remark that, unlike capVC$_s$, every $v \in C$ contributes $\mathtt{wt}_v$ to the weight of the cover even when its capacity B_v is exceeded.

The distributed implementation of our method has an additional input parameter $\epsilon > 0$ and computes a cover of weight at most $(2 + \epsilon)\mathtt{opt}$ that violates the capacity bound of each cover vertex by a factor of at most $(4 + \epsilon)$. In the synchronous, message-passing model of computation the distributed algorithm takes $O(\log(nW)/\epsilon)$ many rounds, where

$$W = \mathtt{wt}_{max}/\mathtt{wt}_{min}$$

is the ratio of largest to smallest vertex weight in the given instance. This reduces to $O(\log n/\epsilon)$ for the interesting case of unit weights. We remark that our algorithm is deterministic, while typically efficient distributed algorithms for graph problems require randomization (see [11, 20, 22, 23, 27, 28] among others).

Observe that any sub-linear distributed algorithm for capVC must violate the capacity constraints. Consider for instance a ring where every vertex has unit capacity. A feasible solution provides a consistent orientation of the ring, something that requires a linear number of communication rounds. Therefore a bicriteria solution is the best one can hope for in a distributed setting. In this paper we show that indeed every efficient distributed approximation algorithm for capVC must violate the capacity constraints by a large additive term.

In our opinion the most interesting aspect of our work is that the distributed algorithm is derived in a systematic fashion from a sequential primal-dual algorithm. To our knowledge, the first result of this kind is the $(2+\epsilon)$-approximate vertex cover algorithm described in [16]. Although described for the PRAM setting, the algorithm can be easily adapted to the distributed case. Our paper takes the primal-dual approach pioneered in [16] one step further, giving a new and considerably more sophisticated example. Chudak et al. [4] recently showed that the techniques introduced in this paper can be extended to yield efficient distributed primal-dual algorithms for vertex cover with soft capacities and for the facility location problem. The power of the primal-dual method in the design of approximation algorithms is well established. In this paper we provide further evidence to the fact that it is also a valuable tool in the design of distributed algorithms.

Capacity constraints arise naturally in distributed computing and computer networking. E.g., the scatternet-formation problem of ad hoc Bluetooth networks asks for a small dominating set where each vertex in the set dominates at most 7 vertices [6]. More generally, a small dominating set can act as the backbone of the routing infrastructure of an ad hoc network (see [29, 26] and references therein). Capacities model computational and energy limitations and provide effective means to enforce load distribution among the vertices of the backbone. To the best of our knowledge, our paper is the first result that considers a capacitated network design problem from the distributed computing point of view. Recently, Moscibroda and Kuhn gave an LP-based, bicriteria distributed solution to the capacitated dominating set problem [17].

Related work. Vertex cover with capacities has received considerable attention in recent years in the sequential setting, while our main motivation is to study it from a distributed point of view. The capacitated vertex cover problem was first introduced by Guha et al. [12] who presented a simple 4-approximate LP-rounding based algorithm for capVC_s. Later on, the authors showed a 2-approximate primal-dual algorithm. Subsequently, Gandhi et al. [10] presented a 2-approximate LP-rounding algorithm for capVC_s.

The hard-capacitated vertex-cover problem is significantly harder than its soft-capacitated variant. Chuzhoy and Naor [5] first gave a sophisticated 3-approximate LP-rounding algorithm for the special case of capVC with uniform vertex weights. Finally, in [9], Gandhi et al. presented an LP-rounding-based 2-approximation algorithm for capVC with uniform weights.

In [5], Chuzhoy and Naor also showed that capVC in the presence of non-uniform vertex weights is as hard to approximate as set-cover. Lund and Yannakakis [24] proved that there is no $o(\log n)$-approximation for the latter problem unless NP $\subseteq$ DTIME$(n^{O(\log\log n)})$ (see also [8] for a refined result). Based on work by Bellare et al. [3], and Raz and Safra [30], Alon et. al [1] recently improved upon this result and showed that no $o(\log n)$-approximation for the set-cover problem is possible unless P $=$ NP. Chuzhoy and Naor's work implies that these hardness bounds translate to the capVC problem.

The best known approximation algorithm for the vertex-cover problem without capacity constraints, is due to Karakostas [15] who presented a $(2-\Theta(1/\sqrt{\log(n)})$-approximation for the problem. This improves upon earlier $(2-o(1))$-approximation algorithms due to Halperin [13], Bar-Yehuda and Even [2] and Hochbaum [14]. As mentioned, the same bound is essentially achievable in the distributed setting [16].

Unconditional lower-bounds based on communication constraints, as opposed to unproven complexity theoretic assumptions, have been proved since the early stages [21, 25]. For more recent work see [18]. Also, Elkin [7] recently established trade-offs between the performance guarantee of a distributed approximation algorithm for the minimum-cost spanning tree problem and the number of communication rounds it needs.

Finally, we mention that LP-duality has been previously used to design distributed algorithms for the dominating set problem [19, 28].

Our contribution. The first result we give is a bicriteria primal-dual approximation algorithm for capVC.

Theorem 1 *Given a feasible* capVC *instance with capacities $B_v \geq 1$ for all $v \in V$. There is a polynomial-time primal-dual algorithm that computes a vertex cover (C, π) of weight at most $2\mathtt{opt}$ that assigns at most $2B_v$ edges to each vertex $v \in C$.*

We remark that if the input instance does not have a feasible solution, then our algorithm either computes a feasible solution for the (capacity) relaxed version of the problem, or it terminates with a certificate of infeasibility.

Theorem 1 is a natural step toward proving the main result of this paper.

Theorem 2 *Given a feasible instance of* capVC *with capacities $B_v \geq 1$ for all $v \in V$, and let $\epsilon \in (0, 1]$ be an input parameter. There is a distributed deterministic algorithm that computes a vertex cover (C, π) of weight at most $(2 + \epsilon)\mathtt{opt}$ that assigns at most $(4 + \epsilon)B_v$ edges to each vertex $v \in C$. The algorithm needs $O(\log(nW)/\epsilon)$ rounds.*

We remark that the message-size of our algorithm is $O(\log n + \log \mathtt{wt}_{max})$.

Note that the running time is strongly polylogarithmic for polynomially large weights only. This includes the important special case of unit weights. Obtaining a strongly polylogarithmic algorithm in general is a challenging open problem.

Similar to the sequential case, if the input instance does not have a feasible solution the algorithm either computes a feasible solution for the relaxed version of the problem, or terminates with a certificate of infeasibility. The latter however is necessarily local in nature. That is, some vertices will know that the algorithm has failed, but it requires a linear number of communication rounds to distribute this information across the network in general.

These theorems are complemented by the following lower-bound on the communication complexity of any algorithm for the weighted capVC problem with hard capacities:

Theorem 3 *Let $B, k \geq 1$ be integer parameters. There is a* capVC *instance with uniform vertex capacities B, for which any distributed approximation algorithm that assigns less than $(1 + 1/k) \cdot B$ edges to all vertices, must take at least k communication rounds.*

This result shows in particular that violating the capacity constraints is necessary and provides a trade-off between violation of capacities and running time.

Organization of the paper. In the following Section 2 we describe a sequential algorithm for the capacitated vertex-cover problem and give a proof of Theorem 1. The next section shows how to turn the sequential algorithm into a distributed one. This is done in two steps. First, we show how to convert the sequential algorithm into a distributed one that computes a vertex cover that satisfies the approximation requirement. In this step we assign only a subset of the edges. In the second and final step we assign all the remaining edges. The proof of Theorem 3 is given in Section 4.

2 A sequential primal-dual algorithm

We present a so called *primal-dual* algorithm for the capVC problem. The algorithm and its analysis are based on linear programming duality. In the next section we therefore introduce a linear programming formulation of the problem together with its dual. Following that we describe our sequential algorithm and we conclude this section with an analysis of the presented method.

2.1 A linear programming formulation

The problem can be formulated as an integer program where we introduce a binary indicator variable x_v for each $v \in V$. We let $x_v = 1$ if $v \in C$ and $x_v = 0$ otherwise. For each edge $e = (u, v) \in E$ we introduce two binary variables $y_{e,v}$ and $y_{e,u}$. For $w \in \{u, v\}$ we let $y_{e,w} = 1$ if and only if $\pi(e) = w$. In the following let $\delta(v)$ be the set of edges incident to vertex $v \in V$ in G.

$$\begin{aligned}
\min \quad & \sum_{v \in V} \mathtt{wt}_v \cdot x_v && && \text{(IP)} \\
\text{s.t} \quad & y_{e,v} + y_{e,u} \geq 1 && \forall e = (u, v) \in E && (1) \\
& y_{e,w} \leq x_w && \forall e = (u, v) \in E, \\
& && \forall w \in \{u, v\} && (2) \\
& \sum_{e=(v,u) \in \delta(v)} y_{e,v} \leq B_v \cdot x_v && \forall v \in V && (3) \\
& y_{e,v}, x_v \in \{0, 1\} && \forall e \in E, v \in V && (4)
\end{aligned}$$

We now let (LP) be the standard LP relaxation obtained from (IP) by replacing the constraints (4) by

$$\begin{aligned}
& y_{e,v} \geq 0 && \forall e = (u, v) \in E \\
& 0 \leq x_v \leq 1 && \forall v \in V && (5)
\end{aligned}$$

In the following we use $(i)_v$, $(i)_e$, and $(i)_{e,v}$ to denote constraint (i) for vertex $v \in V$, edge $e \in E$, and pair $(e, v) \in E \times V$, respectively. In the linear-programming dual of (LP) we associate variables $\alpha_e, \beta_{e,w}, \gamma_v$ and ω_v with constraints $(1)_e$, $(2)_{e,w}$, $(3)_v$, and $(5)_v$, respectively. The linear programming dual of (LP) is then

$$\begin{aligned}
\max \quad & \sum_{e \in E} \alpha_e - \sum_{v \in V} \omega_v && && \text{(D)} \\
\text{s.t} \quad & \alpha_e \leq \beta_{e,w} + \gamma_w && \forall e = (u, v) \in E, \\
& && \forall w \in \{u, v\} && (6) \\
& \sum_{e=(u,v) \in E} \beta_{e,v} \leq \mathtt{wt}_v + (\omega_v - B_v \cdot \gamma_v) && \forall v \in V && (7) \\
& \alpha, \beta, \gamma, \omega \geq 0
\end{aligned}$$

2.2 The algorithm

We remark that the following simple LP rounding scheme similar to that proposed by Guha et al. [12] yields a 2-approximate vertex cover in which each vertex v covers at most $2B_v$ edges: Solve the LP relaxation

(LP) and let (x, y) be its optimal solution. The cover set C consists of all vertices v with $x_v \geq 1/2$. For $e = (u, v) \in E$, constraint $(1)_e$ implies that there is $w \in \{u, v\}$ such that $y_{e,w} \geq 1/2$. We assign edge e to vertex w in this case. Clearly, the weight of the vertices in C is at most twice the optimal LP value. Moreover, each vertex $v \in C$ has at most $2B_v$ assigned edges.

We provide an alternate primal-dual algorithm in this section. As we shall see later, this algorithm possesses an efficient distributed implementation.

The high-level idea in primal-dual algorithms is to find a pair of feasible solutions for (D) and (IP). Subsequently we upper-bound the performance ratio of the algorithm by bounding the multiplicative gap between the objective values of the two solutions. Thus, our goal is to find a primal-dual pair of solutions whose objective functions values are within a small multiplicative constant of each other. Primal-dual algorithms typically construct such a pair in an iterative manner: starting from a trivial feasible dual solution and an infeasible primal one, the algorithm continuously raises the objective value of the dual solution while maintaining its feasibility, and it changes the partial primal solution in order to attain feasibility.

Our primal-dual `capVC` algorithm starts with the dual feasible solution $\alpha = \beta = \gamma = \omega = 0$ and the infeasible primal solution $x = y = 0$. In order to obtain a feasible vertex cover, we have to a) select a set of cover vertices, and b) assign each edge $e \in E$ to one of its end-points (which must be in the cover). As is typical in primal-dual approximation algorithms, these decisions are governed by *primal complementary slackness*.

In the following we say that a vertex $v \in V$ is *tight* for a current dual solution $(\alpha, \beta, \gamma, \omega)$ if constraint $(7)_v$ holds with equality. Similarly, a pair $(e, w) \in E \times V$ is tight if constraint $(6)_{e,w}$ is satisfied with equality. Our algorithm will now increase the value of some of the dual variables and as a consequence create tight vertices and tight edge-vertex pairs. Tight vertices are candidates for our final cover and we will eventually choose a subset of these. Each edge e will eventually be assigned to one of its tight endpoints. In particular, edge $e = (u, v) \in E$ will only be assigned to endpoint $w \in \{u, v\}$ if (e, w) is tight. Once an edge is assigned to a vertex, we will remove it from the graph and continue. Similarly, once all edges incident to a certain vertex $v \in V$ have been decided, the vertex is removed from the graph. The algorithm terminates, when the graph is empty and, hence, when all edges have been assigned.

We now describe the algorithm. As customary with primal-dual algorithms, we describe it as a continuous process that can be implemented in polynomial-time by standard techniques. Initially all edges are unassigned. At any given point, the algorithm increases the value of dual variables α_e of all unassigned edges uniformly at the same (unit) rate. Increasing variables α_e for unassigned edges increases the left-hand side of constraints of type (6) and we will have to also increase some of the β and γ variables in order to maintain dual feasibility. We describe the update process for these variables for each vertex $v \in V$ depending on its tightness:

v **is non-tight** In this case, we increase $\beta_{e,v}$ for all $e \in \delta(v)$ uniformly. Thus, the left- and right-hand side of constraint $(6)_{e,v}$ for all $e \in \delta(v)$ increase at the same rate and feasibility is maintained.

v **is tight** If v has at most $2B_v$ incident edges, we add v to the cover, assign all edges in $\delta(v)$ to v and delete v and the newly assigned edges from G. Otherwise v has more than $2B_v$ incident edges. In this case, we increase ω_v at rate B_v, γ_v at unit rate and we leave $\beta_{e,v}$ as is for all $e \in \delta(v)$. As a consequence, left- and right-hand side of $(7)_v$ remain unchanged, and left- and right-hand side of $(6)_{e,v}$ for all $e \in \delta(v)$ change at the same (unit) rate. Feasibility is therefore maintained also in this case.

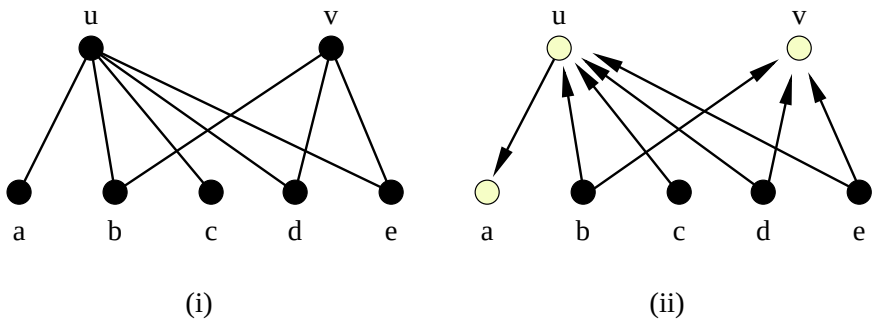

Figure 1: An example instance for the primal-dual `capVC` algorithm.

We emphasize that our algorithm maintains a feasible dual solution for (D) for the original instance. In particular, deleting a vertex v and an edge $e \in \delta(v)$ means that the values of variables $\alpha_e, \beta_{e,v}, \omega_v$ and γ_v are frozen at their current state from this point on in the algorithm. The algorithm terminates when all edges have been assigned.

We demonstrate the algorithm using the example instance in Figure 1 (i) where we let $B_u = 2$, $B_v = 3$ and $B_w = \infty$ for all other vertices w. We also choose $\texttt{wt}_a = 2$, $\texttt{wt}_u = 5$, $\texttt{wt}_v = 6$ and all other vertices have infinite weight. In the following we use V and E to refer to the vertex and edge sets of the given instance.

Running our algorithm for one time unit results in $\alpha_e = 1$ for all $e \in E$ and $\beta_{e,w} = 1$ for all $w \in V$ and for all $e \in \delta(w)$. At this point, constraint $(7)_u$ is tight. As u has $5 > 2 \cdot B_u = 4$ unassigned incident edges, we can not assign any edge at this point. Thus, we continue to increase α_e for all edges $e \in E$. Simultaneously, we increase $\beta_{e,w}$ for all $w \in V \setminus \{u\}$ and for all $e \in \delta(w)$ at unit rate, we increase ω_u at rate $B_u = 2$ and γ_u at unit rate.

After 1 more time unit, the positive dual variable values are

$$\begin{aligned}
\alpha_e &= 2 \quad \forall e \in E \\
\beta_{e,u} &= 1 \quad \forall e \in \delta(u) \\
\beta_{e,w} &= 2 \quad \forall w \in V \setminus \{u\}, \forall e \in \delta(w) \\
\omega_u &= 2 \\
\gamma_u &= 1.
\end{aligned}$$

At this point, vertices a, u and v are tight. Vertex a and v have one and three incident edges, respectively, and we can hence add both vertices to the cover and delete them and their incident edges from the graph. The number of remaining edges all of which are incident to u is now $4 = 2B_u$. We can assign all of them to u.

Figure 1 (ii) shows the computed primal solution; cover vertices are shaded and arc directions indicate edge assignment. We note that the primal solution is feasible for (IP) only if we relax the capacity constraint of vertex u. In fact, it is not hard to see that any feasible solution to (IP) for this instance must have infinite weight.

2.3 Analysis

In this section we present a proof of Theorem 1.

Assume first that the algorithm from Section 2.2 does not terminate for a given input instance. It is then not hard to see that the algorithm must reach a point in the execution, where each tight vertex $v \in V$ has degree more than $2B_v$ and where each remaining edge is incident to tight vertices on both ends. Using the pigeon-hole principle it follows that, in any assignment of edges to vertices, there must be at least one tight vertex v that is assigned more than B_v edges. Thus the given input instance is infeasible, and the set of tight vertices together with the set of unassigned edges certifies this fact.

In the following we focus on feasible `capVC` instances. For such instances our algorithm terminates with a cover $\mathcal{C}$, an assignment $\{y_{e,v}\}_{e\in E, v\in V}$ of edges to vertices in the cover, and a corresponding dual solution $(\alpha, \beta, \omega, \gamma)$. We first show that the dual is feasible for (D).

Lemma 1 *The dual solution $(\alpha, \beta, \omega, \gamma)$ is feasible for (D).*

Proof: We can think of the execution of the algorithm as a process over time: The algorithm starts at time 0 and then raises α_e by 1 for all edges per unit of time. We prove the lemma by induction on (appropriately discretized) time.

Our initial dual solution is clearly feasible. Now consider a later time in the algorithm. Let $\mathcal{O}$ be the set of tight vertices at that time.

For a vertex $v \in V \setminus \mathcal{O}$ and for an edge $e \in \delta(v)$ we raise α_e and $\beta_{e,v}$ simultaneously and hence maintain dual feasibility. For a vertex $v \in \mathcal{O}$ we raise ω_v at a rate of B_v per time unit and we raise γ_v at unit rate. For all edges $e \in \delta(v)$ we raise α_e at unit rate as well. It is not hard to see that we maintain dual feasibility this way. ■

We are ready to prove that our algorithm computes a 2-approximate primal solution.

Lemma 2 *Our algorithm terminates with a vertex cover $\mathcal{C}$ and a corresponding feasible dual solution $(\alpha, \beta, \gamma, \omega)$ whenever there exists a feasible solution (x, y) for (LP). In particular, we must have*

$$\sum_{v\in\mathcal{C}} \mathtt{wt}_v \leq 2\left(\sum_{e\in E}\alpha_e - \sum_{v\in V}\omega_v\right).$$

Proof:

Let $v \in \mathcal{C}$ be a vertex in the computed vertex-cover and let $e \in \delta(v)$ be an edge that is incident to v. Notice that our algorithm always maintains

$$\alpha_e \geq \beta_{e,v} \tag{8}$$

since α_e is raised whenever $\beta_{e,v}$ increases and the rate of increase is the same.

Observe also that γ_v is only increased if the degree $\mathtt{deg}(v)$ of vertex v exceeds $2B_v$. Let $\delta_1(v) \subseteq \delta(v)$ be the set of edges that are incident to v when γ_v is increased for the last time in the algorithm and notice that we must have $|\delta_1(v)| > 2B_v$.

Consider an edge $e \in \delta_1(v)$ and note that γ_v and α_e increase at the same rate after the point of time where v becomes tight. Notice also that the algorithm increases α_e and $\beta_{e,v}$ at the same rate before v becomes

tight. Variable $\beta_{e,v}$ is not increased after v becomes tight, and γ_v is not increased before v becomes tight. Therefore, for all $e \in \delta_1(v)$ we must have

$$\alpha_e = \beta_{e,v} + \gamma_v. \tag{9}$$

Since v is tight when the algorithm adds it to the cover and deletes it from the graph, it must also be the case that

$$\sum_{e \in \delta(v)} \beta_{e,v} = \mathtt{wt}_v + \omega_v - B_v \cdot \gamma_v = \mathtt{wt}_v \tag{10}$$

where the last equality follows from the fact that we raise ω_v at a rate of B_v if and only if we raise γ_v at a rate of 1.

We use $\delta_2(v) = \delta(v) \setminus \delta_1(v)$ and obtain

$$\mathtt{wt}_v \leq \sum_{e \in \delta(v)} \beta_{e,v} \leq \sum_{e \in \delta_1(v)} (\alpha_e - \gamma_v) + \sum_{e \in \delta_2(v)} \alpha_e \leq \sum_{e \in \delta(v)} \alpha_e - 2B_v\gamma_v \tag{11}$$

where the first inequality uses (10), the second inequality uses (8) and (9), and the last inequality follows from the fact that v is incident to at least $2B_v$ edges whenever γ_v is increased.

Summing (11) over all $v \in \mathcal{C}$ gives

$$\sum_{v \in \mathcal{C}} \mathtt{wt}_v \leq \sum_{e \in E} |e \cap \mathcal{C}| \cdot \alpha_e - 2 \cdot \sum_{v \in \mathcal{C}} B_v\gamma_v. \tag{12}$$

Now observe that we raise γ_v and ω_v only for tight vertices in our algorithm. Given that the input instance is feasible, the degree of any tight vertex v will eventually drop below $2B_v$. It therefore follows from the algorithm description that any vertex that becomes tight during the execution of the algorithm is eventually included in the vertex cover $\mathcal{C}$. Hence (12) implies

$$\sum_{v \in \mathcal{C}} \mathtt{wt}_v \leq \sum_{e \in E} |e \cap \mathcal{C}| \cdot \alpha_e - 2 \cdot \sum_{v \in V} B_v\gamma_v.$$

The lemma follows from $B_v\gamma_v = \omega_v$ for all $v \in V$ and from the fact that $|e \cap \mathcal{C}| \leq 2$. ■

Lemmas 1 and 2 complete the proof of Theorem 1.

3 A distributed algorithm

In this section we will describe a distributed primal-dual algorithm for `capVC` which uses the ideas developed in Section 2. As before, the algorithm maintains a pair of (infeasible) primal and (feasible) dual solutions at all times. However, these solutions need to be stored in a distributed fashion: each vertex $v \in V$ stores and manipulates

(a) primal variables x_v and $y_{e,v}$ for all $e \in \delta(v)$, and

(b) dual variables γ_v, ω_v, α_e and $\beta_{e,v}$ for all $e \in \delta(v)$.

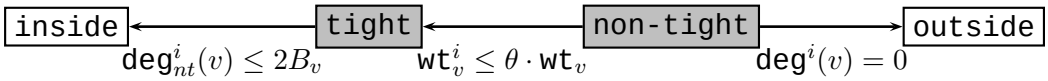

Figure 2: The figure shows the possible states of vertex $v \in V$ in the vertex-selection phase. The arrows indicate the possible transitions between the states. Shaded states are active while others are inactive states.

Note that, for every edge $e = (u, v)$, both u and v store a copy of α_e. The algorithm guarantees the consistency of the two copies.

Observe that a naive distributed implementation of the method described in Section 2 yields an algorithm that needs a linear number of communication rounds in the worst case. In fact, consider a graph G with vertex set $\{v_1, \ldots, v_n, u_1, \ldots, u_{2B}\}$ for some $n, B \geq 1$. Let the edge-set of G be

$$E = \{(v_i, v_{i+1}) \,:\, 1 \leq i \leq n-1\} \cup \{(v_i, u_j) \,:\, 1 \leq i \leq n, 1 \leq j \leq 2B-1\} \cup \{(v_n, u_{2B})\}$$

and notice that vertex v_1 has degree $2B$ while vertices $v_2, \ldots, v_n$ have degree $2B+1$. Let the cost of vertices $v_1, \ldots, v_n$ be 0 and assign a unit cost to all other vertices in G. In the execution of the sequential primal-dual algorithm, all vertices $v_1, \ldots, v_n$ are tight immediately and all other vertices are non-tight. Vertex v_1 is the only tight vertex with degree at most $2B$. After assigning the $2B$ edges in $\delta(v_1)$ to v_1, the degree of vertex v_2 drops to $2B$. In general, the degree of vertex v_i drops to $2B$ after assigning edges to vertices $v_1, \ldots, v_{i-1}$ for all $1 \leq i \leq n$. Doing this in in a distributed fashion takes n communication rounds.

Adapting the algorithm in order to cope with the above synchronization problem is not an easy task. In fact it can be seen that synchronous increase of the duals is at the heart of Lemma 2 where it is used to argue that the dual constraints of type (6) are satisfied with equality at all times.

The distributed algorithm has two main phases:

Vertex-Selection In this phase we compute a vertex cover $\mathcal{C} \subseteq V$ that is $(2+\epsilon)$-approximate. It is here that we solve the above mentioned synchronization problem. While computing an approximate cover, we also assign part of the edges to the vertices in $\mathcal{C}$. At most $2B_v$ edges are assigned to each $v \in \mathcal{C}$.

Edge-Assignment Here, we assign all the remaining edges to the vertices in $\mathcal{C}$. This time, at most $(2+\epsilon)B_v$ edges are assigned to each $v \in \mathcal{C}$.

For ease of presentation we assume from now on that the given `capVC` instance is feasible.

3.1 Vertex-selection phase

As said, the distributed algorithm mimics the primal-dual algorithm from Section 2. Each vertex $v \in V$ stores part of the dual solution and it initially sets $\gamma_v = \omega_v = 0$ and it also lets $\alpha_e = \beta_{e,v} = 0$ for all edges $e \in \delta(v)$.

The distributed algorithm works in rounds. At the beginning of any given round i, we let the *residual weight* $\mathtt{wt}^i_v$ of vertex v be the difference between the right-hand side and the left-hand side of $(7)_v$ for the current feasible dual solution. Thus, we initialize $\mathtt{wt}^0_v$ to $\mathtt{wt}_v$ for all $v \in V$. A vertex $v \in V$ is either *active* or *inactive* in any given round i. An active vertex v can be in one of two states:

`non-tight` Vertex v is `non-tight` whenever the slack in constraint $(7)_v$ is more than $\theta \cdot \mathtt{wt}_v$ for a parameter $\theta \geq 0$ whose exact value will be determined later.

`tight` We let the state of vertex v be `tight` if $\mathtt{wt}_v^i \leq \theta \cdot \mathtt{wt}_v$ and if v has more than $2B_v$ non-tight neighbors. Intuitively, a tight vertex v will eventually be part of the computed cover. It will be assigned a subset of at most $2B_v$ of the edges to its non-tight neighbors.

We will say that an edge $(u, v) \in E$ is active in round i if both u and v are active in that round. For any vertex $v \in V$, we let $\deg_t^i(v)$ and $\deg_{nt}^i(v)$ be the number of its tight and non-tight neighbors in round i. We also let $\deg^i(v) = \deg_t^i(v) + \deg_{nt}^i(v)$ be the active neighbors of v in round i. An inactive vertex v can be in one of two states:

`inside` A `tight` vertex switches its state to `inside` if the number $\deg_{nt}^i(v)$ of non-tight neighbors is at most $2B_v$. Vertex v will be part of the final cover and we assign all edges between v and any of its non-tight neighbors to vertex v.

`outside` We switch the state of a `non-tight` vertex v to `outside` if it has no `tight` or `non-tight` neighbors. We will later argue that all neighbors of v in G are `inside` in this case.

The vertex-selection phase terminates when no active vertices remain. The resulting vertex cover $\mathcal{C}$ consists of all vertices whose final state is `inside`.

We proceed with a detailed description of round i of the distributed algorithm. The round has two steps:

Step 1**:** All non-tight vertices are dormant. Each tight vertex $v \in V$ counts the number of active non-tight neighbors. If this number is at most $2B_v$ we assign all edges connecting v to non-tight neighbors to v. We also switch v's state to `inside` and let v communicate its state-switch to all active neighbors. At this point each active vertex $v \in V$ knows the number $\deg^i(v)$ of active neighbors in G.

Step 2**:** The behavior of an active vertex $v \in V$ depends on its current state:

v is `non-tight`: If $\mathtt{deg}^i(v)$ is 0 we know that all edges incident to v have been assigned to other vertices. Therefore, we can switch the state of v to `outside`.

On the other hand, assume that v has active neighbors. Raising α_e and $\beta_{e,v}$ uniformly by $\mathtt{wt}_v^i/\mathtt{deg}^i(v)$ for all active edges $e \in \delta(v)$ decreases the residual weight of v to 0. Vertex v *strives* for tightness and therefore proposes to any active neighbor u to raise $\alpha_{(u,v)}$ and also $\beta_{(u,v),v}$ by its *proposal*

$$p_v = \frac{\mathtt{wt}_v^i}{\mathtt{deg}^i(v)}.$$

Consider an active edge $e = (u, v) \in \delta(v)$. We raise α_e and $\beta_{e,v}$ by $\min\{p_u, p_v\}$ and decrease the residual weight $\mathtt{wt}_v^i$ of v by the same amount.

v is `tight`: Notice that step 1 guarantees that v has more than $2B_v$ `non-tight` neighbors. Vertex v receives proposals from all such neighbors and lets p_v be their minimum. Vertex v then sends p_v to all such neighbors.

For all `non-tight` neighbors u of v we increase $\alpha_{(u,v)}$ by p_v. In order to maintain dual feasibility, we cannot increase $\beta_{(u,v),v}$ since v is `tight`. Hence we increase ω_v by $B_v p_v$ and γ_v by p_v.

Observe that `tight` vertices have to wait for the proposals of their non-tight neighbors before making their own proposal. Hence two communication rounds are needed to update all the variables.

We can show that the number of communication rounds needed to complete the vertex-selection phase is small. Recall that W denotes the ratio of largest to smallest vertex weights.

Lemma 3 *The vertex-selection phase ends in $O(\log(nW)/\theta)$ rounds.*

Proof: We use a potential function argument in order to show the bound on the number of communication rounds. For round $j \geq 0$ we define the potential of each vertex $v \in V$ as $\Phi_v^j = \mathtt{wt}_v/\mathtt{deg}^j(v)$ if $\mathtt{deg}^j(v) > 0$ and we let $\Phi_v^j = \mathtt{wt}_{max}$ otherwise. Then let

$$\Phi^j = \min_{v\ \mathtt{non\text{-}tight}} \Phi_v^j.$$

Note that Φ^j is a non-decreasing function of j. In fact, we will show that Φ^j doubles at least every $\lceil 2/\theta \rceil$ rounds. The lemma then follows since $\frac{\mathtt{wt}_{min}}{n} \leq \Phi^j \leq \mathtt{wt}_{max}$ for all rounds j.

Consider any given round i. Let V_i^j be the set of non-tight vertices at the beginning of round j, $j \geq i$, with $\Phi_v^j \leq 2\Phi^i$, i.e.

$$V_i^j = \{v \in V\ :\ \deg^j(v) > 0, \mathtt{wt}_v^j > \theta \cdot \mathtt{wt}_v, \Phi_v^j \leq 2\Phi^i\}.$$

Observe that $V_i^{j+1} \subseteq V_i^j$, since the $\mathtt{wt}_v^j$'s and $\deg^j(v)$'s are non-increasing, while the Φ_v^j's are non-decreasing. Consider any vertex $v \in V_i^i$. We will show that $v \notin V_i^{j'}$ for $j' \geq i + \lceil 2/\theta \rceil$. As a consequence, for any non-tight vertex v, $\Phi_v^{j'} > 2\Phi^i$, and hence $\Phi^{j'} > 2\Phi^i$.

Assume by contradiction that $v \in V_i^{j'}$. Then, by the observation above, $v \in V_i^j$ for any $j \in \{i, i+1, \ldots, j'\}$. Suppose that $w \in V$ is a `non-tight` vertex with the smallest proposal p_w in round j. Recall that $\mathtt{wt}_w^j \geq \theta\, \mathtt{wt}_w$ for `non-tight` vertices. We then have

$$p_{\min,j} = p_w = \frac{\mathtt{wt}_w^j}{\mathtt{deg}^j(w)} \geq \frac{\theta \cdot \mathtt{wt}_w}{\mathtt{deg}^j(w)} \geq \theta \cdot \Phi^j. \tag{13}$$

It follows that the reduction of the residual weight of v in round j is at least

$$\mathtt{deg}^j(v) \cdot p_{\min,j} \geq \mathtt{deg}^j(v) \cdot \theta \cdot \Phi^j \geq \mathtt{deg}^j(v) \cdot \theta \cdot \Phi^i \geq \mathtt{deg}^j(v) \cdot \theta \cdot \Phi_v^j/2 = \theta \cdot \mathtt{wt}_v/2,$$

where the first inequality uses (13) and the third inequality uses the definition of the set V_i^j. Hence

$$\mathtt{wt}_v^{j'} \leq \mathtt{wt}_v - \frac{\theta\, \mathtt{wt}_v}{2} \lceil 2/\theta \rceil \leq 0 \leq \theta \cdot \mathtt{wt}_v,$$

which contradicts the assumption that $v \in V_i^{j'}$. ■

We now prove that the weight of the vertices in $\mathcal{C}$ is small.

Lemma 4 *The total weight of the vertices in $\mathcal{C}$ is at most $\frac{2}{1-\theta}$ times the optimum.*

Proof: Assume that the distributed algorithm finishes after $t \geq 0$ rounds and let $(\alpha, \beta, \gamma, \omega)$ be the final dual. A proof very similar to that of Lemma 1 shows that the dual is indeed feasible. We proceed as in the proof of Lemma 2.

Consider a vertex $v \in \mathcal{C}$ and observe that v must have been `tight` before switching to the `inside` state. Thus $\mathtt{wt}_v^t \leq \theta \mathtt{wt}_v$, and

$$\sum_{e \in \delta(v)} \beta_{e,v} \geq \mathtt{wt}_v(1-\theta). \tag{14}$$

We will now show that:

$$\sum_{e \in \delta(v)} \beta_{e,v} \leq \sum_{e \in \delta(v)} \alpha_e - 2\omega_v. \tag{15}$$

Equation (15) is trivially satisfied if we consider only the steps in which v is `non-tight`. In fact, in these steps $\omega_v = 0$ and $\beta_{e,v} = \alpha_e$, for all $e \in \delta(v)$.

Consider now a step in which v is `tight`. The value of the left-hand side of Equation (15) does not change. If ω_v increases by a quantity $B_v \cdot p_v$, γ_v increases by a quantity p_v. It follows that, for all edges $e = (v, u)$ between v and a `non-tight` neighbor u of v in the current step, the value of α_e also increases by at least p_v. Since there are at least $2B_v$ such neighbors, the right-hand side of (15) cannot decrease.

Let `apx` denote the weight of $\mathcal{C}$. Hence,

$$\mathtt{apx} = \sum_{v \in \mathcal{C}} \mathtt{wt}_v \leq \frac{1}{1-\theta} \sum_{v \in C} \sum_{e \in \delta(v)} \beta_{e,v} \leq \frac{1}{1-\theta} \sum_{v \in C} \sum_{e \in \delta(v)} (\alpha_e - 2\omega_v),$$

where the first inequality uses (14) and the second inequality (15). Since every edge is incident to at most two vertices from $\mathcal{C}$ we have that the right hand-side of the last inequality is bounded by

$$\frac{2}{1-\theta} \left(\sum_{e \in E} \alpha_e - \sum_{v \in V} \omega_v \right).$$

The claim follows by weak-duality. ■

For a given accuracy parameter $\epsilon \geq 0$ we now let $\theta = 1 - 2/(2+\epsilon)$. Note that this choice implies that $1/\theta = O(1/\epsilon)$ for $\epsilon \in (0, 1]$. Hence, given a feasible instance of `capVC` our distributed algorithm terminates within $O(\log(nW)/\epsilon)$ communication rounds with a cover of weight at most $(2+\epsilon)\mathtt{opt}$ as was claimed in Theorem 2.

3.2 Edge-assignment phase

At the end of the vertex-selection phase we are left with a subset $\mathcal{C}' \subseteq \mathcal{C}$ of the tight vertices such that all unassigned edges have both their end-points in $\mathcal{C}'$. In the following we let $G^0 = G[\mathcal{C}'] = (V, E)$ be the graph induced by the vertices in $\mathcal{C}'$. Assuming that the given `capVC` instance is feasible, there must be an assignment of the edges in G^0 to the vertices in $\mathcal{C}'$ that obeys the original capacity bounds. We describe a deterministic distributed algorithm which assigns at most $(2+\epsilon)B_v$ edges to each $v \in \mathcal{C}'$ in $O(\log n/\epsilon)$ rounds.

Our algorithm starts with all edges unassigned and computes a final assignment iteratively. In each round t we consider all vertices $v \in V$ with at most $(2+\epsilon)B_v$ incident unassigned edges, and we assign all such edges $(u, v) \in \delta(v)$ to v. We continue until no unassigned edges remain.

To prove that the number of rounds is polylogarithmic we need the following lemma. Let H be the set of vertices with degree more than $(2+\epsilon)B_v$ and let $E(H)$ be the set of those edges that have both of their endpoints in H. Finally use $\overline{E(H)}$ as an abbreviation for the set $E \setminus E(H)$ of edges that have at most one endpoint in H.

Lemma 5 *If there is a feasible assignment, then we must have* $|\overline{E(H)}| \geq \epsilon |E(H)|$.

Proof: Let $\pi : E \to V$ be a feasible assignment of edges to vertices. We have that:

$$\sum_{v \in H} |\delta(v)| \leq 2|E(H)| + |\overline{E(H)}| \tag{16}$$

as every edge in $E(H)$ is counted exactly twice in the sum on the left-hand side while an edge in $\overline{E(H)}$ is counted at most once. Moreover,

$$|E(H)| \leq \sum_{v \in H} |\pi^{-1}(v)| \tag{17}$$

since every edge in $E(H)$ must be assigned to some vertex in H. From equations (16) and (17) it follows that:

$$(2+\epsilon) \sum_{v \in H} B_v \leq \sum_{v \in H} |\delta(v)| \leq 2 \sum_{v \in H} |\pi^{-1}(v)| + |\overline{E(H)}| \leq 2 \sum_{v \in H} B_v + |\overline{E(H)}|.$$

Hence

$$|\overline{E(H)}| \geq \epsilon \sum_{v \in H} B_v \geq \epsilon |E(H)|$$

which proves the lemma. ■

Lemma 6 *If there is a feasible assignment, then the algorithm above assigns at most* $(2+\epsilon)B_v$ *edges to each* $v \in V$. *The number of rounds required is* $O(\log n/\epsilon)$.

Proof: The capacity bound in the theorem follows immediately since for each vertex v in V there is at most one round t in which we assign at most $(2+\epsilon)B_v$ edges to it.

Let E^t be the set of unassigned edges at the beginning of iteration t and let $G^t = G[E^t]$ be the subgraph of G induced by E^t. We also use H^t to denote the set of vertices $v \in V$ whose degree is more than $(2+\epsilon)B_v$ in G^t. Note that for any t, there must exist a feasible assignment in G^t as G^t is a subgraph of the initial graph G where a feasible assignment exists. So we can apply Lemma 5 and conclude that:

$$|E^t| = |\overline{E(H^t)}| + |E(H^t)| \geq (1+\epsilon)|E(H^t)|.$$

In round t all the edges in $\overline{E(H^t)}$ are assigned to some vertex and so $|E^{t+1}| \leq |E(H^t)|$. Hence, $|E^{t+1}| \leq \frac{1}{1+\epsilon}|E^t|$ and the number of unassigned edges decreases by a factor of $(1+\epsilon)$ in every round. ■

Since at most $2B_u$ edges are assigned to each u during the vertex-selection phase, this concludes the proof of Theorem 2.

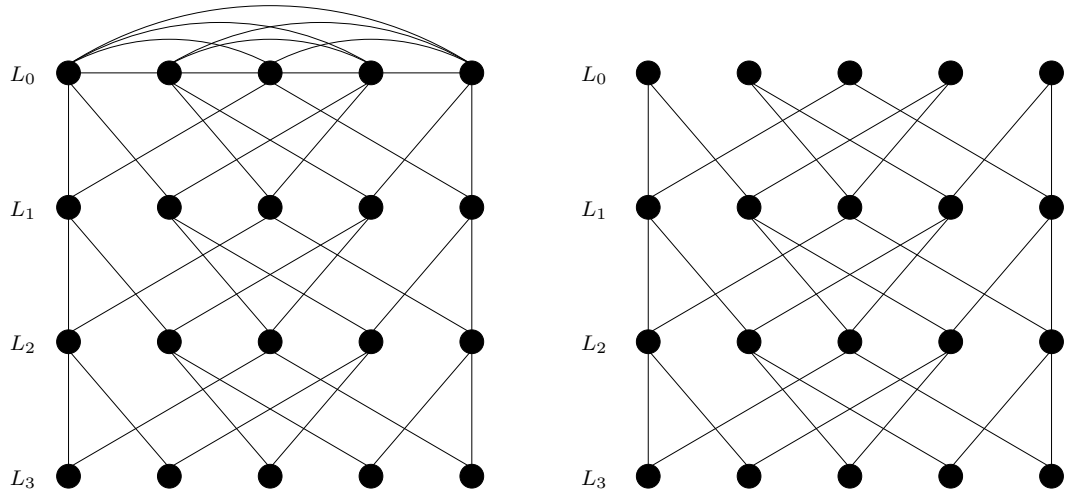

Figure 3: The figure shows graphs $G^1_{B,k}$ (on the left) and $G^0_{B,k}$ for $B = 2$ and $k = 3$.

4 A Lower Bound

In this section we show that every efficient distributed approximation algorithm for `capVC` needs to violate the capacity constraints by a large additive term.

Consider the following two families of graphs $G^0_{B,k}$ and $G^1_{B,k}$, where $B, k \geq 1$. Graph $G^0_{B,k}$ has $k+1$ levels $L_0, L_1 \ldots L_k$, each one containing $2B + 1$ vertices. Each vertex in level L_i, $i = 0, 1 \ldots k-1$, is adjacent to exactly B vertices in level L_{i+1}. Symmetrically, each vertex in level L_i, $i = 1, 2 \ldots k$, is adjacent to exactly B vertices in level L_{i-1}. There are no other edges in the graph. In particular, each level L_i induces an independent set. Graph $G^1_{B,k}$ is obtained from $G^0_{B,k}$ by adding an edge between each pair of vertices in L_0. Let the capacity of all vertices in both graphs be B. Moreover, all vertices have cost zero, except for the vertices in level L_k, which have cost one. Figure 3 shows an instance of the two graphs.

For $0 \leq i \leq k-1$, let δ_i be the set of edges that connect vertices in L_i to those in L_{i+1}. We obtain a feasible solution for $G^0_{B,k}$ as follows: Let

$$C = L_0 \cup L_1 \cup \ldots \cup L_{k-1}$$

and assign all edges in δ_i to the vertices in L_i for $0 \leq i \leq k-1$.

Graph $G^1_{B,k}$ has $n = (k+1)(2B+1)$ vertices and Bn edges. Thus, any feasible capacitated vertex cover must contain all vertices. Moreover, the edges belonging to the clique on L_0 clearly have to be assigned to the vertices in L_0. Thus, the unique feasible `capVC` solution for $G^0_{B,k}$ assigns all edges in δ_i to the vertices in L_{i+1} for all $0 \leq i \leq k-1$.

The following lemma turns out to be useful in the proof of the lower bound.

Lemma 7 *Consider a solution for $G^1_{B,k}$ that assigns at most $(B + c)$ edges to each vertex, for some $c \geq 1$. For $0 \leq i \leq k-1$, let A_i be the number of edges in δ_i that are assigned to vertices in L_i. Then*

$$A_i \leq (2B+1)(i+1)c$$

for all $0 \leq i \leq k-1$.

Proof: The proof is by induction on i. For $i = 0$, all clique edges need to be assigned to vertices in L_0. The spare capacity of the vertices in L_0 is thus $(2B + 1)c$ and this is the maximum number of edges in δ_0 that can be assigned to the vertices in L_0.

Now assume that the claim is true for all $0 \leq i < k$. Using the induction hypothesis, at most $(2B+1)(i+1)c$ edges in δ_i are assigned to the vertices in L_i. Therefore, the remaining $(2B + 1)(B - (i + 1)c)$ edges need to be assigned to vertices in L_{i+1}. The remaining capacity of the vertices in L_{i+1} is thus

$$(2B + 1)(B + c) - (2B + 1)(B - (i + 1)c) = (2B + 1)(i + 2)c$$

and this is the maximum number of edges in δ_{i+1} that can be assigned to vertices in L_{i+1}. ■

Armed with the above lemma we are now ready to provide a proof of Theorem 3. We restate it here for completeness.

Theorem 3 *Let $B, k \geq 1$ be integer parameters. There is a* capVC *instance with uniform vertex capacities B, for which any distributed approximation algorithm that assigns less than $(1 + 1/k) \cdot B$ edges to all vertices, must take at least k communication rounds.*

Proof: The proof is by contradiction. Let $c < B/k$ and consider a distributed approximation algorithm for capVC that assigns at most $(B + c)$ edges to each vertex for any given (uniform capacity) instance whose running time is less than k.

We first execute this algorithm on graph $G^1_{B,k}$. Lemma 7 shows that at most $(2B + 1) \cdot kc$ of the edges in δ_{k-1} are assigned to the vertices in L_{k-1}. The number of edges that need to be assigned to vertices in L_k is therefore at least

$$(2B + 1)(B - kc) > 0$$

where the inequality uses our assumption on c. Hence at least one vertex in L_k needs to be in any cover of $G^1_{B,k}$ that assigns at most $B + c$ edges to each vertex. Let u be this vertex.

We now run the algorithm again on $G^0_{B,k}$. Since the graphs $G^0_{B,k}$ and $G^1_{B,k}$ are identical up to distance k from u, this vertex will be included in the cover in this case too. On the other hand, no vertex in L_k can be part of any approximate solution for $G^0_{B,k}$. ■

For instance, consider an (efficient) distributed approximation algorithm for capVC with running time $O(\log^d n)$, where d is a positive constant. Theorem 3 then shows that there is a family of (uniform capacity) capVC instances for which this algorithm must assign at least $B + \Omega(\frac{B}{\log^d B})$ edges to some vertex.

We observe that graphs $G^0_{B,k}$ and $G^1_{B,k}$ have $n = (2B + 1)(k + 1)$ vertices. This implies that $B = \Theta\left(\frac{n}{k}\right)$, which is large in the interesting case when k is polylogarithmic. However, this *hitch* is easily removed by defining a graph G_0 (resp. G_1) consisting of t disjoint copies of the main building block $G^0_{B,k}$ (resp. $G^1_{B,k}$). Using the new parameter t we can now produce instances in which B is arbitrarily small in comparison to n while our proof argument goes through unchanged.

Acknowledgments We would like to thank Volker Kaibel for pointing out a much simplified proof of Lemma 5.

References

[1] N. Alon, D. Moshkovitz, and S. Safra. Algorithmic construction of sets for k-restrictions. *ACM Trans. on Algorithms*, 2(2):153–177, 2006.

[2] R. Bar-Yehuda and S. Even. A local-ratio theorem for approximating the weighted vertex cover problem. *Annals of Discrete Mathematics*, 25:27–45, 1985.

[3] M. Bellare, S. Goldwasser, C. Lund, and A. Russell. Efficient probabilistically checkable proofs: Applications to approximation. In *Proceedings, ACM Symposium on Theory of Computing*, pages 294–304, 1993.

[4] F. Chudak, T. Erlebach, and A. Panconesi. Primal-dual based distributed algorithms for vertex cover with soft capacities and facility location. *Manuscript*, 2004.

[5] J. Chuzhoy and J. Naor. Covering problems with hard capacities. In *Proceedings, IEEE Symposium on Foundations of Computer Science*, pages 481–489, 2002.

[6] D. Dubhashi, O. Häggström, G. Mambrini, A. Panconesi, and C. Petrioli. Blue pleieades, a new solution for device discovery and scatternet formation in multi-hop bluetooth networks. *ACM-Kluwer Wireless Networks (Winet)*, 13(1):107–125, 2007.

[7] M. Elkin. Unconditional lower bounds on the time-approximation tradeoffs for the distributed minimum spanning tree problem. In *Proceedings, ACM Symposium on Theory of Computing*, pages 331–340, 2004.

[8] U. Feige. A threshold of ln n for approximating set cover. *J. ACM*, 45(4):634–652, 1998.

[9] R. Gandhi, E. Halperin, S. Khuller, G. Kortsarz, and A. Srinivasan. An improved approximation algorithm for vertex cover with hard capacities (extended abstract). In *Proceedings, International Colloquium on Automata, Languages and Programming*, pages 164–175, 2003.

[10] R. Gandhi, S. Khuller, S. Parthasarathy, and A. Srinivasan. Dependent rounding in bipartite graphs. In *Proceedings, IEEE Symposium on Foundations of Computer Science*, pages 323–332, 2002.

[11] D. Grable and A. Panconesi. Nearly optimal distributed edge colouring in o(log log n) rounds. *Random Structures and Algorithms*, 10(3):385–405, 1997.

[12] S. Guha, R. Hassin, S. Khuller, and E. Or. Capacitated vertex covering. *J. Algorithms*, 48(1):257–270, 2003.

[13] E. Halperin. Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs. *SIAM J. Comput.*, 31(5): 1608–1623, 2002.

[14] D. S. Hochbaum. Approximation algorithms for set covering and vertex cover problems. *SIAM J. Comput.*, 11:555–556, 1982.

[15] G. Karakostas. A better approximation ratio for the vertex cover problem. In *Proceedings, International Colloquium on Automata, Languages and Programming*, pages 1043–1050, 2005.

[16] S. Khuller, U. Vishkin, and N. Young. A primal-dual parallel approximation technique applied to weighted set and vertex covers. *J. Algorithms*, 17(2):280–289, 1994.

[17] F. Kuhn, T. Moscibroda. Distributed Approximation of Capacitated Dominating Sets. In *Proceedings, ACM Symposium on Parallelism in Algorithms and Architectures*, pages 161–170, 2007.

[18] F. Kuhn, T. Moscibroda, and R. Wattenhofer. What cannot be computed locally! In *Proceedings, ACM Symposium on Principles of Distributed Computing*, pages 300–309, 2004.

[19] F. Kuhn and R. Wattenhofer. Constant-time distributed dominating set approximation. In *Proceedings, ACM Symposium on Principles of Distributed Computing*, pages 25–32, 2003.

[20] L. Jia, R. Rajaraman and T. Suel. An efficient distributed algorithm for constructing small dominating sets. *Distributed Computing*, 15(4):193-205, 2002.

[21] N. Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21(1):193–201, 1992.

[22] M. Luby. A simple parallel algorithm for the maximal independent set problem. *SIAM J. Comput.*, 15:1036–1053, 1986.

[23] M. Luby. Removing randomness in parallel without processor penalty. *J. Comput. System Sci.*, 47(2):250–286, 1993.

[24] C. Lund and M. Yannakakis. On the hardness of approximating minimization problems. *J. ACM*, 41(5):960–981, 1994.

[25] M. Naor. A lower bound on probabilistic algorithms for distributive ring coloring. *SIAM J. Discrete Math.*, 4(3):409-412, 1991.

[26] P. Wan, K. M. Alzoubi and O. Frieder Distributed construction of connected dominating set in wireless ad hoc networks. *Proceedings of INFOCOM*, 2002.

[27] A. Panconesi and A. Srinivasan. The local nature of delta-coloring and its algorithmic applications. *Combinatorica*, 15(2):255–280, 1995.

[28] S. Rajagopalan and V. Vazirani. Primal-dual RNC approximation algorithms for (multi)set (multi)cover and covering integer programs. *SIAM J. Comput.*, 28(2):525–540, 1998.

[29] R. Rajaraman. Topology control and routing in ad hoc networks: a survey. *SIGACT News*, 2002.

[30] R. Raz and S. Safra. A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP. In *Proceedings, ACM Symposium on Theory of Computing*, pages 475–484, 1997.