Primal-Dual Based Distributed Algorithms for Vertex Cover with Semi-Hard Capacities

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

October 25, 2004

Abstract

In this paper we consider the weighted, capacitated vertex cover problem with hard capacities ($\mathtt{capVC}$). Here, we are given an undirected graph $G = (V, E)$, non-negative vertex weights $\mathtt{wt}_v$ for all vertices $v \in V$, and node-capacities $B_v \geq 1$ for all $v \in V$. A feasible solution to a given $\mathtt{capVC}$ instance consists of a vertex cover $\mathcal{C} \subseteq V$. Each edge $e \in E$ is assigned to one of its endpoints in $\mathcal{C}$ and the number of edges assigned to any vertex $v \in \mathcal{C}$ is at most B_v. The goal is to minimize the total weight of $\mathcal{C}$.

For a parameter $\epsilon > 0$ we give a deterministic, distributed algorithm for the $\mathtt{capVC}$ problem that computes a vertex cover $\mathcal{C}$ of weight at most $(2 + \epsilon) \cdot \mathtt{opt}$ where $\mathtt{opt}$ is the weight of a minimum-weight feasible solution to the given instance. The number of edges assigned to any node $v \in \mathcal{C}$ is at most $(4 + \epsilon) \cdot B_v$. The running time of our algorithm is $O(\log(nW)/\epsilon)$, where n is the number of nodes in the network and $W = \mathtt{wt}_{max}/\mathtt{wt}_{min}$ is the ratio of largest to smallest weight, respectively.

This result is complemented by a lower-bound saying that any distributed algorithm for the $\mathtt{capVC}$ problem that obeys capacity bounds exactly needs to spend $\Omega(n)$ communication rounds to compute an approximate solution. The main feature of the algorithm is that it is derived in a systematic fashion starting from a primal-dual sequential algorithm.

*Max Planck Institut für Informatik, Saarbrücken, Germany 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 e Sistemistica at Università di Roma "La Sapienza", Italy.

‡Informatica, La Sapienza di Roma, via Salaria 113, 00198, Roma, Italy Email: `ale@dsi.uniroma1.it`.

§Informatica, La Sapienza di Roma, via Salaria 113, 00198, Roma, Italy Email: `sozio@di.uniroma1.it`.

1 Introduction

The *capacitated vertex cover* problem ($\texttt{capVC}$) is the variant of vertex cover in which there is a limit to the number of edges that a vertex can cover. This kind of limitation is quite natural in many scenarios, including network applications. A precise formulation of the problem is as follows. We are given an n-node undirected graph $G = (V, E)$, non-negative weights $\texttt{wt}_v$ and parameters $b_v \geq 1$ for all vertices $v \in V$. Moreover, we have node-capacities $B_v \geq 1$ for all nodes $v \in V$. A solution to a given $\texttt{capVC}$ instance consists of an integer vector $\{x_v\}_{v \in V}$ and an assignment $\pi : E \rightarrow C$ of edges to nodes such that

1. $0 \leq x_v \leq b_v$ for all nodes $v \in V$,
2. $\pi(e) \in \{u, v\}$ for all edges $e = (u, v) \in E$, and
3. $|\pi^{-1}(v)| \leq B_v \cdot x_v$ for all $v \in C$.

The cost of the solution is $\sum_{v \in V} x_v \texttt{wt}_v$ and the goal is to minimize it. The first set of constraints specifies the maximum number of copies of each vertex that can be opened. The second says that every edge must be covered by some vertex. The third set, the *capacity constraints*, specifies the maximum number of edges that each vertex can cover.

Several relaxations of the problem have been considered in the literature. We consider the hard-capacity version of the problem where at most one copy of each node is opened. For a parameter $\epsilon > 0$ our algorithms computes a vertex cover $\mathcal{C}$ of weight at most $(2 + \epsilon) \cdot \texttt{opt}$. We give a distributed algorithm that assigns each edge $e \in E$ to one of its endpoints in $\mathcal{C}$ in such a way that $|\pi^{-1}(v)| \leq (4 + \epsilon) \cdot B_v$ for all $v \in \mathcal{C}$. A sequential version of this algorithm assigns at most $(2 + \epsilon) \cdot B_v$ edges to each node. Note that our algorithms may assign more than B_v edges to a node in the cover; hence the term *semi-hard*.

In the synchronous, message-passing model of computation the algorithm takes $O(\log(nW)/\epsilon)$ many rounds, where W is the maximum weight among all nodes. This reduces to $O(\log n/\epsilon)$ for the interesting case of unit weights. These results are complemented by strong lower-bounds and by (previously known) hardness of approximation results. Our algorithms are deterministic, while typically efficient distributed algorithms for graph problems require randomization.

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 [9]. Although described for the PRAM setting, the algorithm can be easily adapted to the distributed case. Although not exactly based on a primal-dual algorithm, recent work on the dominating set problem showed more applications of LP-duality to the design of distributed algorithms [10]. This paper takes the primal-dual approach pioneered in [9] one step further, giving a new and non trivial example of what could become a powerful methodology for the design and analysis of distributed algorithms.

Previous work. The capacitated vertex cover problem was first introduced by Guha et al. [6] who consider the *soft*-capacity version of the problem ($\texttt{capVC}_s$) where $b_v = \infty$ for all $v \in V$. Intuitively this means that a vertex can make as many copies of itself as is needed to cover all the edges. Guha et al. first present a simple 4-approximate LP-rounding based algorithm. Later on, the authors show a 2-approximate primal-dual algorithm. Subsequently, Gandhi et al. [5] present a 2-approximate LP-rounding algorithm for $\texttt{capVC}_s$.

The capacitated vertex-cover problem becomes much harder once we allow $b_v < \infty$ for nodes $v \in V$. Chuzhoy and Naor [2] give a sophisticated 3-approximate LP-rounding algorithm for the special case of capVC with uniform vertex weights. Finally, in [4], Gandhi et al. give an LP-rounding-based 2-approximation algorithm for capVC with uniform weights.

In [2], Chuzhoy and Naor also show that capVC in the presence of non-uniform vertex weights is as hard to approximate as set-cover. Feige [3] shows that it is impossible to obtain better than a $(1+o(1))\log(n)$-approximation for the set-cover problem unless $\mathbf{NP} \subseteq \mathtt{DTIME}(\mathbf{n}^{\log\log \mathbf{n}})$.

The best known approximation algorithm for the vertex-cover problem without capacities is due to Halperin [7] and achieves a performance ratio of $(2 - o(1))$ for general graphs. Earlier 2-approximations are due to Bar-Yehuda and Even [1] and Hochbaum [8]. As mentioned, the same bound is essentially achievable in a distributed setting [9].

Our contribution. The first result we give is a polynomial-time primal-dual approximation algorithm for semi-hard constraints.

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 solution $(\{x_v\}_{v\in V}, \pi)$ such that*

$$|\pi^{-1}(v)| \leq 2B_v$$

for all $v \in V$ and $\sum_{v\in V} \mathtt{wt}_v x_v \leq 2 \cdot \mathtt{opt}$ where opt *is the weight of an optimum feasible solution.*

We remark that if the input instance does not have a feasible solution then our algorithm terminates with a certificate of infeasibility. The above result is interesting in view of the following lower bound proven in [2]: capVC with hard capacities and polynomially large weights is as hard to approximate as set cover. Therefore, unless $\mathbf{NP} \subseteq \mathtt{DTIME}(\mathbf{n}^{\log\log \mathbf{n}})$, if one is looking for constant approximation, capacity constraints must be violated.

Besides being interesting in its own right, Theorem 1 is a natural step toward proving the main result of this paper. We use $\mathtt{wt}_{min}$ and $\mathtt{wt}_{max}$ to denote the minimum and maximum vertex weights in the given capVC instance and let $W = \mathtt{wt}_{max}/\mathtt{wt}_{min}$ be their ratio.

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 which computes a solution $(\{x_v\}_{v\in V}, \pi)$ such that*

$$|\pi^{-1}(v)| \leq (4 + \epsilon)B_v$$

for all $v \in V$ and $\sum_{v\in V} \mathtt{wt}_v x_v \leq (2+\epsilon) \cdot \mathtt{opt}$ where opt *is the weight of an optimum feasible solution. The algorithm needs $O(\log(nW)/\epsilon)$ rounds.*

Note that the running time is strongly poly-logarithmic for polynomially large weights only. This includes the important special case of unit weights. Obtaining a strongly poly-logarithmic 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 terminates with a certificate of infeasibility. This 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 strong lower-bound on the communication complexity of any algorithm for the weighted `capVC` problem with hard capacities:

Theorem 3 *For any $\alpha > 0$, any distributed α-approximation algorithm for* `capVC` *with polynomially large weights requires $\Omega(n)$ many communication rounds.*

This result shows that violating the capacity constraints is necessary, since our distributed algorithms produce a $(2+\epsilon)$-approximation. Together with the set cover hardness of [2] this shows that the approximation factors achieved by our distributed algorithm are best possible, modulo constants.

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 modify 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 the appendix.

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 $y_{e,w} = 1$ iff $\pi(e) = w$. In the following let $\delta(v)$ be the set of edges incident to node $v \in V$ in G.

$$\min \quad \sum_{v \in V} \mathtt{wt}_v \cdot x_v \tag{IP}$$

$$\text{s.t} \quad y_{e,v} + y_{e,u} \geq 1 \qquad \forall e = (u,v) \in E \tag{1}$$

$$y_{e,v} \leq x_v \qquad \forall e = (u,v) \in E \tag{2}$$

$$\sum_{e=(v,u) \in \delta(v)} y_{e,u} \leq B_v \cdot x_v \qquad \forall v \in V \tag{3}$$

$$y_{e,v}, x_v \in \{0,1\} \qquad \forall e \in E, v \in V \tag{4}$$

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

$$y_{e,v} \geq 0 \qquad \forall e = (u,v) \in E$$

$$0 \leq x_v \leq 1 \qquad \forall v \in V \tag{5}$$

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 a variable α_e with constraint $(1)_e$ for all $e \in E$, $\beta_{e,v}$ for constraint $(2)_{e,v}$ for all edges $e = (u, v) \in E$, γ_v for $(3)_v$ for all $v \in V$, and ω_v for the upper-bound constraints $(5)_v$ for all $v \in V$. The linear programming dual of (LP) is then

$$\max \quad \sum_{e \in E} \alpha_e - \sum_{v \in V} \omega_v \tag{D}$$

$$\text{s.t} \quad \alpha_e \leq \beta_{e,w} + \gamma_w \qquad \forall e = (u, v) \in E, \forall w \in \{u, v\} \tag{6}$$

$$\sum_{e=(u,v) \in E} \beta_{e,v} \leq \mathtt{wt}_v + (\omega_v - B_v \cdot \gamma_v) \qquad \forall v \in V \tag{7}$$

$$\alpha, \beta, \gamma, \omega \geq 0$$

2.2 The algorithm

A primal-dual algorithm is an algorithm that starts with a feasible dual solution and an infeasible primal one. Throughout its execution such an algorithm improves the dual-objective function value of the kept dual solution and it reduces the *degree of infeasibility* of the primal one at the same time. The algorithm terminates as soon as the primal solution is feasible. The final dual solution is used as a lower-bound for the optimum solution value by means of weak duality.

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. We also say that edge $e = (u, v) \in E$ is w-tight for $w \in \{u, v\}$ if $(6)_{e,w}$ is satisfied with equality. Finally, for a node $v \in V$ let $\mathtt{deg}(v)$ be its degree in G, i.e. $\mathtt{deg}(v) = |\delta(v)|$.

We start with a dual feasible solution and let

$$\alpha = \beta = \gamma = \omega = 0.$$

For a current feasible dual solution we let $\mathcal{O} \subseteq V$ be the set of tight vertices. We denote the final vertex cover by $\mathcal{C}$. Initially all edges in the graph are *unassigned* (i.e., $y_{e,w} = 0$ for all $w \in e$) and $\mathcal{O}$ and $\mathcal{C}$ are empty.

The initial part of our primal-dual algorithm resembles a standard primal-dual algorithm for the vertex-cover problem (e.g., see [1, 8]). The algorithm raises variables α_e for all edges $e \in E$ uniformly at unit rate. In order to maintain dual feasibility we also have to raise $\beta_{e,w}$ for edges $e \in E$ that are w-tight. Notice that this is only possible if w itself is non-tight.

Once a node $v \in V$ becomes tight we distinguish two cases depending on the degree of v in G:

$\mathtt{deg}(v) \leq 2B_v$ Assign all edges $e \in \delta(v)$ to v and subsequently delete v and $\delta(v)$ from G. Add v to $\mathcal{C}$, delete isolated non-tight vertices from G and continue.

$\mathtt{deg}(v) > 2B_v$ We can not assign the edges in $\delta(v)$ to v since this would violate the (relaxed) capacity constraints of vertex v. Our algorithm maintains a set of vertices $\mathcal{H} \subseteq \mathcal{O}$ such that $u \in \mathcal{H}$ iff $\mathtt{deg}(u) > 2B_v$. We therefore add v to $\mathcal{H}$.

For all $v \in \mathcal{H}$ we raise ω_v at a rate of B_v and γ_v at unit rate. Notice this implies that node v remains tight since the right-hand side of $(7)_v$ does not change.

Also observe that variables $\beta_{e,v}$ cannot be raised further without rendering our dual solution infeasible. On the other hand the right-hand side of constraint $(6)_{e,v}$ for $e \in \delta(v)$ increases at unit rate due to the increase in γ_v. Hence, for $e \in \delta(v)$ we can continue to raise α_e at unit rate without increasing $\beta_{e,v}$.

The algorithm stops when all edges have been assigned.

2.3 Analysis

In this section we present a proof of Theorem 1. In the following we assume that the algorithm from Section 2 terminates and that it outputs a cover $\mathcal{C}$ together with an assignment $\{y_{e,v}\}_{e\in E, v\in V}$ as well as a dual solution $(\alpha, \beta, \omega, \gamma)$.

We first show that the computed dual solution is feasible.

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 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*

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

Proof: Assume first, for the sake of contradiction, that our primal-dual algorithm does not terminate. It is then not hard to see that the algorithm must reach a point in the execution, where each node $v \in \mathcal{O}$ has degree more than $2B_v$. All remaining unassigned edges have both endpoints in $\mathcal{O}$. Using the pigeon-hole principle it follows that, in any assignment of edges to nodes, there must be at least one node $v \in \mathcal{O}$ that has been assigned more than B_v edges. In other words, the given input instance is infeasible.

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 node 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. Therefore, for all $e \in \delta_1(v)$ we must have

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

Since $v \in \mathcal{O}$ at deletion time 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

$$\mathtt{wt}_v = \sum_{e\in\delta(v)} \beta_{e,v} \le \left(\sum_{e\in\delta_1(v)} \alpha_e - \gamma_v\right) + \sum_{e\in\delta_2(v)} \alpha_e \le \left(\sum_{e\in\delta(v)} \alpha_e\right) - (2B_v+1)\gamma_v \tag{11}$$

where the first inequality uses (10), the second inequality uses (8) and (9), and the last inequality follows from the fact that $\delta_1(v)$ is large.

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

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

Now observe that we raise γ_v and ω_v only for tight vertices in our algorithm and all tight vertices are eventually included in $\mathcal{C}$. Hence (12) implies

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

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

Lemmas 1 and 2 complete the proof of Theorem 1.

3 A distributed algorithm for capVC

In this section we present a distributed deterministic algorithm for capVC and provide a proof for Theorem 2. The algorithm is based on the ideas presented in Section 2. The main problem in giving a distributed implementation of the sequential primal-dual algorithm from the last section is the way we increase the dual variables.

Consider for instance a path of tight nodes $P = \langle u_0 u_1 \ldots u_\ell \rangle$. Suppose that u_0 and u_ℓ have two non-tight neighbors a and b, respectively, which raise their variables $\beta_{(a,u_0),a}$ and $\beta_{(b,u_\ell),b}$. To match these proposals

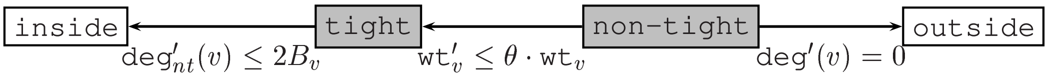

Figure 1: The figure shows the possible states of node $v \in V$ in the node-selection phase. The arrows indicate the possible transitions between the states. Shaded states are active while others are inactive states.

the variables γ_{u_0} and γ_{u_ℓ} must be raised, and if $\beta_{(a,u_0),a} \neq \beta_{(b,u_\ell),b}$ this requires synchronization along the path P, something that requires linear time.

Ridding the algorithm of this need for synchronization 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:

Node-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 nodes 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 nodes in $\mathcal{C}$. Again, 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 Node-selection phase

As said, the goal in the node-selection phase is to find a vertex cover $\mathcal{C}$. A node $v \in V$ can be in one of four states: `non-tight`, `tight`, `inside`, and `outside`. A node v is called *active* when v's state is either `non-tight` or `tight`. The phase terminates when no active node remains and the final vertex cover $\mathcal{C}$ consists of all nodes whose state is `inside` at this point. Refer to Figure 1 for an illustration of the node-states and the transitions between them.

The distributed algorithm mimics the primal-dual algorithm from Section 2. For the purpose of analysis and in order to motivate the algorithm we will let the algorithm construct a feasible dual solution for (D). Initially, we let

$$\alpha_e = \beta_{e,v} = \gamma_v = \omega_v = 0$$

for all $e \in E$ and for all $v \in V$. This dual solution is clearly feasible for (D).

Our distributed algorithm works in rounds. At the beginning of any given round i, we let the *residual weight* $\mathtt{wt}^i_v$ of node v be the difference between right-hand side and 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 $v \in V$.

An active node $v \in V$ is `non-tight` in round i if and only if its residual weight $\mathtt{wt}^i_v$ is more than $\theta \cdot \mathtt{wt}_v$ for some positive threshold θ whose value will be fixed later (we will eventually choose θ such that $1/\theta = O(1/\epsilon)$). Once the residual weight of an active node v drops below its threshold $\theta \cdot \mathtt{wt}_v$, node v becomes `tight`.

Nodes in *inactive* states (i.e., their state is either $\texttt{inside}$ or $\texttt{outside}$) are passive in the algorithm. Round i consists of two steps.

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

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

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

On the other hand, assume that v has active neighbors. Raising α_e and $\beta_{e,v}$ uniformly by $\texttt{wt}_v^i/\texttt{deg}^i(v)$ for all active edges $e \in \delta(v)$ decreases the residual weight of v to 0. Node 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*

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

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 $\texttt{wt}_v^i$ of v by the same amount.

[v **is** $\texttt{tight}$**]** Notice that step 1 guarantees that v has more than $2B_v$ $\texttt{non-tight}$ neighbors. Node v receives proposals from all such neighbors and lets p_v be their minimum. Node v then sends p_v to all such neighbors. Note that $\texttt{tight}$ nodes do not send proposals to each other.

For all $\texttt{non-tight}$ 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 $\texttt{tight}$. Hence we increase ω_v by $B_v p_v$ and γ_v by p_v.

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

Lemma 3 *The node-selection phase ends in* $O(\log(nW)/\epsilon)$ *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 $\Phi_v^j = \texttt{wt}_v/\texttt{deg}^j(v)$ for all $\texttt{non-tight}$ nodes $v \in V$. Then let

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

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

Suppose that $v \in V$ is a $\texttt{non-tight}$ node with the smallest proposal p_v in round j. We then have

$$p_{\min} = p_v = \frac{\texttt{wt}_v^j}{\texttt{deg}^j(v)} \geq \frac{\theta \cdot \texttt{wt}_v}{\texttt{deg}^j(v)} \geq \theta \cdot \Phi^j. \tag{13}$$

For rounds $0 \leq i \leq j$ we let V_i^j be the set of $\texttt{non-tight}$ nodes at the beginning of round j with $\Phi_v^j \leq 2\Phi^i$:

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

Consider a node $v \in V_i^j$. The reduction in its residual weight in round j is at least

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

where the first inequality uses (13) and the third inequality uses the definition of the set V_i^j. Therefore, a node $v \in V_i^i$ either leaves V_i^i or becomes tight within $\lceil 2/\theta \rceil$ rounds. ■

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

Lemma 4 *The total weight of the nodes in $\mathcal{C}$ is at most $(2+\epsilon)$ 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 node $v \in \mathcal{C}$ 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

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

We will now show that:

$$\sum_{e \in \delta(v)} \beta_{e,v} \leq \sum_{e \in \delta(v)} \alpha_e - 2\omega_v. \tag{14}$$

Equation (14) 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 (14) 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 (14) cannot decrease.

Let `apx` and `opt` denote the weight of $\mathcal{C}$ and that of an optimum solution, respectively. By weak-duality:

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

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 lemma follows by choosing θ such that $\theta = 1 - 2/(2+\epsilon)$. Note that, as required, $1/\theta = O(1/\epsilon)$ for $\epsilon \in (0, 1]$. ■

3.2 Edge-assignment phase

At the end of the node-selection phase we are left with a subset $\mathcal{C}' \subseteq \mathcal{C}$ of the tight nodes such that all unassigned edges have both their end-points in $\mathcal{C}'$. In the following we let $G^0 = G[\mathcal{C}']$ be the graph induced by the nodes 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 poly-logarithmic we need the following lemma. In the following 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: Recall that $\delta(v)$ is the set of edges incident to v. Let $\pi : E \to V$ be a feasible assignment. We have that,

$$\sum_{v \in H} \delta(v) \leq 2|E(H)| + |\overline{E(H)}| \tag{15}$$

since in the sum we count every edge in $E(H)$ twice, but all other edges are counted only once. Moreover,

$$|E(H)| = \sum_{v \in H} |\pi^{-1}(v)| \tag{16}$$

since every edge in $E(H)$ must be assigned to some node in H. From equations (15) and (16) it follows that:

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

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 nodes $v \in V$ 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 node 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 node-selection phase, this concludes the proof of Theorem 2.

Acknowledgments

We would like to thank Thomas Erlebach for useful discussions, and Volker Kaibel for suggesting a simplified proof of Lemma 5.

References

[1] 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.

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

[3] U. Feige. A threshold of ln n for approximating set cover. In *Proceedings, ACM Symposium on Theory of Computing*, pages 314–318, 1996.

[4] 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 Processing*, 2003.

[5] 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.

[6] S. Guha, R. Hassin, S. Khuller, and E. Or. Capacitated vertex covering with applications. In *Proceedings, ACM-SIAM Symposium on Discrete Algorithms*, pages 858–865, 2002.

[7] Halperin. Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs. *SIAM J. Comput.*, 31, 2002.

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

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

[10] F. Khun and R. Wattenhofer. Constant-time distributed dominating set approximation. In *Proceedings, ACM Symposium on Principles of Distributed Computing*, 2003.

Appendix

A A Lower Bound

In this section we provide a proof of Theorem 3, which we restate for completeness.

Theorem 4 *There is an instance of* `capVC` *where each vertex has weight either* 0 *or* 1 *and for which any distributed approximation algorithm requires* $\Omega(n)$ *communication rounds.*

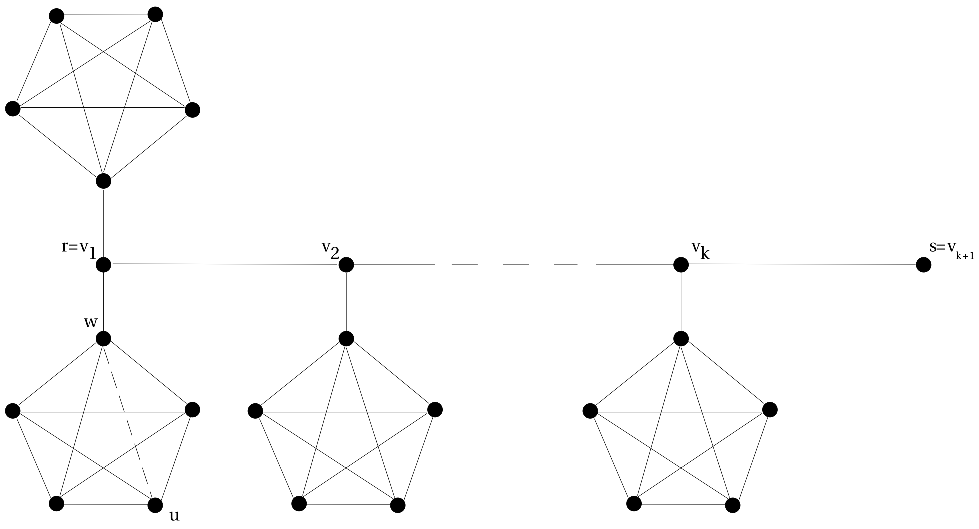

Figure 2: The graphs G_2^0 and G_2^1. The dashed edge (w, u) belongs to graph G_2^1 only.

Proof: Let $B \geq 1$ be a constant. To prove the theorem we introduce, two weighted graphs G_B^0 and G_B^1 with n vertices. In G_B^1 there are two special vertices r and s and between these two nodes there is a simple path $\langle v_1, \ldots, v_{k+1}\rangle$ where $r = v_1$, $s = v_{k+1}$, and $k = \Theta(\frac{n}{B})$.

Then we have $k + 1$ cliques $C_0, \ldots, C_k$ each of them having $(2B + 1)$ vertices. We also add $B - 1$ edges between v_i and arbitrary $B - 1$ vertices of C_i for all $1 \leq i \leq k$. Vertex s is not directly connected to a clique. Moreover, there is an edge between r and an arbitrary vertex w of C_0.

Graph G_B^0 is obtained from G_B^1 by removing one edge (u, w) of clique C_0. All vertices in both graphs have capacities equal to B and all vertices have cost zero except for s which has cost one. Figure 2 shows the two graphs G_2^0 and G_2^1.

Notice that in a clique with $h = 2B + 1$ vertices there exists a feasible assignment. We define it in the following way. Let $v_0, \ldots, v_{2B}$ be the vertices of the clique. We think of these vertices lying on the perimeter of a circle where a vertex with index $i \mod (2B + 1)$ precedes that with index $(i + 1) \mod (2B + 1)$ for all $i \geq 0$.

Now consider an edge (v_i, v_j). Starting from v_i let us count the vertices that, in the circle, are between v_i and v_j. If this number is less than or equal to B we assign (v_i, v_j) to v_i. Otherwise we assign this edge to v_j. Note that this is a feasible assignment. Moreover note that in any feasible assignment for such a clique, every vertex must cover exactly B edges since the number of edges is hB.

These facts imply that, in any feasible solution for the graph G_B^1, r must cover the B edges connecting it to the two cliques C_0 and C_1. Hence the edge (r, v_2) must be assigned to v_2. But v_2 must also cover the $B - 1$ edges connecting it to C_2 and so it has no remaining capacity to cover the edge (v_2, v_3). Hence, generalizing, the edge (v_i, v_{i+1}) must be assigned to v_{i+1} implying that $s = v_{k+1}$ must be included in any feasible solution.

The situation is different on input-graph G_B^0: Remember that G_B^0 is obtained from G_B^1 by removing uw and so w can cover the edge (w, r). Hence r can cover the edge (r, v_2) and in general v_i can cover (v_i, v_{i+1}) for all $i \geq 1$. This implies that the edge (v_k, v_{k+1}) can be assigned to v_k and, since v_{k+1} has cost one any approximate solution must not include s.

Intuitively, s and r need to communicate in order for s to determine whether it is needed or not. Since they are at distance $\Omega(n)$ we need $\Omega(n)$ rounds. We formalize this intuition by defining on the given two graphs a problem called *simple-mail*. The input is a bit in $\{0, 1\}$ provided to vertex r and the output, returned by the vertex s is the same bit provided to r.

Trivially, since the distance between r and s is $\Omega(\frac{n}{B})$ any distributed algorithm for simple-mail requires $\Omega(\frac{n}{B})$ communication rounds.

Suppose there is a distributed algorithm $\mathcal{A}$ for capacitated vertex cover that needs $o(\frac{n}{B})$ rounds. We can use it to solve the *simple-mail* problem. In a setup phase if the bit given to r is 0, r asks w to delete (u, w) and thereby creates graph G_B^0. Otherwise (the bit is 1), w creates graph G_B^1 by adding this edge.

Afterwards, every vertex runs Algorithm $\mathcal{A}$ on the modified graph. If s belongs to the vertex cover solution then s outputs 1 and otherwise it outputs 0. Since the setup phase requires only one round, we have found a distributed algorithm for *simple-mail* which needs $o(\frac{n}{B})$ rounds, which is a contradiction. $\blacksquare$