SIAM J. DISCRETE MATH.
Vol. 24, No. 4, pp. 1441–1469

APPROXIMATION OF PARTIAL CAPACITATED VERTEX COVER*

REUVEN BAR-YEHUDA†, GUY FLYSHER‡, JULIÁN MESTRE§, AND DROR RAWITZ¶

Abstract. We study the *partial capacitated vertex cover problem* (PCVC) in which the input consists of a graph G and a covering requirement L. Each edge e in G is associated with a *demand* (or *load*) $\ell(e)$, and each vertex v is associated with a (soft) *capacity* $c(v)$ and a *weight* $w(v)$. A feasible solution is an assignment of edges to vertices such that the total demand of assigned edges is at least L. The weight of a solution is $\sum_v \alpha(v)w(v)$, where $\alpha(v)$ is the number of copies of v required to cover the demand of the edges that are assigned to v. The goal is to find a solution of minimum weight. We consider three variants of PCVC. In PCVC with *separable demands* the only requirement is that the total demand of edges assigned to v is at most $\alpha(v)c(v)$. In PCVC with *inseparable demands* there is an additional requirement that if an edge is assigned to v, then it must be assigned to one of its copies. The third variant is the unit demands version. We present 3-approximation algorithms for both PCVC with separable demands and PCVC with inseparable demands. We also present a 2-approximation algorithm for PCVC with unit demands. We show that similar results can be obtained for PCVC in hypergraphs and for the *prize collecting* version of capacitated vertex cover. Our algorithms are based on a unified approach for designing and analyzing approximation algorithms for capacitated covering problems. This approach yields simple algorithms whose analyses rely on the local ratio technique and sophisticated charging schemes.

Key words. approximation algorithms, capacitated covering, local ratio technique, partial covering, prize collecting covering

AMS subject classifications. 68W25, 68W40, 05C85

DOI. 10.1137/080728044

1. Introduction.

1.1. The problems. Given a graph $G = (V, E)$, a *vertex cover* is a subset $U \subseteq V$ such that each edge in G has at least one endpoint in U. In the *vertex cover problem*, we are given a graph G and a weight function w on the vertices, and our goal is to find a minimum weight vertex cover. Vertex cover is NP-hard [22], and cannot be approximated within a factor of $10\sqrt{5} - 21 \approx 1.36$, unless P = NP [12]. On the positive side, there are several 2-approximation algorithms for vertex cover (for more details, see [20, 28, 6] and the references therein).

The *capacitated vertex cover problem* (CVC) is an extension of vertex cover in which each vertex $u \in V$ has a *capacity* $c(u) \in \mathbb{N}$ that determines the number of edges it may cover. That is, u may cover up to $c(u)$ incident edges. Multiple copies of u may be used to cover additional edges, provided that the weight of u is counted for each copy (*soft capacities*). A feasible solution is an assignment of every edge to one of its endpoints. Vertex cover is the special case where $c(u) = \deg(u)$ for every vertex u.

*Received by the editors June 23, 2008; accepted for publication (in revised form) August 26, 2010; published electronically November 4, 2010. An extended abstract of this paper was presented at the 15th Annual European Symposium on Algorithms (ESA), 2007.

http://www.siam.org/journals/sidma/24-4/72804.html

†Department of Computer Science, Technion—Israel Institute of Technology, Haifa 32000, Israel (reuven@cs.technion.ac.il). This author's research was supported in part by REMON—Israel 4G Mobile Consortium.

‡Google Inc., Building 30, MATAM, Haifa 31905, Israel (guyfl@google.com). This author's research was done while the author was a graduate student at the Technion.

§Max-Planck-Institute für Informatik, Saarbrücken, Germany (jmestre@mpi-inf.mpg.de).

¶School of Electrical Engineering, Tel Aviv University, Tel Aviv 69978, Israel (rawitz@eng.tau.ac.il).

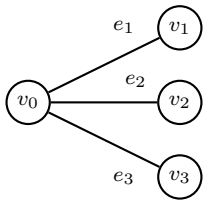


FIG. 1.1. *A* CVC *instance:* $c(v_0) = 3$ *and* $\ell(e_i) = 2$ *for* $i \in \{1, 2, 3\}$. *If the demands are separable, then two copies of* v_0 *are sufficient to cover the three edges, but if the demands are inseparable, then three copies of* v_0 *are needed.*

A *capacitated vertex cover* is formally defined as follows. An *assignment* is a function $A : V \to 2^E$. That is, for every vertex u, $A(u) \subseteq E(u)$, where $E(u)$ denotes the set of edges incident on u. An edge e is said to be *covered by* A (or simply *covered*) if there exists a vertex u such that $e \in A(u)$. Since there is no reason to cover an edge more than once, we henceforth assume, without loss of generality, that A induces a partition of the edges, namely we assume that $A(u) \cap A(v) = \emptyset$, if $u \neq v$. An assignment A is a *cover* if every edge is covered by A, i.e., if $\bigcup_{u \in V} A(u) = E$. The *multiplicity* (or number of copies) of a vertex u with respect to an assignment A is the smallest integer $\alpha(u)$ for which $|A(u)| \leq \alpha(u)c(u)$. The weight of a cover A is $w(A) = \sum_u \alpha(u)w(u)$. Thus, although feasible solutions are defined as assignments of edges to vertices, when considering their weight, they can be viewed as vectors of α values, and thus can be treated within the local ratio framework.

Note that the presence of zero-capacity vertices may render the problem instance infeasible, but detecting this is easy: The instance is infeasible if and only if there is an edge whose endpoints have zero capacity.

In a more general version of CVC we are given a *demand* or *load* function $\ell : E \to \mathbb{N}$. In the *separable* edge demands, case $\alpha(u)$ is the number of copies of u required to cover the total demand of edges in $A(u)$, namely $\alpha(u)$ is the smallest integer for which $\sum_{e \in A(u)} \ell(e) \leq \alpha(u)c(u)$. On the other hand, in the case of *inseparable* edge demands, there is an additional requirement that each edge in $A(u)$ must be assigned in its entirety to one of u's copies, i.e., an edge cannot be assigned to two or more copies of u, each covering part of its demand. Clearly, given an assignment A, this additional requirement may only increase $\alpha(u)$, since each vertex faces its own *bin packing* problem. For example, in Figure 1.1 two copies of v_0 can cover e_1, e_2, and e_3 in the case of separable demands, while they are not enough in the case of inseparable demands. Furthermore, observe that if all edges are covered in the inseparable demands sense, they are covered in the separable sense. Hence, the optimum value for CVC with separable demands is not larger than the optimum value for CVC with inseparable demands. Also note that if the demand of an edge is larger than the capacity of either of its endpoints, then it cannot be covered if the demands are inseparable. Henceforth, in the inseparable demands case, we assume that each edge has at least one endpoint that can cover it.

Another extension of vertex cover is the *partial vertex cover problem* (PVC). In PVC the input consists of a graph G and an integer L, and the objective is to find a minimum weight subset $U \subseteq V$ that covers at least L edges. PVC extends vertex cover, since in vertex cover $L = |E|$. In a more general version of PVC we are given edge *demands*, and the goal is to find a minimum weight subset U that covers edges whose total demand is at least L.

In this paper we mainly focus on the *partial capacitated vertex cover problem* (PCVC) that extends both PVC and CVC. In PCVC we are asked to find a minimum weight capacitated vertex cover that covers edges whose total demand is at least L. We consider three variants of PCVC: PCVC with *separable demands*, PCVC with *inseparable demands*, and PCVC with unit demands.

A fourth extension of vertex cover is *prize collecting vertex cover* (also called *generalized vertex cover*). In the *prize collecting* version of capacitated vertex cover (PC-CVC) we are not obligated to cover all edges, but must pay a *penalty* for uncovered edges. More specifically, both vertices and edges have nonnegative weights; every set of vertices is a feasible solution; and the weight of a feasible solution is the total weight of its vertices plus the total weight of uncovered edges. In PC-CVC the goal is to find an assignment A that minimizes the expression $w(A) + \sum_{e \not\subseteq \cup_u A(u)} w(e)$.

1.2. Motivation. CVC was proposed by Guha et al. [17], who were motivated by a problem from the area of bioinformatics. They described a chip-based technology, called GMID (Glycomolecule ID), that is used to discover the connectivity of building blocks of glycoproteins. Given a set of building blocks (vertices) and their possible connections (edges), the goal is to identify which connections exist between the building blocks in order to discover the variant of the glycoprotein in question. Given a building block B and a set S that consists of k of B's neighbors, GMID can reveal which of the building blocks from S are connected to B. The problem of minimizing the number of GMID operations needed to cover the required information graph is exactly CVC with uniform capacities. Since not all possible combinations of ties between building blocks may appear, it is sometimes sufficient to probe only a fraction of the edges in the information graph. In this case the problem of minimizing the number of GMID operations is PCVC with uniform capacities. For details about the structure of glycoproteins we refer the reader to [24].

CVC can be seen as a *capacitated facility location problem*, where the edges are the clients and the vertices are the capacitated facilities. For instance, CVC in hypergraphs can be used to model a cellular network planning problem, in which we are given a set of potential base stations that are supposed to serve clients that are located in receiving areas. Each such area corresponds to a specific subset of the potential base stations. We are supposed to assign areas (edges) to potential base stations (vertices) and to locate transmitters with limited bandwidth capacity in each potential base station that will serve the demand of receiving areas that are assigned to it. Our goal is to assign areas to base stations so as to minimize transmitter costs. In terms of the above cellular network planning problem, PCVC is the case where we are required to cover only a given percentage of the demands. For details about cellular planning we refer the reader to [1] and the references therein.

1.3. Related work. Capacitated covering problems date back to Wolsey [29] (see also [3, 11]). Wolsey studied the version of set cover in which one may use only one copy of each set. In this case the capacities are referred to as *hard*. Wolsey presented a greedy algorithm for weighted set cover with hard capacities that achieves a logarithmic approximation ratio.

Guha et al. [17] presented a primal-dual 2-approximation algorithm for CVC (a local ratio interpretation was given in [6]) and a 3-approximation algorithm for CVC with separable edge demands. Both algorithms can be extended to hypergraphs in which the maximum size of an edge is Δ. The resulting approximation ratios are Δ and $\Delta + 1$, respectively. Guha et al. [17] studied CVC in trees. They obtained

polynomial time algorithms for the unit demands case and for the unweighted case, and proved that CVC with separable demands in trees is weakly NP-hard. Guha et al. [17] also presented a relatively simple LP-rounding 4-approximation algorithm for CVC with unit demands. We note that this approach does not directly extend to PCVC; indeed, to better motivate our approach we show in Appendix A that a natural randomized LP rounding algorithm is not a constant factor approximation algorithm for PCVC (or even for PVC).

Gandhi et al. [14] presented a 2-approximation algorithm for CVC using an LP-rounding method called *dependent rounding*. Chuzhoy and Naor [11] presented a 3-approximation algorithm for unweighted CVC with hard capacities, which is based on randomized rounding with alterations. They also proved that the weighted version of this problem is as hard to approximate as set cover. Gandhi et al. [13] improved the approximation ratio for unweighted CVC with hard capacities to 2. Recently, Grandoni et al. [16] presented a primal-dual bicriteria distributed algorithm for CVC with hard capacities.

Partial set cover was first studied by Kearns [23], who proved that the performance ratio of the greedy algorithm is at most $2H_m + 3$, where H_m is the mth harmonic number. Slavík [26] showed that it is actually bounded by H_m. Bshouty and Burroughs [10] obtained the first polynomial time 2-approximation algorithm for PVC. Bar-Yehuda [4] studied PVC with demands and presented a local ratio 2-approximation algorithm for this problem. His algorithm extends to a Δ-approximation algorithm in hypergraphs in which the maximum size of an edge is Δ. Gandhi, Khuller, and Srinivasan [15] presented a different algorithm with the same approximation ratio for PVC with unit demands in hypergraphs that is based on a primal-dual analysis. Building on [15], Srinivasan [27] obtained a polynomial time $(2-\Theta(\frac{1}{d}))$-approximation algorithm for PVC, where d is the maximum degree of a vertex in G. Halperin and Srinivasan [19] developed a $(2 - \Theta(\frac{\ln \ln d}{\ln d}))$-approximation algorithm for PVC.

Recently, Mestre [25] studied PCVC with unit demands. He presented a primal-dual 2-approximation algorithm that is based on the 2-approximation algorithm for CVC by Guha et al. [17] and on the 2-approximation algorithm for PVC by Gandhi, Khuller, and Srinivasan [15].

We note that the parameterized complexity of PVC and CVC was investigated by Guo, Niedermeier, and Wernicke [18], who showed that CVC is fixed-parameter tractable, while PVC is $W[1]$-hard. It follows that PCVC is also $W[1]$-hard.

The prize collecting version of vertex cover was studied by Hochbaum [21], who presented a 2-approximation algorithm based on LP-rounding and maximum flow. Bar-Yehuda and Rawitz [9] presented a linear time local ratio Δ-approximation algorithm for prize collecting vertex cover in hypergraphs that extends the approximation algorithm for vertex cover in hypergraphs [7].

1.4. Our results. We present constant factor approximation algorithms to several variants of PCVC. Our algorithms are based on a unified approach for designing and analyzing approximation algorithms for capacitated covering problems. This approach yields simple algorithms whose analyses rely on the local ratio technique and sophisticated charging schemes. Our method is inspired by the local ratio interpretations of the approximation algorithms for CVC from [17] and the local ratio 2-approximation algorithm for PVC from [4].

Local ratio analyses typically show that a solution that belongs to a certain family (e.g., solutions that are minimal with respect to set inclusion) are r-approximate with respect to some weight function (or weight functions). Our approach is powerful

since we consider the family of solutions that are computed by our algorithm. This means that we do not account for solutions that our algorithm cannot compute. This also makes our analyses more complicated than typical local ratio analyses. We believe that our approach can be used to obtain approximation algorithms for other capacitated covering problems.

We present a 3-approximation algorithm for PCVC with separable demands. Note that the analysis of this algorithm is one of the most sophisticated local ratio analyses in the literature. We present a 3-approximation algorithm for PCVC with inseparable demands. As far as we know, this algorithm is the first 3-approximation algorithm for CVC with inseparable demands. We also present a 2-approximation algorithm for PCVC with unit demands. This algorithm is much simpler and more intuitive than Mestre's 2-approximation algorithm [25]. It is important to note that our algorithm is not a local ratio manifestation of Mestre's algorithm. While his algorithm relies on the algorithm for PVC of Gandhi, Khuller, and Srinivasan [15], our algorithm extends the algorithm for PVC from [4], and these two algorithms are different.

We show that our algorithms can be extended to PCVC in hypergraphs, where the maximum size of an edge is Δ. The approximation ratios for separable demands, inseparable demands, and unit demands are $\Delta+1$, $\Delta+1$, and Δ, respectively. Finally, we show that the same approximation ratios can be obtained for PC-CVC in hypergraphs.

1.5. Overview. The remainder of this paper is organized as follows. Section 2 contains some definitions and a short explanation about the local ratio technique. We first consider PCVC with unit demands in section 3, where we present a 2-approximation algorithm. In section 4 we present a 3-approximation algorithm for a special case of PCVC with separable demands in which any edge can be covered by a single copy of either of its endpoints. Then, in sections 5 and 6, we show how to use this algorithm to design 3-approximation algorithms for PCVC with inseparable demands, and for PCVC with separable demands. Our results are extended to hypergraphs in section 7, and finally we consider PC-CVC in section 8.

2. Preliminaries.

2.1. Notation and terminology. Given an undirected graph $G = (V, E)$, let $E(u)$ be the set of edges incident on u, and let $N(u)$ be the set of u's neighbors. Given an edge set $F \subseteq E$ and a demand (or load) function ℓ on the edges of G, we denote the total demand of the edges in F by $\ell(F)$. That is, $\ell(F) = \sum_{e \in F} \ell(e)$. We define $\deg(u) = \ell(E(u))$. Hence, in the unit demands case $\deg(u)$ is the degree of u, i.e., $\deg(u) = |E(u)| = |N(u)|$.

We define $\tilde{c}(u) = \min\{c(u), \deg(u)\}$. This definition may seem odd, since we may assume that $c(u) \leq \deg(u)$ for every vertex u in the input graph. However, our algorithms repeatedly remove vertices from the given graph, and therefore we may encounter a graph in which there exists a vertex where $\deg(u) < c(u)$. We also define $b(u) = \min\{c(u), \deg(u), L\} = \min\{\tilde{c}(u), L\}$. $b(u)$ can be seen as the covering potential of u in the current graph. A single copy of u can cover $c(u)$ of the demand if $\deg(u) \geq c(u)$, but if $\deg(u) < c(u)$, we cannot cover more than a total of $\deg(u)$ of the demand. Moreover, if L is smaller than $\tilde{c}(u)$, we have nothing to gain from covering more than L.

Let A be an assignment. The *support* of A is the set of vertices $V(A) = \{u : A(u) \neq \emptyset\}$. We denote by $E(A)$ the set of edges covered by A. That is, $E(A) = \cup_u A(u)$. We define $|A| = |E(A)|$ and $\ell(A) = \ell(E(A))$. Note that in the unit demand case $\ell(A) = |A|$.

2.2. Small, medium, and large edges. Given a PCVC instance, we refer to an edge $e = (u, v)$ as *large* if $\ell(e) > c(u)$ and $\ell(e) > c(v)$. If $\ell(e) \leq c(u)$ and $\ell(e) \leq c(v)$, the edge e is called *small*. Otherwise, e is called *medium*.

Given an edge $e = (u, v)$ we sometimes add a new vertex u_e to the graph, where $w(u_e) = \infty$ and $c(u_e) = \ell(e)$, and connect e to u_e instead of to u. In this case we say that e was *detached* from u, and we refer to this operation as an *edge detachment*.

Consider the case of PCVC with inseparable demands. Since a large edge cannot be assigned to a single copy of one of its endpoints, it follows that we may ignore large edges in the case of PCVC with inseparable demands. Notice that the instance may contain a medium edge $e = (u, v)$ such that $c(u) < \ell(e) \leq c(v)$. If there exist such an edge e, then it cannot be assigned to u, and therefore we may detach e from u. Since $e \notin A(u_e)$ in any solution A of finite weight, we may assume, without loss of generality, that all edges are small. In what follows, when we discuss PCVC with inseparable demand, we assume that all edges are small.

A similar problem may happen in the separable demands case when there exists a vertex u such that $c(u) = 0$. In what follows, we assume without loss of generality that there are no vertices with zero capacity. If there exists such a vertex u, we simply define $c(u) = 1$ and $w(u) = \infty$.

2.3. Local ratio. The *local ratio technique* [8, 2, 5] is based on the local ratio theorem, which applies to optimization problems of the following type. The input is a nonnegative weight vector $w \in \mathbb{R}^n$ and a set of feasibility constraints $\mathcal{F}$. The problem is to find a vector $x \in \mathbb{R}^n$ that minimizes (or maximizes) the inner product $w \cdot x$ subject to the set of constraints $\mathcal{F}$.

THEOREM 2.1 (local ratio [5]). *Let $\mathcal{F}$ be a set of constraints and let w, w_1, and w_2 be weight vectors such that $w = w_1 + w_2$. Then, if x is r-approximate with respect to $(\mathcal{F}, w_1)$ and with respect to $(\mathcal{F}, w_2)$, for some r, then x is also an r-approximate solution with respect to $(\mathcal{F}, w)$.*

3. Partial capacitated vertex cover with unit demands. In this section we present a local ratio 2-approximation algorithm for PCVC with unit demands. Our algorithm is inspired by the local ratio interpretation of the approximation algorithms for CVC from [17] and the local ratio approximation algorithm for PVC from [4].

3.1. The algorithm. The structure of our algorithm follows the typical structure of local ratio approximation algorithms for covering problems [5, 6]. However, the analysis of the algorithm does not follow the standard covering theme. In local ratio analyses it is typically shown that a certain family of solutions (e.g., solutions that are minimal with respect to set inclusion) are r-approximate with respect to some weight function (or weight functions). Our approach is different, since we consider the family of solutions that are computed by our algorithm. It follows that we do not account for solutions that our algorithm cannot compute. This also makes our analyses more complicated than typical local ratio analyses.

Our algorithm is recursive, and during its push phase it repeatedly removes vertices with their incident edges from the current graph. In its pop phase, when the algorithm puts back a vertex, it assigns edges to vertices. Assignments are never changed. During this process, the number of edges assigned to a vertex v may be less than $\tilde{c}(v)$. Roughly speaking, this means that we already paid for one copy of v, but did not fully utilize this copy. Our algorithm prefers to assign edges to such vertices.

This brings us to the following definition.

DEFINITION 3.1. *Given an assignment A, a vertex v is called* vulnerable *if* $0 < |A(v)| < \tilde{c}(v)$ *and there exists an uncovered edge e that is incident on it.*

Algorithm 1 is our local ratio 2-approximation algorithm for PCVC with unit demands. It consists of a four-way if condition.

1. If $L = 0$, return the empty assignment.
2. If there exists a vertex with zero degree, remove this vertex and solve the problem recursively on the resulting instance.
3. If there exists a zero weight vertex u, remove u and $E(u)$ and solve the problem recursively. Then, as long as there are less than L covered edges, uncovered edges are assigned to vertices while giving precedence to certain vulnerable vertices. That is, as long as there exists a vulnerable vertex of a certain kind an uncovered edge is assigned to it. If there are no more such vulnerable vertices, then edges are assigned to u.
4. Otherwise, if there are no zero weight vertices in G, then construct a weight function w_1 which is proportional to b and subtracts w_1 from w. (Recall that $b(u) = \min\{\deg(u), c(u), L\}$.) Then, recursively compute a solution with respect to the new weights and return this solution. Observe that w_1 is constructed in a way that ensures that there exists a zero weight vertex with respect to the weight function $w_2 = w - w_1$.

Vulnerable vertices play a crucial role in both the definition and the analysis of the algorithm. As we shall see, the way we pay for the necessary copies of each vertex is by charging the edges assigned to that vertex. Vulnerable vertices cause problems because they do not have enough edges assigned to pay for themselves. Therefore, in lines 6–8 we give priority to vulnerable vertices before assigning edges to the vertex that is currently being processed.

Algorithm 1: **UNIT**(V, E, w, L)

1: **if** $L = 0$ **then return** $A(v) = \emptyset$ for all $v \in V$
2: **if** there exists $u \in V$ such that $\deg(u) = 0$ **then return UNIT**$(V \setminus \{u\}, E, w, L)$
3: **if** there exists $u \in V$ such that $w(u) = 0$ **then**
4: $\quad A \leftarrow$ **UNIT**$(V \setminus \{u\}, E \setminus E(u), w, \max\{L - \deg(u), 0\})$
5: $\quad$ **while** $|A| < L$ **do**
6: $\quad\quad$ **if** $V(A) = \{v_0\}$ and v_0 is vulnerable **then**
$\quad\quad\quad A(v_0) \leftarrow A(v_0) \cup \{e\}$, where $e \in E(v_0)$ is uncovered
7: $\quad\quad$ **else**
$\quad\quad$ **if** there exists a vulnerable vertex $v \in N(u)$ **then**
$\quad\quad\quad A(v) \leftarrow A(v) \cup \{(u, v)\}$
8: $\quad\quad$ **else**
$\quad\quad\quad A(u) \leftarrow A(u) \cup \{e\}$, where $e \in E(u)$ is uncovered
9: $\quad$ **return** A
10: Let $\varepsilon = \min_{u \in V}\{w(u)/b(u)\}$
11: Define the weight functions $w_1(v) = \varepsilon \cdot b(v)$, for every $v \in V$, and $w_2 = w - w_1$
12: **return UNIT**(V, E, w_2, L)

Observe that in each recursive call of Algorithm 1 there are three options: (i) a zero degree vertex is removed, (ii) a zero weight vertex is removed, or (iii) the weight of a vertex decreases to zero. It follows that there are $O(|V|)$ recursive calls. Hence, the running time of the algorithm is polynomial.

Consider the recursive call made in line 4. In order to distinguish between the assignment obtained by this recursive call and the assignment returned in line 9, we

denote the former by A' and the latter by A. Assignment A' is for graph $G' = (V', E')$, and we denote the corresponding parameters α', $\deg'$, $\tilde{c}'$, b', and L'. Similarly, we use α, deg, $\tilde{c}$, b, and L, for the parameters of A and $G = (V, E)$.

LEMMA 3.2. *Algorithm* 1 *computes a partial capacitated vertex cover.*

Proof. First, notice that we assign only uncovered edges. We prove by induction on the recursion that $|A| = L$. In the base case $|A| = 0 = L$ and we are done. For the inductive step, if the recursive call was made in line 2 or in line 12, then $|A| = L$ by the inductive hypothesis. If the recursive call was made in line 4, then $|A'| = \max\{L - \deg(u), 0\}$ by the inductive hypothesis. Since the edges incident on u are not covered by A', the while loop is able to increase the number of covered edges by $\deg(u)$, if necessary. Hence, $|A| = L$ after the while loop. ∎

To analyze the algorithm, we use a charging scheme by which at each point we have at our disposal $2L$ coins that we may distribute between the vertices. Let $\$(v)$ be the number of coins that are given to v. We aim to show that there is a way to distribute the coins so that $\$(v) \geq \alpha(v)b(v)$. We distribute the coins as follows. First, as long as there is only one vertex in $V(A)$, then this vertex gets all the coins. Furthermore, whenever an edge is assigned to a vulnerable vertex v, then one coin is given to v and the second to u, otherwise both of them are given to u. Since $|A| = L$, by Lemma 3.2, it follows that exactly $2L$ coins were distributed.

LEMMA 3.3. *Let A be an assignment that was computed by Algorithm* 1 *for a graph G and a covering demand L. Then, there exists a charging scheme $\$$ (with respect to G and A) such that* (i) $\$(u) \geq \alpha(u)b(u)$, *for every vertex u, and* (ii) $\sum_v \$(v) \leq 2L$.

We will prove Lemma 3.3 in what follows. We first show how Lemma 3.3 is used to provide an approximation guarantee for Algorithm 1.

THEOREM 3.4. *Algorithm* 1 *computes a* 2*-approximate partial capacitated vertex cover.*

Proof. The proof is by induction on the recursion. In the base case the algorithm returns an empty assignment, which is optimal. For the inductive step there are three cases.

If the recursive invocation is made in line 2, then by the inductive hypothesis the assignment is 2-approximate with respect to $(V \setminus \{u\}, E)$. It follows that it is 2-approximate with respect to (V, E) and we are done.

If the recursive invocation is made in line 4, then by the inductive hypothesis the assignment A' is 2-approximate with respect to G' and $\max\{L - \deg(u), 0\}$. Clearly, the optimum value for G can only be greater than or equal to the optimum value for G', and because $w(u) = 0$ and $\alpha(v) = \alpha'(v)$ for all $v \in V \setminus \{u\}$ by Observation 3, it follows that $w(A) = w(A')$. Thus A is 2-approximate with respect to G.

If the recursive call is made in line 12, then by the inductive hypothesis the assignment is 2-approximate with respect to w_2. By Lemma 3.3 $w_1(A) \leq \varepsilon \cdot 2L$. Furthermore, we show that $w_1(\bar{A}) \geq \varepsilon \cdot L$ for every feasible assignment $\bar{A}$. If there exists $v \in V(\bar{A})$ such that $b(v) = L$, then $w_1(\bar{A}) \geq L \cdot \varepsilon$. Otherwise, $b(v) = \tilde{c}(v) < L$ for every $v \in V(\bar{A})$. It follows that

$$w_1(\bar{A}) = \varepsilon \cdot \sum_v \bar{\alpha}(v)\tilde{c}(v) \geq \varepsilon \cdot \sum_v \left|\bar{A}(v)\right| \geq \varepsilon \cdot L.$$

Thus, A is 2-approximate with respect to w_1 too, and by the local ratio theorem it is 2-approximate with respect to w as well. ∎

3.2. Proof of Lemma 3.3. The following observations will allow us to design the necessary charging scheme to prove Lemma 3.3.

First, we observe that if the solution uses a vertex v whose covering potential is L, then this is the only vertex used by the solution.

OBSERVATION 1. *If there exists $v \in V(A)$ such that $b(v) = L$, then $V(A) = \{v\}$.*

Proof. The proof is by induction on the recursion. In the base case A is the empty assignment, and we are done. For the inductive step there are two cases. If $|V(A)| = 1$, then the claim is satisfied. Otherwise, $|V(A)| > 1$. By the inductive hypothesis it follows that either $b'(v) < L'$ for every $v \in V(A')$ or $V(A') = \{v\}$ and $b'(v) = L'$. First, in both cases, $\deg(u) < L$, since otherwise $A'(v) = \emptyset$, for every v, a contradiction. Hence, $b(u) < L$. If $b'(v) < L'$ for every $v \in V(A')$, then $b(v) < L$ for every $v \in V(A')$. On the other hand, assume that $V(A') = \{v\}$ and $b'(v) = L'$. If $b(v) = L$, then v is vulnerable throughout the while loop (lines 5–8), which means that $V(A) = \{v\}$, in contradiction to $|V(A)| > 1$. ∎

OBSERVATION 2. *Suppose a recursive call is made in line 4. Let $v \in N(u)$. Then*

1. *If v is vulnerable with respect to G and A', then $\alpha(v) = \alpha'(v) = 1$ and $\tilde{c}(v) \leq \tilde{c}'(v) + 1$. Thus $\alpha(v)\tilde{c}(v) \leq \alpha'(v)\tilde{c}'(v) + 1$.*
2. *If v is not vulnerable with respect to G and A', then $\alpha(v)\tilde{c}(v) = \alpha'(v)\tilde{c}'(v)$.*

Proof. For the first property, observe that $|A'(v)| < \tilde{c}(v) \leq c(v)$, since v is vulnerable with respect to G and A'. Hence $\alpha(v) = \alpha'(v) = 1$. Also, note that $\deg(v) = \deg'(v) + 1$ since v has u as a neighbor in G but not in G'. It follows by the definition of $\tilde{c}$ that $\tilde{c}(v) \leq \tilde{c}'(v) + 1$.

For the second property, note that $A(v) = A'(v)$ and therefore $\alpha(v) = \alpha'(v)$. If $A(v) = \emptyset$, then $\alpha(v) = \alpha'(v) = 0$ and the property holds trivially. Otherwise, because v is not vulnerable with respect to G and A', we have $|A'(v)| \geq \tilde{c}(v)$ and thus

$$\deg(v) > \deg'(v) \geq |A'(v)| \geq \tilde{c}(v) = \min\{\deg(v), c(v)\}.$$

Thus, it must be the case that $\deg(v) > \deg'(v) \geq c(v)$ and thus $\tilde{c}(v) = \tilde{c}'(v) = c(v)$. Together with the fact that $\alpha(v) = \alpha'(v)$, the second property follows. ∎

OBSERVATION 3. *Suppose a recursive call is made in line 4. Then, $\alpha(v) = \alpha'(v)$ for every $v \in V \setminus \{u\}$.*

Proof. Clearly, $\alpha(v) = \alpha'(v)$ for every v that is not vulnerable with respect to G and A'. Let v be a vulnerable vertex. If $v \in N(u)$, then this follows from Observation 2. Otherwise, it follows from the fact that edges are assigned to v only if it is vulnerable. ∎

OBSERVATION 4. $\alpha(u)b(u) \leq |A(u)| + b(u)$.

Proof. $|A(u)| + b(u) = \left(\frac{|A(u)|}{b(u)} + 1\right) \cdot b(u) \geq \left(\frac{|A(u)|}{c(u)} + 1\right) \cdot b(u) > \alpha(u)b(u)$. ∎

We are now ready to prove Lemma 3.3.

Proof of Lemma 3.3. The proof is by induction on the recursion. In the base case $A(v) = \emptyset$, for every $v \in V$. Hence, every vertex v gets $\alpha(v)b(v) = 0$ coins, and $\sum_v \alpha(v)b(v) = 0 = 2L$. For the inductive step, there are three possible types of recursive calls corresponding to line 2, line 4, and line 12 of Algorithm 1. The calls in line 2 and line 12 do not change the assignment that is returned by the recursive call. Thus, the claim follows from the inductive hypothesis. The only correction is needed in the case of line 2, where we need to extend the corresponding charging scheme by assigning $\$(u) = 0$. For the rest of the proof we concentrate on recursive calls that are made in line 4.

If the recursive call is made in line 4, then by the inductive hypothesis there exists a charging scheme that allots at least $\alpha'(v)b'(v)$ coins to every vertex in $G' = (V', E')$ and uses $2L'$ coins. Observe that the transition from G' to G consists of adding a single vertex u and the edges incident on it. Furthermore, there are three

possible transformations from A' to A. In the first case, $|A'| = 0$ and hence L edges are assigned to u. In the second case only u and a vertex v_0 such that $V(A') = \{v_0\}$ participate in the change. v_0 is not necessarily vulnerable. The third possible transformation involves u and its neighbors. We extend the charging scheme of G' and A' by distributing the remaining $2(L - L')$ coins in the three cases.

In the first case, u receives $2|A(u)| = 2L$ coins. It follows that $\$(u) = |A(u)| + L$. Since $|A(u)| + L \geq |A(u)| + b(u)$, u is satisfied due to Observation 4.

In the second case, $\alpha(v) = \alpha'(v) = 0$ for every $v \neq u, v_0$, and therefore any vertex $v \neq u, v_0$ is satisfied. We distribute $2(L - L') = 2\deg(u)$ coins as follows. If $A(u) = \emptyset$, then the coins are given to v_0. This means that v_0 possesses $2|A(v_0)|$ coins, and using the same argument as in the first case we are done. We now assume that $A(u) \neq \emptyset$. In this case, for each edge that was assigned to v_0, both u and v_0 receive a single coin, and for each edge that was assigned to u, we give u two coins and none to v_0. That is, u receives $\deg(u) + |A(u)|$ coins, while v_0 gets the rest of the coins. By Observation 4, u receives at least $\alpha(u)b(u)$ coins. As for v_0, it now has $2|A'(v_0)| = 2L'$ coins from the charging scheme of G' and A', and also $|A(v_0)| - |A'(v_0)|$ coins from this recursive call. If v_0 was not vulnerable in the beginning of the while loop, then $A(v_0) = A'(v_0)$ and $b(v_0) \leq \tilde{c}(v_0) \leq A(v_0)$. Hence, $\$(v_0) = 2|A(v_0)| \geq |A(v_0)| + b(v_0)$, and therefore by Observation 4 v_0 has enough coins. Next, we assume that v_0 was vulnerable in the beginning of the while loop. Since it is not vulnerable at the end of the while loop and $A(u) \neq \emptyset$, it follows that $|A(v_0)| = \tilde{c}(v_0)$. Hence, $\alpha(v_0) = 1$ and $\$(v_0) \geq |A(v_0)| = \alpha(v_0)b(v_0)$.

In the third case, we need only take care of u and its neighbors that participate in the cover. We note that due to Observation 1 we know that $b'(v) < L'$, for every $v \in V(A')$. Hence, $b'(v) = \tilde{c}'(v)$, for every $v \in V(A')$. Hence, the inductive hypothesis implies that $\$(v) \geq \alpha'(v)b'(v) = \alpha'(v)\tilde{c}'(v)$. We extend the charging scheme of G' and A' in the following way. For each $v \in N(u)$, if (u, v) is assigned to v, both u and v receive a coin, otherwise we give u both coins. Consider $v \in N(u)$ such that $A(v) \neq \emptyset$. If v is vulnerable, then the edge (u, v) is assigned to v, and therefore it receives a coin, which by Observation 2 is enough to satisfy v. If v is not vulnerable, then due to Observation 2 we are done. Finally, as seen in Lemma 3.2, $|A'| = L'$, and therefore exactly $\deg(u)$ edges are assigned by the while loop. Therefore, all the neighbors of u will be satisfied after the while loop. As for u itself, it is given $|A(u)| + \deg(u)$ coins and it is satisfied due to Observation 4. ∎

4. Partial capacitated covering of small edges. In this section we present a 3-approximation algorithm for the special case of PCVC with separable demands in which all edges are small.

4.1. The algorithm. Algorithm 2 is a recursive local ratio algorithm that computes 3-approximation solutions for PCVC with separable demands and small edges. In the description of the algorithm we use a function called **Next-Uncovered** that, given a vertex u, returns an uncovered edge e incident on u with maximum demand. If all edges in $E(u)$ are covered, it returns NIL).

The algorithm consists of a four-way if condition similar to the one that is found in Algorithm 1. Specifically, Algorithm 2 differs from Algorithm 1 only in the third entry of the if condition. If there exists a zero weight vertex u, remove u and $E(u)$ and solve the problem recursively. Now, there are two options: either the solution returned is the empty assignment or it is not. In the first case u collects uncovered edges in a nonincreasing order of demands until the total demand of assigned edges is at least L. Clearly, this is essential to ensure feasibility. Then, if the total demand of assigned

edges is less than $c(u)$, u continues to collect as many edges as possible as long as the total demand is not greater than $c(u)$. Intuitively, if a single copy of u is used, then it would be wise to assign as many edges as possible to this copy. Specifically, this process ensures that if $\ell(E(u)) > c(u)$, then this single copy of u is at least half used. In the second option the solution returned is not the empty assignment. In this case, if the total demand of covered edges is less than L, all edges in $E(u)$ are assigned to u. Otherwise, no edges are assigned to u.

Algorithm 2: **PCVC**(V, E, w, L)

1: **if** $L = 0$ **then return** $A(v) \leftarrow \emptyset$ for all $v \in V$
2: **if** there exists $u \in V$ such that $\deg(u) = 0$ **then return PCVC**$(V \setminus \{u\}, E, w, L)$
3: **if** there exists $u \in V$ such that $w(u) = 0$ **then**
4: $\quad A \leftarrow$ **PCVC**$(V \setminus \{u\}, E \setminus E(u), w, \max\{L - \deg(u), 0\})$
5: $\quad$ **if** $V(A) = \emptyset$
6: $\quad\quad$ **while** $\ell(A(u)) < L$ **do**
$\quad\quad\quad A(u) \leftarrow A(u) \cup \{$**Next-Uncovered**$(u)\}$
7: $\quad\quad$ **while** (**Next-Uncovered**$(u) \neq$ NIL) and
$\quad\quad\quad\quad (\ell(A(u)) + \ell($**Next-Uncovered**$(u)) \leq c(u))$ **do**
$\quad\quad\quad A(u) \leftarrow A(u) \cup \{$**Next-Uncovered**$(u)\}$
8: $\quad$ **else**
9: $\quad\quad$ **if** $\ell(A) < L$ **then** $A(u) \leftarrow E(u)$
10: $\quad$ **return** A
11: Let $\varepsilon = \min_{u \in V} \{w(u)/b(u)\}$
12: Define the weight functions $w_1(v) = \varepsilon \cdot b(v)$, for every $v \in V$, and $w_2 = w - w_1$
13: **return PCVC**(V, E, w_2, L)

As in Algorithm 1, observe that there are $O(|V|)$ recursive calls. Hence, the running time of the algorithm is polynomial.

In order to distinguish between the assignment obtained by a recursive call made in line 4 and an assignment that is returned in line 10, we denote the former by A' and the latter by A. Assignment A' is for graph $G' = (V', E')$, and we denote the corresponding parameters α', $\deg'$, $\tilde{c}'$, b', and L'. Similarly, we use α, deg, $\tilde{c}$, b, and L, for the parameters of A and $G = (V, E)$.

LEMMA 4.1. *Algorithm* 2 *computes a partial capacitated vertex cover.*

Proof. First, notice that we assign only uncovered edges. We prove that $\ell(A) \geq L$ by induction on the recursion. In the base case $|A| = 0 = L$ and we are done. For the inductive step, if the recursive call was made in line 2 or in line 13, then $\ell(A) \geq L$ by the inductive hypothesis. If the recursive call was made in line 4, then $\ell(A') \geq \max\{L - \deg(u), 0\}$ by the inductive hypothesis. If $V(A') = \emptyset$, then $\deg(u) \geq L$, and therefore $\ell(A(u)) \geq L$. Otherwise, if $V(A') \neq \emptyset$, then there are two options. If $\ell(A') \geq L$, then $A = A'$ and we are done. If $\ell(A') < L$, the edges in $E(u)$ are assigned to u, and since their combined demand is $\deg(u)$ it follows that

$$\ell(A) = \ell(A') + \deg(u) \geq L' + \deg(u) = L,$$

and the lemma follows. ∎

We use the following observations in our analysis.

OBSERVATION 5. *If* $\alpha(v) > 1$, *then* $\deg(v) > c(v)$.

Proof. In the separable demands case, $\alpha(v) = \lceil \ell(A(v))/c(v) \rceil$ for every vertex v. Hence, $\alpha(v) > 1$ implies $\deg(v) \geq \ell(A(v)) > c(v)$. ∎

OBSERVATION 6. *Suppose a recursive call is made in line* 5. *Then,* $\alpha(v) = \alpha'(v)$ *for every* $v \in V \setminus \{u\}$.

Proof. This follows from the fact that only $A(u)$ may change in lines 6–10. ∎

OBSERVATION 7. $\alpha(v)\tilde{c}(v) \leq 2\deg(v)$. *Moreover, if* $\alpha(v) > 1$, *then* $\alpha(v)\tilde{c}(v) \leq 2\ell(A(v))$.

Proof. Since $\tilde{c}(v) \leq \deg(v)$, the first claim is trivial if $\alpha(v) \leq 2$. If $\alpha(v) > 1$, then $\deg(v) > c(v)$, and therefore

$$\alpha(v)\tilde{c}(v) = \alpha(v)c(v) < \ell(A(v)) + c(v) \leq 2\ell(A(v)) \leq 2\deg(v),$$

as required. ∎

To analyze the algorithm, we use a charging scheme by which at each point we have at our disposal $3L$ coins that we distribute between the vertices. Let $\$(v)$ be the number of coins that are given to v. A charging scheme $\$ is called *valid* if $\sum_v \$(v) \leq 3L$. We aim to show that if an assignment A was computed by Algorithm 2, then there is a way to distribute $3L$ coins so that $\$(v) \geq \alpha(v)b(v)$.

LEMMA 4.2. *Let* A *be an assignment that was computed by Algorithm* 2 *for a graph* G *and a covering demand* L. *Then, there exists a valid charging scheme* $\$ *(with respect to* G *and* A*) such that* $\$(u) \geq \alpha(u)b(u)$, *for every vertex* u.

We will prove Lemma 4.2 in the next section. Meanwhile we use it to provide an approximation guarantee for Algorithm 2.

THEOREM 4.3. *Algorithm* 2 *computes* 3*-approximate solutions for* PCVC *with separable demands and small edges.*

Proof. The proof is by induction on the recursion. In the base case the algorithm returns an empty assignment which is optimal. For the inductive step there are three cases. First, if the recursive call is made in line 2, then by the inductive hypothesis the assignment A' is 3-approximate with respect to $(V \setminus \{u\}, E)$. A' is clearly 3-approximate with respect to (V, E) since $\deg(u) = 0$. Second, if the recursive invocation is made in line 4, then by the inductive hypothesis the assignment A' is 3-approximate with respect to $(V \setminus \{v\}, E \setminus E(u))$, w, and $\max\{L - \deg(u), 0\}$. Since $w(u) = 0$, the optimum with respect to (V, E), w, and L is equal to the optimum with respect to $(V \setminus \{v\}, E \setminus E(u))$, w, and $\max\{L - \deg(u), 0\}$. Moreover, since $\alpha(v) = \alpha'(v)$ for every $v \in V \setminus \{u\}$ due to Observation 6, it follows that $w(A) = w(A')$. Thus, A is 3-approximate with respect to (V, E), w, and L. Third, if the recursive call is made in line 13, then by the inductive hypothesis the assignment A' is 3-approximate with respect to (V, E), w_2, and L. By Lemma 4.2 $w_1(A) \leq \varepsilon \cdot 3L$. We show that $w_1(\bar{A}) \geq \varepsilon \cdot L$ for every feasible assignment $\bar{A}$. First, if there exists $v \in V(\bar{A})$ such that $b(v) = L$, then $w_1(\bar{A}) \geq L \cdot \varepsilon$. Otherwise, $b(v) = \tilde{c}(v) < L$ for every $v \in V(\bar{A})$. It follows that

$$w_1(\bar{A}) = \varepsilon \cdot \sum_v \bar{\alpha}(v)\tilde{c}(v) \geq \varepsilon \cdot \sum_v \ell(\bar{A}(v)) \geq \varepsilon \cdot L.$$

Hence, A is 3-approximate with respect to w_1 too, and by the local ratio theorem it is 3-approximation with respect to w as well. ∎

4.2. Proof of Lemma 4.2. In this section we provide a proof for Lemma 4.2. We aim to show that if an assignment A was computed by Algorithm 2, then there is a way to distribute $3L$ coins so that $\$(v) \geq \alpha(v)b(v)$. The proof of the next lemma contains a recursive definition of our charging scheme. Furthermore, Lemma 4.2 is implied by Lemma 4.4 and the definition of b.

We denote by $v_0, v_1, \ldots$ the vertices of $V(A)$ in the order they join the set.

LEMMA 4.4. *Let A be an assignment that was computed by Algorithm 2 for a graph $G = (V, E)$ and a covering demand L. Then, one of the following conditions must hold:*

1. *$V(A) = \emptyset$. Also, the charging scheme $\$(v) = 0$ for every $v \in V$ is valid.*
2. *$V(A) = \{v_0\}$, $\alpha(v_0) = 1$, and $\ell(A(v_0)) \geq \frac{c(v_0)}{2}$. Also, the charging scheme $\$(v_0) = 3L$ and $\$(v) = 0$ for every $v \neq v_0$ is valid.*
3. *$V(A) = \{v_0\}$ and $\alpha(v_0) = 2$. Also, the charging scheme $\$(v_0) = 3L$ and $\$(v) = 0$ for every $v \neq v_0$ is valid.*
4. *$V(A) = \{v_0, v_1\}$, $\alpha(v_0) = 1$, $\alpha(v_1) \geq 2$, and $\ell(A(v_0)) \geq \frac{c(v_0)}{2}$. Also, there exists a valid charging scheme $\$$ such that $\$(v_0) \geq L$, $\$(v_0) \geq 2\ell(A(v_0)) - (\ell(A) - L)$, $\$(v_1) \geq \alpha(v_1)c(v_1)$, and $\$(v) = 0$ for every $v \neq v_0, v_1$.*
5. *$\alpha(v_0) = 1$, $\ell(A(v_0)) < \frac{c(v_0)}{2}$, $\alpha(v) \geq 2$ for every $v \in V(A) \setminus \{v_0\}$, and $L > \deg(v_0) - \ell(A(v_0))$. Also, there exists a valid charging scheme $\$$ such that $\$(v_0) \geq 2L + \ell(A(v_0)) - \ell(A)$, $\$(v) \geq \alpha(v)c(v)$ for every $v \in A(v) \setminus \{v_0\}$, and $\$(v) = 0$ for every $v \notin V(A)$.*
6. *$V(A) \neq \emptyset$. Also, there exists a valid charging scheme $\$$ such that $\$(v) \geq \alpha(v)\tilde{c}(v)$ for every $v \in V$.*

Proof. We prove the lemma by induction on the length of the creation series of the solution, that is, on the number of recursive calls made by the algorithm. (Recall that there are at most $O(|V|)$ recursive calls.)

Each of the six conditions listed in the statement of the lemma characterizes the solution returned by a call to Algorithm 2 and specifies a valid charging scheme for that particular solution. The plan is to show that the algorithm is always able to make a valid transition from one condition to another. More specifically, each call to Algorithm 2, other than the base case, makes a recursive call; we assume that one of the conditions holds for the solution returned by the recursive call and then prove that the augmented solution returned by the current call satisfies one of the conditions. A succinct statement of the six conditions and a transition diagram for the conditions appear in Figure 4.1.

At the base of the induction, the computed solution is the empty assignment, and therefore condition 1 holds. For the inductive step, there are three possible recursive calls corresponding to line 2, line 4, and line 13 of Algorithm 2. If the algorithm uses a recursive call either in line 2 or in line 13, then it simply returns the assignment that was computed by the recursive call, namely $A = A'$. Furthermore, $b(v) = b'(v)$ for any vertex $v \in V'$. Thus, if the assignment satisfies one of the conditions, it continues to satisfy them. The only correction is needed in the case of line 2, where we need to extend the corresponding charging scheme by assigning $\$(u) = 0$.

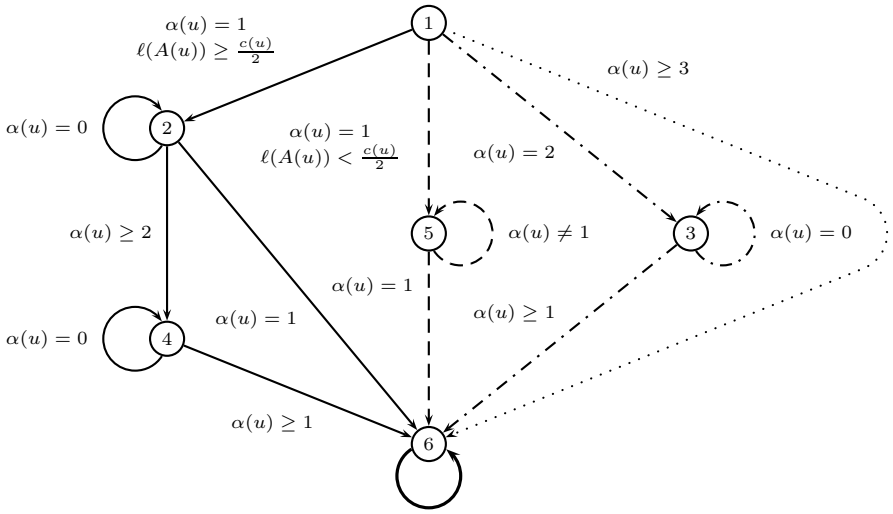
For the rest of the proof we concentrate on recursive calls that are made in line 4. We consider a solution A' that was computed by the recursive call, and denote by $\$'$ the charging scheme that corresponds to A'. For the rest of the proof we show how to construct a charging scheme $\$$ for A that is based on $\$'$.

Consider a recursive call in which A' satisfies condition 1. In this case, $V(A) = \{v_0\}$. How the charging scheme is constructed will depend on how many copies of this first vertex v_0 the algorithm decides to use. There are four cases to consider corresponding to the four transitions[1] out of condition 1 in Figure 4.1.

[1] Earlier proofs of this lemma contained additional cases that were later merged with other cases, thus reducing the complexity of the proof. However, the authors believe that all present cases are necessary in the sense that the changing scheme has to be done differently along different transitions depicted in Figure 4.1.

Condition	Solution	Valid charging scheme
1	$V(A) = \emptyset$	
2	$V(A) = \{v_0\},\ \alpha(v_0) = 1,$ $\ell(A(v_0)) \geq \frac{c(v_0)}{2}$	$\$(v_0) = 3L$
3	$V(A) = \{v_0\},\ \alpha(v_0) = 2$	$\$(v_0) = 3L$
4	$V(A) = \{v_0, v_1\},\ \alpha(v_0) = 1,$ $\alpha(v_1) \geq 2,\ \ell(A(v_0)) \geq \frac{c(v_0)}{2}$	$\$(v_0) \geq \max\{L,\ 2\ell(A(v_0))-(\ell(A)-L)\},$ $\$(v_1) \geq \alpha(v_1)c(v_1)$
5	$\alpha(v_0) = 1,\ \alpha(v_i) \geq 2$ for $i \geq 1$ $\deg(v_0) - L < \ell(A(v_0)) < \frac{c(v_0)}{2}$	$\$(v_0) \geq 2L + \ell(A(v_0)) - \ell(A),$ $\$(v_i) \geq \alpha(v)c(v_i)$ for $i > 0$
6	$V(A) \neq \emptyset$	$\$(v_i) \geq \alpha(v)\tilde{c}(v_i)$ for $i \geq 0$

(a) The six conditions of Lemma 4.4 at a glance. In the charging scheme column, vertices that are not listed get $\$(v) = 0$.



(b) Possible transitions between the conditions.

FIG. 4.1. *High level view of the statement of Lemma* 4.4 *and its proof.*

We will consider the transitions from right to left (as shown in Figure 4.1) starting with the simplest case, which is when the algorithm picks three or more copies of v_0; that is, $\alpha(v_0) \geq 3$, in which case we take the path $(1 \to 6)$ (see the dotted lines in Figure 4.1). We have to argue that each transition is valid.

$(1 \to 6)$ We assume that condition 1 holds for A' and that $\alpha(u) \geq 3$. Let e_u be the last edge assigned to u. Observe that $\ell(e_u) \leq c(u) < L$, since all edges

are small, and $\ell(A(u)) < L + \ell(e_u)$ due to line 7. Hence, $\ell(A(u)) < 2L$. Furthermore, note that $\tilde{c}(u) = c(u)$, since $\deg(u) > c(u)$. Let the charging scheme be $\$(u) = 3L$ and $\$(v) = 0$, for every $v \neq u$. Condition 6 holds, since

$$\alpha(u)\tilde{c}(u) = \alpha(u)c(u) < \ell(A(u)) + c(u) < \tfrac{3}{2}\ell(A(u)) < \tfrac{3}{2}2L = 3L.$$

$(6 \to 6)$ We assume that condition 6 holds for A'. We define a charging scheme $\$ as follows: $\$(u) = 2\deg(u)$, $\$(v) = \$'(v) + \ell(u,v)$ if $v \in V(A)$ and $(u,v) \in E$, and $\$(v) = \$'(v)$ otherwise. Notice that $\$(u) = 2\deg(u) \geq \alpha(u)\tilde{c}(u)$ by Observation 7.

It remains to show that $\$(v) \geq \alpha(v)\tilde{c}(v)$ for every $v \neq u$. First, if $\alpha(v) = 0$ or $\tilde{c}(v) = \tilde{c}'(v)$, then this is clearly true. Let v be a vertex for which $\alpha(v) > 0$ and $\tilde{c}(v) \neq \tilde{c}'(v)$. In this case $\deg'(v) < c(v)$ while $\deg(v) > \deg'(v)$. It follows that $\alpha(v) = \alpha'(v) = 1$. Also, since $\$(v) = \$'(v) + \ell(v,u)$, it follows that $\$(v) \geq \tilde{c}'(v) + \ell(v,u) \geq \tilde{c}(v)$. Thus, condition 6 also holds for A.

Let us now consider the case when Algorithm 2 uses two copies of v_0; that is, $\alpha(v_0) = 2$, in which case we take the path $(1 \to 3 \to 6)$ (see dot-dashed lines in Figure 4.1). We have to argue that each transition is valid.

$(1 \to 3)$ We assume that condition 1 holds for A' and $\alpha(u) = 2$. Let the charging scheme be $\$(u) = 3L$ and $\$(v) = 0$, for every $v \neq u$. Thus, condition 3 holds for A since the charging scheme is valid:

$$\$(u) = 3L > \alpha(u)L \geq \alpha(u)b(u).$$

$(3 \to 3)$ We assume that condition 1 holds for A' and $\alpha(u) = 0$. Then A also satisfies condition 3 since the charging scheme remains valid by the same argument as before.

$(3 \to 6)$ We assume that condition 3 holds for A' and $\alpha(u) \geq 1$. Consider the charging scheme $\$(v_0) = \$'(v_0) + \deg(u)$, $\$(u) = 2\deg(u)$, and $\$(v) = \$'(v)$ for every $v \neq v_0, u$. First, since $\ell(A(u)) = \deg(u) \geq \tilde{c}(u)$, Observation 7 implies that $\$(u) = 2\deg(u) \geq \alpha(u)\tilde{c}(u)$.

As for v_0, first observe that since $\alpha(v_0) = 2$ and $\alpha(u) > 0$, it follows that $c(v_0) < \ell(A(v_0)) < L$. Furthermore, since all edges are small, $A(v_0)$ contains at least two edges. Since edges were added to $A(v_0)$ in a nondecreasing order of demand it follows that $\ell(e_0) \leq \ell(A(v_0))/2$ which means that $\ell(A'(v_0)) - \ell(e_0) < L_0 \leq L'$, where L_0 is the covering requirement when v_0 joined $V(A)$. This implies that $L' > \ell(A(v_0))/2 > c(v_0)/2$. (Recall that $A(v_0) = A'(v_0)$.) It follows that $\$(v_0) = 2L' + L \geq c(v_0) + c(v_0) = \alpha(v_0)\tilde{c}(v_0)$.

Therefore, condition 6 holds for A.

$(6 \to 6)$ Already shown.

Let us now consider the case when Algorithm 2 uses one copy of v_0 and at least half of v_0's capacity is unused; that is, $\alpha(v_0) = 1$ and $\ell(A(v_0)) < \frac{c(v_0)}{2}$, in which case we take the path $(1 \to 5 \to 6)$ (see dashed lines in Figure 4.1). We have to argue that each transition is valid.

$(1 \to 5)$ We assume that condition 1 holds for A' and $\alpha(u) = 1$ and $\ell(A(u)) < \frac{c(u)}{2}$. We claim that in this case $\ell(A(u)) = \deg(u)$. Observe that edges are added to $A(u)$ in a nonincreasing order of demands in lines 6 and 7. Hence, $\ell(A(u)) < \frac{c(u)}{2}$ implies that $A(u) = E(u)$. It follows that $\deg(u) - \ell(A(u)) = 0 < L$. Consider the charging scheme $\$(u) = 3L$ and $\$(v) = 0$, for every $v \neq u$. $\$(u) = 3L \geq 2L + \ell(A(u)) - \ell(A)$ since $\ell(A) = \ell(A(u))$. Hence, condition 5 holds for A.

($5 \to 5$) We assume that condition 5 holds for A' and $\alpha(u) \neq 1$. We first show that $\deg(v_0) - \ell(A(v_0)) < L$. We know that $\deg'(v_0) - \ell(A'(v_0)) < L'$ since A' satisfies condition 5. Since $\ell(A(v_0)) = \ell(A'(v_0))$, $\deg(v_0) \leq \deg'(v_0) + \deg(u)$, and $L = L' + \deg(u)$, it follows that $\deg(v_0) - \ell(A(v_0)) < L$.
If $\alpha(u) = 0$, we define $\$(v_0) = \$'(v_0) + 2\deg(u)$, $\$(u) = 0$, and $\$(v) = \$'(v)$ for every $v \neq v_0, u$. In this case,

$$\begin{aligned} \$(v_0) &= \$'(v_0) + 2\deg(u) \\ &\geq 2L' + \ell(A'(v_0)) - \ell(A') + 2\deg(u) \\ &= 2L + \ell(A(v_0)) - \ell(A). \end{aligned}$$

If $\alpha(u) \geq 2$, we define $\$(v_0) = \$'(v_0) + \deg(u)$, $\$(u) = 2\deg(u)$, and $\$(v) = \$'(v)$ for every $v \neq v_0, u$. In this case,

$$\begin{aligned} \$(v_0) &= \$'(v_0) + \deg(u) \\ &\geq 2L' + \ell(A'(v_0)) - \ell(A') + \deg(u) \\ &= 2L + \ell(A(v_0)) - \ell(A). \end{aligned}$$

Also, $\$(u) = 2\deg(u) \geq \alpha(u)c(u)$ due to Observation 7.
In both cases if $v \in A(v) \setminus \{v_0, u\}$, then $\$(v) = \$'(v) = \alpha'(v)c(v) = \alpha(v)c(v)$. Thus, condition 5 also holds for A.

($5 \to 6$) We assume that condition 5 holds for A' and $\alpha(u) = 1$. We define a charging scheme $\$ as follows: $\$(v_0) = \$'(v_0) + 2\deg(u)$, $\$(u) = \deg(u)$, and $\$(v) = \$'(v)$ for every $v \neq v_0, u$.
First, notice that $\$(v) \geq \alpha'(v)c(v) = \alpha(v)c(v)$, for every $v \in V(A) \setminus \{v_0, u\}$. Also, $\$(u) = \deg(u) \geq \alpha(u)\tilde{c}(u)$ since $\alpha(u) = 1$. It remains to take care of v_0. Observe that

$$\begin{aligned} \$(v_0) &= \$'(v_0) + 2\deg(u) \\ &\geq 2L' + \ell(A'(v_0)) - \ell(A') + 2\deg(u) \\ &> L + \ell(A(v_0)), \end{aligned}$$

because $\ell(A') - L' < \deg(u)$. Also, since $\deg'(v_0) - \ell(A'(v_0)) < L'$, it follows that $\deg(v_0) - \ell(A(v_0)) < L$. Therefore, $\$(v_0) > \deg(v_0) \geq \alpha(v_0)\tilde{c}(v_0)$. Thus, condition 6 holds for A.

($6 \to 6$) Already shown.

The final case is when the algorithm uses one copy of v_0 and at least half of v_0's capacity is used; that is, $\alpha(v_0) = 1$ and $\ell(A(v_0)) \geq \frac{c(v_0)}{2}$, in which case we take the path $(1 \to 2 \to 4 \to 6)$ or the path $(1 \to 2 \to 6)$ (see solid lines in Figure 4.1). This case is slightly more involved than the cases we have seen so far. Let us argue that each possible transition is valid.

($1 \to 2$) We assume that condition 1 holds for A' and $\alpha(u) = 1$ and $\ell(A(u)) \geq \frac{c(u)}{2}$. Let the charging scheme be $\$(u) = 3L$ and $\$(v) = 0$, for every $v \neq u$. Thus, condition 2 holds for A since the charging scheme is valid:

$$\$(u) \;=\; 3L \;>\; \alpha(u)L \;\geq\; \alpha(u)b(u).$$

($2 \to 2$) We assume that condition 2 holds for A' and $\alpha(u) = 0$. Then A still satisfies condition 2 since the charging scheme remains valid by the same argument.

($2 \to 6$) We assume that condition 2 holds for A' and $\alpha(u) = 1$. Observe that $\ell(A(v_0)) < L$ since $\alpha(u) > 0$. It follows that $b(v_0)$ may have increased, since $L > L'$. Hence, we need to increase the number of coins allotted to v_0. Consider the charging scheme $\$(v_0) = \$'(v_0) + 2\deg(u)$, $\$(u) = \deg(u)$, and $\$(v) = \$'(v)$ for every $v \neq v_0, u$. Since $\deg(u) \leq c(u)$, we have $\$(u) = \deg(u) = \alpha(u)\tilde{c}(u)$. Also, $\ell(A(v_0)) < L$ implies that

$$\$(v_0) \;=\; 3L' + 2\deg(u) \;\geq\; 2(L' + \deg(u)) \;=\; 2L \;\geq\; 2\ell(A(v_0)) \;\geq\; c(v_0).$$

Hence, $\$(v) \geq \alpha(v)\tilde{c}(v)$ for every v and condition 6 holds.

($2 \to 4$) We assume that condition 2 holds for A' and $\alpha(u) \geq 2$. As in the previous transition $\alpha(u) > 0$ implies $L' \leq \ell(A(v_0)) < L$. Consider the charging scheme $\$(v_0) = \$'(v_0) + \deg(u)$, $\$(u) = 2\deg(u)$, and $\$(v) = \$'(v)$ for every $v \neq v_0, u$.

Notice that $A(u) = E(u)$ due to line 9, which means that $\ell(A(u)) = \deg(u) > c(u)$. It follows that $\$(u) = 2\deg(u) \geq \alpha(u)c(u)$ due to Observation 7. Moreover, $\$(v_0) \geq L$ since $\$'(v_0) \geq L'$. Hence, it remains to show that $\$(v_0) \geq 2\ell(A(v_0)) - (\ell(A) - L)$. Since $\alpha(u) > 0$ we know that $\deg(u) > \ell(A') - L'$. Also, observe that $\ell(A') - L' = \ell(A) - L$. Hence,

$$\begin{aligned}\$(v_0) \;&\geq\; 2L' + \deg(u)\\ &=\; 2\ell(A') - 2(\ell(A') - L') + \deg(u)\\ &>\; 2\ell(A(v_0)) - (\ell(A) - L),\end{aligned}$$

and condition 4 holds for A.

($4 \to 4$) We assume that condition 4 holds for A' and $\alpha(u) = 0$. Since $A = A'$ the assignment properties hold. Consider the charging scheme $\$(v_0) = \$'(v_0) + \deg(u)$, $\$(u) = 0$, and $\$(v) = \$'(v)$ for every $v \neq v_0, u$. First, $\$(v_1) = \$'(v_1) \geq \alpha(v_1)c(v_1)$. Furthermore, $\$(v_0) = \$'(v_0) + \deg(u) \geq L' + \deg(u) = L$. Also,

$$\begin{aligned}\$(v_0) \;&=\; \$'(v_0) + \deg(u)\\ &\geq\; 2\ell(A'(v_0)) - (\ell(A') - L') + \deg(u)\\ &=\; 2\ell(A(v_0)) - (\ell(A) - L),\end{aligned}$$

and condition 4 also holds for A.

($4 \to 6$) We assume that condition 4 holds for A' and $\alpha(u) \geq 10$. Consider the charging scheme $\$(v_0) = \$'(v_0) + \deg(u)$, $\$(u) = 2\deg(u)$, and $\$(v) = \$'(v)$ for every $v \neq v_0, u$. First, since $\alpha(u) > 0$ we know that $\ell(A') - L' < \deg(u)$. Hence,

$$\begin{aligned}\$(v_0) \;&=\; \$'(v_0) + \deg(u)\\ &\geq\; 2\ell(A'(v_0)) - (\ell(A') - L') + \deg(u)\\ &\geq\; 2\ell(A(v_0))\\ &\geq\; c(v_0).\end{aligned}$$

It follows that $\$(v_0) \geq \alpha(v_0)\tilde{c}(v_0)$. v_1 is funded since $\$'(v_1) \geq \alpha(v_1)c(v_1)$. As for u, $\$(u) = 2\deg(u) \geq \alpha(u)\tilde{c}(u)$. Thus, condition 6 holds for A.

($6 \to 6$) Already shown.

This finishes our case analysis. In every case we have shown that Algorithm 2 makes a valid transition between the conditions. Hence, the lemma follows. ∎

5. Partial capacitated covering with inseparable demands. In this section we show that Algorithm 2 actually computes 3-approximate solutions for PCVC with inseparable demands. We show that the charging scheme that was defined in Lemma 4.4 distributes enough coins in order to fund the additional copies needed due to the inseparable demands.

Given an assignment A, let $\beta(v)$ be the number of copies of v needed when the edges in $A(u)$ are assigned to copies of v using FIRST-FIT in a nonincreasing order of demands.

OBSERVATION 8. *Let A be an assignment that was computed by Algorithm* 2*. If $\alpha(v) \geq 2$, then $\beta(v)c(v) \leq 2\ell(A(v))$.*

Proof. Since β is the number of copies needed using FIRST-FIT, it follows that all copies of v, except maybe one, must be more than half full. The total length of edges assigned to the two most vacant copies of v is more than $c(v)$. Hence, $\beta(v) \leq 2 + \frac{\ell(A(v))-c(v)}{c(v)/2} = \frac{2\ell(A(v))}{c(v)}$. ∎

LEMMA 5.1. *Let A be an assignment that was computed by Algorithm* 2 *for a graph G and a covering demand L. Then, there exists a charging scheme $\$$ (with respect to G and A) such that* (i) $\$(u) \geq \beta(u)b(u)$*, for every vertex u, and* (ii) $\sum_v \$(v) \leq 3L$.

Proof. Let $\$$ be the charging scheme that is defined in the proof of Lemma 4.4. We show that $\$$ satisfies $\$(v) \geq \beta(v)b(v)$ for every v. First, if $\alpha(v) \leq 1$, then $\beta(v) = \alpha(v)$ and by Lemma 4.2 we are done. Let v be a vertex such that $\alpha(v) \geq 2$, and consider the recursive call in which v joined $V(A)$. Since $\alpha(v) \geq 2$, it follows that $\ell(A(v)) > c(v)$. There are two options: v was given edges by either line 6 or by line 9. Line 7 is not involved since $\ell(A(v)) > c(v)$.

If v was given edges by line 9, then $A(v) = E(v)$, and by the definition of $\$$, v gets $2\deg(v) = 2\ell(A(v))$ coins. This can be verified in the transitions $(2 \to 4)$, $(3 \to 6)$, $(4 \to 6)$, $(5 \to 5)$, and $(6 \to 6)$ of Lemma 4.4. Hence, by Observation 8 it follows that $\$(v) \geq \beta(v)c(v) \geq \beta(v)b(v)$.

If edges were assigned to v in line 6, then A satisfies condition 3 ($\alpha(v) = 2$) or condition 6 ($\alpha(v) \geq 3$). If A satisfies condition 6, then $\$(v) = 3L$. Let e_v be the last edge that was assigned to v. Clearly, $\ell(A(v)) - \ell(e_v) < L$. Also, since $\alpha(v) \geq 3$ and all edges are small, it follows that $\ell(e_v) \leq \ell(A(v))/3$. Hence,

$$\$(v) = 3L > 3(\ell(A(v)) - \ell(e_v)) \geq 2\ell(A(v)),$$

and due Observation 8 it follows that $\$(v) \geq \beta(v)c(v) \geq \beta(v)b(v)$.

If A satisfies condition 3, then $\ell(A(v)) - \ell(e_0) < L \leq \ell(A(v))$, where e_0 is the last edge that was assigned to v. First, consider the case where $\ell(A(v)) - \ell(e_0) \leq c(v)$. In this case $\beta(v) = \alpha(v) = 2$ and by Lemma 4.2 we get that $\$(v) \geq \beta(v)b(v)$. Next, consider the case where $\ell(A(v)) - \ell(e_0) > c(v)$. Assume that we use FIRST-FIT to assign the edges from $A(v) \setminus \{e_0\}$ to copies of v, and denote by $\beta'(v)$ the resulting number of copies. Due to Observation 8, it follows that $\beta'(v) \leq 2\ell(A(v)\setminus\{e_0\})/c(v) \leq 2L/c(v)$. Since $L > \ell(A(v) \setminus \{e_0\}) > c(v)$ and $\beta(v) \leq \beta'(v) + 1$, we have that

$$\beta(v)b(v) = \beta(v)c(v) \leq \beta'(v)c(v) + c(v) \leq 3L = \$(v),$$

as required. ∎

By using Lemma 5.1 instead of Lemma 4.4 in the proof of Theorem 4.3, we obtain the following theorem.

THEOREM 5.2. *Algorithm* 2 *computes* 3*-approximations for* PCVC *with inseparable demands.*

6. Partial capacitated covering with separable demands. In this section we present our 3-approximation algorithm for PCVC with separable demands. Our approach is based on educated guessing and a modified version of Algorithm 2.

6.1. Modified algorithm. We modify Algorithm 2 by adding the following line between lines 1 and 2:

1.5: **if** there exists $x \in V$ and $e_x \in E(x)$ such that $\ell(e_x) > \max\{L, c(x)\}$
then return $A(x) \leftarrow \{e_x\}$ and $A(v) \leftarrow \emptyset$ for all $v \neq x$

The modified algorithm now consists of a five-way if condition, where the second if condition states that if there exists an edge e_x incident on a vertex $x \in V$ such that $\ell(e_x) > \max\{L, c(x)\}$, then return the assignment $A(x) \leftarrow \{e_x\}$ and $A(v) \leftarrow \emptyset$ for all $v \neq x$. We refer to this modified algorithm as Algorithm **PCVC-SEP**.

As in the original algorithm there are $O(|V|)$ recursive calls. Hence, the running time of the algorithm is polynomial.

LEMMA 6.1. *Algorithm **PCVC-SEP** computes a partial capacitated vertex cover.*

Proof. The proof is similar to the proof of Lemma 4.1. The only difference is that Algorithm **PCVC-SEP** has two base cases. If the recursion ends in line 1, then $\ell(A) = 0 = L$. Otherwise the recursion ends in line 1.5 and $\ell(A) = \ell(e_x) > L$. In both cases the solution is feasible. The inductive step is identical to the one in the proof of Lemma 4.1. ∎

The assignment returned by Algorithm **PCVC-SEP** is not 3-approximate in general. For example, if there exists a very large edge whose covering is enough to attain feasibility, then line 1.5 will choose to cover the edge no matter how expensive its endpoints may be. Nevertheless, we can still offer the following slightly weaker guarantee.

THEOREM 6.2. *If the recursion of Algorithm **PCVC-SEP** ends in line* 1*, then the assignment returned is* 3*-approximate. Otherwise, if the recursion ends in line* 1.5*, assigning edge e_x to vertex x, then the solution returned is* 3*-approximate compared to the cheapest solution that assigns e_x to x.*

The proof of the theorem is given in section 6.2.

Observe that if all edges are small, then line 1.5 is never executed. Hence, by Theorem 6.2, Algorithm **PCVC-SEP** computes 3-approximations when the instance contains only small edges. In Appendix B it is shown that, in the absence of line 1.5, the algorithm may fail to provide a 3-approximation if the problem instance contains medium or large edges.

Based on Theorem 6.2 it is straightforward to design a 3-approximation algorithm using Algorithm **PCVC-SEP**. First, **PCVC-SEP** is run on the input instance. Suppose the recursion ends in line 1.5 with edge e_x assigned to vertex x. The assignment found is considered to be a *candidate solution* and set aside. Then the instance is modified by detaching edge e_x from x. These steps are repeated until an execution of Algorithm **PCVC-SEP** ends its recursion in line 1 with the empty assignment. At the end, a candidate solution with minimum cost is returned.

THEOREM 6.3. *There exists a* 3*-approximation algorithm for* PCVC *with separable demands.*

Proof. Notice that in the algorithm just described once an edge is detached from an endpoint it cannot be detached from the same endpoint later on. Hence, Algorithm **PCVC-SEP** is executed at most $O(|E|)$ times and the overall running time is polynomial.

We argue by induction on the number of calls to **PCVC-SEP** that at least one of the candidate solutions produced is 3-approximate. For the base case, the first call to

PCVC-SEP ends with the empty assignment, and by Theorem 6.2 the assignment is 3-approximate. For the inductive step, the first run ends with edge e_x assigned to vertex x. If there exists an optimal solution that assigns e_x to x, then by Theorem 6.2 the assignment is 3-approximate. Otherwise, the problem of finding a cover in the new instance, where e_x is detached from x, is equivalent to the problem on the input instance. By inductive hypothesis, one of the later calls is guaranteed to produce a 3-approximate solution. We ultimately return a candidate solution with minimum weight, thus the theorem follows. ∎

6.2. Analysis of Algorithm PCVC-SEP. In the next two sections we pave the way for the proof of Theorem 6.2. As before, our approach involves constructing a subtle charging scheme by which at each point we have at our disposal a number of *coins* that we distribute between the vertices. The charging scheme is later used in conjunction with the local ratio theorem to relate the cost of the solution produced by the algorithm to the cost of an optimal solution. Since Algorithm **PCVC-SEP** has two recursive bases, we prove Theorem 6.2 by using two charging schemes depending on how the recursion ends.

6.2.1. The recursion ends with the empty assignment. We first consider the case where the recursion ends with the empty assignment (line 1). It may seem that Lemma 4.4 already provides a valid charging scheme with respect to G and the computed assignment A in which every vertex v is given at least $\alpha(v)b(v)$ coins. However, the proof of Lemma 4.4 is based on the false assumption that all edges are small. Nevertheless, we will be able to use the charging scheme by fixing only two transitions in the proof:

($1 \to 6$) As in the original proof, let e_u be the last edge assigned to u. Observe that $\ell(e_u) \leq L$, since line 1.5 was not executed, and the rest remains the same.

($3 \to 6$) The transitions remains the same, the only change is that we claim that $A(v_0)$ contains at least two edges, otherwise A' could only have been constructed in line 1.5, and we assume the recursion ends in line 1.

The corrected version of Lemma 4.4 implies the following lemma.

LEMMA 6.4. *Let A be an assignment that was computed by Algorithm **PCVC-SEP** for a graph G and a covering demand L. Furthermore, assume the recursion ended in line* 1. *Then, there exists a charging scheme $\$$ (with respect to G and A) such that* (i) $\$(u) \geq \alpha(u)b(u)$, *for every vertex u, and* (ii) $\sum_v \$(v) \leq 3L$.

6.2.2. The recursion ends with a medium or large edge. It remains to tackle the case where the recursion ends with a medium or large edge. More specifically, it remains to provide a charging scheme for the case where Algorithm **PCVC-SEP** ends its recursion in line 1.5 assigning edge e_x to vertex x. The approach is similar to that used in section 4, the main difference being that since we want to compare our solution to one that assigns e_x to x we have more coins to distribute. Here we say that a charging scheme $\$$ is *valid* if $\sum_v \$(v) \leq 3\max\{L, \alpha(x)c(x)\}$. We show that there is a way to distribute these coins so that $\$(v) \geq \alpha(v)b(v)$. The proof of the next lemma contains a recursive definition of our charging scheme.

LEMMA 6.5. *Let A be an assignment that was computed by Algorithm **PCVC-SEP** for a graph $G = (V, E)$ and a covering demand L. Furthermore, assume the recursion ended in line* 1.5 *assigning edge e_x to x. Then, one the following conditions must hold:*

1. *$V(A) = \{x\}$. Also, the charging scheme $\$(x) = \alpha(x)c(x)$ and $\$(v) = 0$ for every $v \neq x$ is valid.*

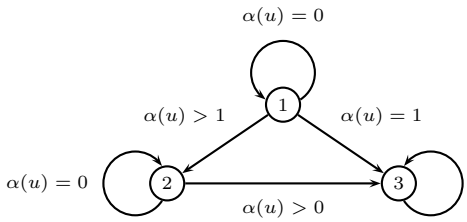


FIG. 6.1. *Possible transitions between the conditions.*

2. $V(A) = \{x, v_1\}$, $\alpha(v_1) > 1$. *Also, the charging scheme* $\$(x) = \alpha(x)c(x)$, $\$(v_1) = 2\ell(A(v_1))$, *and* $\$(v) = 0$ *for every* $v \neq x, v_1$ *is valid.*
3. $V(A) \neq \emptyset$. *Also, there exists a valid charging scheme* $\$$ *such that* $\$(v) \geq \alpha(v)\tilde{c}(v)$ *for every* $v \in V$ *and* $\sum_v \$(v) \leq 3L$.

Proof. We prove the lemma by induction on the length of the creation series of A. We assume that one of the conditions holds and prove that the augmented solution always satisfies one of the conditions. The possible transitions from condition to condition are given in Figure 6.1.

At the base of the induction the computed solution assigns edge e_x to x and condition 1 holds. For the inductive step the same argument used in the proof of Lemma 4.4 handles for recursive calls that occurs in line 2 and 13. Thus, we focus on recursive calls that are made in line 4. We consider a solution A' that was computed by the recursive call, and denote by $\$'$ the charging scheme that corresponds to A'.

Consider a recursive call in which A' satisfies condition 1. In this case, $V'(A) = \{x\}$. There are three possible options:

$(1 \to 1)$ If $\alpha(u) = 0$, then $A' = A$ and condition 1 holds. (This may happen only when $\ell(e_x) = L$.)

$(1 \to 2)$ If $\alpha(u) > 1$, we show that condition 2 holds. Let $\$(u) = 2\deg(u)$ and $\$(v) = \$'(v)$ for any $v \neq u$. Note that $b(u) = c(u) < \deg(u) < L$. Hence, $\$(x) + \$(u) \leq \alpha(x)c(x) + 2\deg(u) \leq \alpha(x)c(x) + 2L$. Therefore, if $L > \alpha(x)c(x)$, then $\$(x) + \$(u) \leq 3L$, otherwise $\$(x) + \$(u) \leq 3\alpha(x)c(x)$.

$(1 \to 3)$ If $\alpha(u) = 1$, then $\ell(A(u)) = \deg(u) < L$ and $\tilde{c}(u) = \deg(u)$. Let $\$(u) = \deg(u)$ and $\$'(v) = \$(v)$ for all $v \neq u$. Thus, $\$(u) = \alpha(u)\tilde{c}(u)$. Since the condition in line 1.5 was not satisfied, we know that $\ell(e_x) \leq L$. Moreover, $\alpha(x)c(x) \leq 2\ell(e_x) \leq 2L$ due to Observation 7. Therefore, $\$(x) + \$(u) = \alpha(x)c(x) + \deg(u) \leq 3L$ and condition 3 holds.

Consider a recursive call in which A' satisfies condition 2. There are two possible options:

$(2 \to 2)$ If $\alpha(u) = 0$, then $A' = A$ and condition 2 holds.

$(2 \to 3)$ If $\alpha(u) > 0$, then consider the charging scheme $\$(u) = 2\ell(A(u))$ and $\$(v) = \$'(v)$ for all $v \neq u$. Notice that $\ell(A(v_1)) + \ell(e_x) < L$ since $\alpha(u) > 0$ (line 9 in u's recursive call); $\ell(A(u)) + \ell(e_x) \leq L$ (line 1.5 in v_1's recursive call) and $\ell(A(u)) + \ell(A(v_1)) < L$ (line 1 in x's recursive call). Furthermore, $\$(x) \leq 2\ell(e_x)$ (by Observation 7), $\$(v_1) = 2\ell(A(v_1))$ and $\$(u) = 2\ell(A(u))$. Adding up these inequalities we get $\$(x) + \$(v_1) + \$(u) \leq 3L$, thus condition 3 holds.

Consider a recursive call in which A' satisfies condition 3. In this case we only have one option: to come back to condition 3. The proof is identical to case $(6 \to 6)$ in Lemma 4.4.

The lemma follows. ∎

LEMMA 6.6. *Let A be an assignment that was computed by Algorithm* ***PCVC-SEP*** *for a graph G and a covering demand L. Furthermore, assume the recursion ended in line* 1.5 *assigning edge e_x to x. Then, there exists a charging scheme $\$$ (with respect to G and A) such that* (i) $\$(u) \geq \alpha(u)b(u)$, *for every vertex u, and* (ii) $\sum_v \alpha(v)b(v) \leq 3\max\{L, \alpha(x)c(x)\}$.

Proof. The lemma follows from Lemma 6.5 and the definition of b. ∎

6.2.3. Proof of Theorem 6.2. For the sake of brevity, when we say that a given assignment is 3-approximate we mean compared to an optimal solution if the recursion of Algorithm **PCVC-SEP** ended in line 1, and compared to the cheapest solution that assigns e_x to x if the recursion ended in line 1.5.

Our goal is to prove that the assignment produced by Algorithm **PCVC-SEP** is 3-approximate. The proof is by induction on the recursion. In the base case the algorithm returns an empty assignment (if the recursion ends in line 1) or assigns e_x to x (if the recursion ends in line 1.5), in both cases the assignment is optimal. For the inductive step there are three cases.

First, if the recursive call is made in line 1.5, then by the inductive hypothesis the assignment A' is 3-approximate with respect to $(V \setminus \{u\}, E)$. A' is clearly 3-approximate with respect to (V, E) since $\deg(u) = 0$.

Second, if the recursive invocation is made in line 4, then by the inductive hypothesis the assignment A' is 3-approximate with respect to $(V \setminus \{u\}, E \setminus E(u))$, w, and $\max\{L - \deg(u), 0\}$. Since $w(u) = 0$, the optimum with respect to (V, E), w, and L is equal to the optimum with respect to $(V \setminus \{u\}, E \setminus E(u))$, w, and $\max\{L - \deg(u), 0\}$. Moreover, since $\alpha(v) = \alpha'(v)$ for every $v \in V \setminus \{u\}$ due to Observation 6, it follows that $w(A) = w(A')$. Thus A is 3-approximate with respect to (V, E), w, and L.

Third, if the recursive call is made in line 13, then by the inductive hypothesis the assignment A' is 3-approximate with respect to (V, E), w_2, and L. If the recursion ended in line 1, then by Lemma 6.4 $w_1(A) \leq \varepsilon \cdot 3L$. We show that $w_1(\bar{A}) \geq \varepsilon \cdot L$ for every feasible assignment $\bar{A}$. First, if there exists $v \in V(\bar{A})$ such that $b(v) = L$, then $w_1(\bar{A}) \geq L \cdot \varepsilon$. Otherwise, $b(v) = \tilde{c}(v) < L$ for every $v \in V(\bar{A})$. It follows that

$$w_1(\bar{A}) = \varepsilon \cdot \sum_v \bar{\alpha}(v)\tilde{c}(v) \geq \varepsilon \cdot \sum_v \ell(\bar{A}(v)) \geq \varepsilon \cdot L.$$

Hence, if the recursion ended in line 1, A is 3-approximate with respect to w_1 too, and by the local ratio theorem it is 3-approximation with respect to w as well. On the other hand, if the recursion ended in line 1.5, then by Lemma 6.6 $w_1(A) \leq \varepsilon \cdot 3\max\{L, \alpha(x)c(x)\}$. We need to show that $w_1(\bar{A}) \geq \varepsilon \cdot \max\{L, \alpha(x)c(x)\}$ for any feasible cover $\bar{A}$ that assigns e_x to x. By the previous argument we know that $w_1(\bar{A}) \geq \varepsilon \cdot L$. In addition, because $e \in \bar{A}(x)$, we have

$$w_1(\bar{A}) \geq \bar{\alpha}(x)w_1(x) \geq \alpha(x)w_1(x) = \varepsilon \cdot \alpha(x)b(x).$$

Since the if condition in line 1.5 was not met in this call we have $\ell(e_x) \leq L$ and so $b(x) = c(x)$. Thus, if the recursion ended in line 1.5, A is 3-approximate with respect to w_1 too, and by the local ratio theorem it is 3-approximate with respect to w as well.

7. Partial capacitated vertex cover in hypergraphs. In this section we consider the partial capacitated vertex cover problem in hypergraphs. We show that our algorithms easily extend to hypergraphs. The resulting approximation ratios are $\Delta + 1$ for PCVC with separable demands or with inseparable demands, and Δ for PCVC with unit demands, where $\Delta \geq 2$ is the maximum size of an edge.

7.1. Notation and terminology. Given a hypergraph $H = (V, E)$, let $E(u)$ be the set of edges incident on u, and let $N(u)$ be the set of vertices that share an edge with u, i.e., $N(u) = \{v : \exists e,\ u, v \in e\}$. As before we define $\deg(u) = \ell(E(u))$. We also denote by $\bar{\ell}(u, v)$ the total load of the edges incident on both u and v. Formally, $\bar{\ell}(u, v) = \sum_{e\,:\,u,v\in e} \ell(e)$. If H is a graph, then $\bar{\ell}(u, v) = \ell(u, v)$ if $(u, v) \in E$, and $\bar{\ell}(u, v) = 0$ otherwise.

Given a PCVC instance, we refer to an edge e as *large* if $\ell(e) > c(u)$ for every $u \in e$. If $\ell(e) \le c(u)$ for every $u \in e$, it is called *small*. Otherwise, e is called *medium*. As in graphs, we may assume without loss of generality that in the case of PCVC with inseparable demands all edges are small. We may also assume without loss of generality that there are no vertices with zero capacity. If there exists such a vertex u, we simply remove u from the hypergraph.

7.2. Nonunit demands. In this section we show that our results for PCVC with separable demands and inseparable demands extend to hypergraphs.

First, we show that the algorithm for PCVC with separable demands (section 6) computes $(\Delta + 1)$-approximate solution on hypergraphs. We do so by extending Theorem 6.2 to hypergraphs. We modify the proof of Theorem 6.2 by fixing Lemmas 4.4 and 6.5.

Consider the case where the recursion ends with the empty assignment. A charging scheme \$ is called *valid* if $\sum_v \$(v) \le (\Delta + 1)L$. We show that if the assignment A was computed by Algorithm **PCVC-SEP**, then there is a way to distribute $(\Delta + 1)L$ coins so that $\$(v) \ge \alpha(v)b(v)$. We do so by fixing the proof of Lemma 4.4. Since $\Delta \ge 2$ it follows that the only transition that should be modified is the transition $(6 \to 6)$. In fact, this transition is the cause for the increase in the approximation ratio. Consider a recursive call in which A' satisfies condition 6. In this case we only have one option:

$(6 \to 6)$ We define a charging scheme \$ as follows: $\$(u) = 2\deg(u)$ and $\$(v) = \$'(v) + \bar{\ell}(u, v)$ for every $v \in V$. First, notice that $\$(u) = 2\deg(u) \ge \alpha(u)\tilde{c}(u)$ by Observation 7. Next, we show that $\$(v) \ge \alpha(v)\tilde{c}(v)$ for every $v \ne u$. First, if $\alpha(v) = 0$ or $\tilde{c}(v) = \tilde{c}'(v)$, then this is clearly true. Let v be a vertex for which $\alpha(v) > 0$ and $\tilde{c}(v) \ne \tilde{c}'(v)$. In this case $\deg'(v) < c(v)$ while $\deg(v) > \deg'(v)$. It follows that $\alpha(v) = \alpha'(v) = 1$. Since $\$(v) = \$'(v) + \bar{\ell}(u, v)$, it follows that

$$\$(v) \ge \tilde{c}'(v) + \bar{\ell}(u, v) = \deg'(v) + \bar{\ell}(u, v) = \deg(v) \ge \alpha(v)\tilde{c}(v).$$

All that is left to show is that the charging scheme is valid. This is easily proved by noticing that the coin distribution was performed only on edges incident on u and that for each such edge no more than $\Delta + 1$ coins were distributed—every vertex in the edge gets one coin except u which gets two coins. Hence, not more than $(\Delta+1)\deg(u)$ coins were distributed. Combined with fact that up until this stage we spent up to $(\Delta + 1)L'$ coins, the total amount of coins used is at most $(\Delta + 1)L' + (\Delta + 1)\deg(u) = (\Delta + 1)L$ and we are done.

Next, consider the case where the recursion ends with a medium or large edge. A charging scheme \$ is called *valid* if $\sum_v \$(v) \le (\Delta + 1)\max\{L, \alpha(x)c(x)\}$. We show that there is a way to distribute these coins such that $\$(v) \ge \alpha(v)b(v)$ for every vertex $v \in V$. We do so by fixing Lemma 6.5 and its proof. First, condition 3 of Lemma 6.5 is changed to

3. $V(A) \ne \emptyset$. Also, there exists a valid charging scheme \$ such that $\$(v) \ge \alpha(v)\tilde{c}(v)$ for every $v \in V$ and $\sum_v \$(v) \le (\Delta + 1)L$.

As for the proof of the lemma, since $\Delta \geq 2$ it follows that the only transition that should be modified is the transition $(3 \to 3)$. The proof of this transition is identical to the transition $(6 \to 6)$ in the hypergraph version of Lemma 4.4.

This leads us to an extended version of Theorem 6.3.

THEOREM 7.1. *There exists a* $(\Delta + 1)$*-approximation algorithm for* PCVC *with separable demands in hypergraphs.*

The proof is identical to that of Theorem 6.3 replacing 3 by $\Delta + 1$.

Given the extended version of Lemma 4.4, it is straightforward to extend Lemma 5.1 and Theorem 5.2 to hypergraphs.

THEOREM 7.2. *Algorithm* 2 *is a* $(\Delta + 1)$*-approximation algorithm for* PCVC *with inseparable demands in hypergraphs.*

7.3. Unit demands. In this section we show that Algorithm 1 computes Δ-approximate solutions for PCVC with unit demands in hypergraphs.

First, we present an extended version of Observation 2 for the case of hypergraphs.

OBSERVATION 9. *Suppose a recursive call is made in line* 4*. Let* $v \in N(u)$*. Then*

1. *If* v *is vulnerable with respect to* G *and* A'*, then* $\alpha(v) = \alpha'(v) = 1$ *and* $\tilde{c}(v) \leq \tilde{c}'(v) + \bar{\ell}(u, v)$*. Thus,* $\alpha(v)\tilde{c}(v) \leq \alpha'(v)\tilde{c}'(v) + (\tilde{c}(v) - \tilde{c}'(v)) \leq \alpha'(v)\tilde{c}'(v) + \bar{\ell}(u, v)$*.*
2. *If* v *is not vulnerable with respect to* G *and* A'*, then* $A(v) = A'(v)$*, and therefore* $\alpha(v) = \alpha'(v)$*. Moreover, there are two (not mutually exclusive) cases:* $\alpha(v) = \alpha'(v) = 0$ *or* $\deg(v) \geq c(v) + \bar{\ell}(u, v)$*. In the latter case the degree of* v *in* G' *is at least* $c(v)$*, and therefore* $\tilde{c}(v) = \tilde{c}'(v) = c(v)$*. Thus in either case* $\alpha(v)\tilde{c}(v) = \alpha'(v)\tilde{c}'(v)$*.*

To analyze the algorithm for the case of hypergraphs, we use a charging scheme by which at each point we have at our disposal ΔL coins that we may distribute between the vertices.

LEMMA 7.3. *Let* A *be an assignment that was computed by Algorithm* 1 *for a graph* G *and a covering demand* L*. Then, there exists a charging scheme* \$ *(with respect to graph* G *and assignment* A*) such that* (i) $\$(u) \geq \alpha(u)b(u)$*, for every vertex* u*, and* (ii) $\sum_v \$(v) \leq \Delta L$*.*

Proof. The proof is almost identical to the proof of Lemma 3.3. The only difference is in recursive calls that where made in line 4. More specifically, in a possible transformation that involves u and its neighbors. In this case, we need only take care of u and its neighbors that participate in the cover. Note that due to Observation 1 we know that $b'(v) < L'$, for every $v \in V(A')$. Hence, $b'(v) = \tilde{c}'(v)$, for every $v \in V(A')$. The inductive hypothesis implies that $\$(v) \geq \alpha'(v)b'(v) = \alpha'(v)\tilde{c}'(v)$. We extend the charging scheme of G' and A' in the following way. For each $e \in E(u)$, if e is assigned to u, then u receives two coins, otherwise we give one coin to each vertex $v \in e$ (including u). Observe that we distribute at most $\Delta \deg(u) = \Delta(L - L')$ coins.

Consider $v \in N(u)$ such that $A(v) \neq \emptyset$. Since $|A'| = L'$ exactly $\deg(u)$ edges are assigned by the while loop. Hence, if v is vulnerable at the beginning of the while loop, it receives either at least $c(v) - c'(v)$ coins or exactly $\bar{\ell}(u, v)$ coins. By Observation 9 this is enough to satisfy v. If v is not vulnerable, then $\$(v) \geq \$'(v)$, and therefore due to Observation 9 we are done. Finally, u is given $|A(u)| + \deg(u)$ coins and it is satisfied due to Observation 4. ∎

The proof of the next theorem is identical to that of Theorem 3.4 where we now use Lemma 7.3 instead of Lemma 3.3.

THEOREM 7.4. *Algorithm* 1 *computes a* Δ*-approximate partial capacitated vertex cover in hypergraphs.*

8. Prize collecting capacitated vertex cover. In this section we extend our results to PC-CVC. Specifically, we show that our algorithms may be used to obtain approximate solutions for PC-CVC with unit, inseparable, and separable demands. The approximation ratios are Δ, $\Delta + 1$, and $\Delta + 1$, respectively.

8.1. Simple reduction to capacitated vertex cover. We first observe that a PC-CVC instance containing a hypergraph with maximum edge size Δ can be viewed as a CVC instance that contains hypergraphs with maximum size edge $\Delta + 1$. Given a PC-CVC instance $(H = (V, E), c, \ell, w)$, we construct a CVC instance $(H' = (V', E'), c', \ell', w')$ as follows. We add a new vertex z_e for every edge $e \in E$, and add z_e to e. The new vertex z_e is called the *anchor* of e. That is, $V' = V \cup \{z_e : e \in E\}$, and $E' = \{e \cup \{z_e\} : e \in E\}$. Observe that the degree of H' is $\Delta' = \Delta + 1$. Furthermore, we define $c'(v) = c(v)$ and $w'(v) = w(v)$ for every $v \in V$, and $c'(z_e) = \ell(e)$ and $w'(z_e) = w(e)$ for every $e \in E$.

This simple reduction implies that PC-CVC can be approximated using algorithms for CVC. That is, to solve PC-CVC we may use the Δ-approximation algorithm for CVC with unit demands and the $(\Delta+1)$-approximation algorithm for CVC with separable demands by Guha et al. [17], and our $(\Delta + 1)$-approximation algorithm from section 5 in the case of CVC with inseparable demands. The resulting approximation ratios are $\Delta + 1$, $\Delta + 2$, and $\Delta + 2$, respectively.

In the rest of this section we assume that a PC-CVC instance is given to us in a form of such a CVC instance, and we show how to improve these ratios to Δ, $\Delta + 1$, and $\Delta + 1$.

8.2. Capacitated vertex cover with separable demands. We consider CVC with separable demands. We show that in this case Algorithm **PCVC-SEP** becomes much simpler. In fact, it converges to a local ratio interpretation of the 3-approximation algorithm for CVC with separable demands from [17].

Examine Algorithm **PCVC-SEP** in the CVC case. In this case $L = \ell(E)$, and therefore the while loop of line 6 always terminates when $A(u) = E(u)$. It follows that the algorithm always skips the while loop in line 7, which means that lines 5–9 can be replaced by $A(u) \leftarrow E(u)$. Furthermore, line 1.5 can be removed since the condition of the if statement is never met, that is, $\ell(e) \leq \ell(E) = L$. Also, notice that $b(v) = \tilde{c}(v)$ for every v. This brings us to Algorithm 3.

Algorithm 3: **CVC**(V, E, w)

1: **if** $E = \emptyset$ **then return** $A(v) = \emptyset$ for all $v \in V$
2: **if** there exists $u \in V$ such that $\deg(u) = 0$ **then return CVC**$(V \setminus \{u\}, E, w)$
3: **if** there exists $u \in V$ such that $w(u) = 0$ **then**
4: $\quad A \leftarrow$ **CVC**$(V \setminus \{u\}, E \setminus E(u), w)$
5: $\quad A(u) \leftarrow E(u)$
6: $\quad$ **return** A
7: Let $\varepsilon = \min_{u \in V} \{w(u)/\tilde{c}(u)\}$
8: Define the weight functions $w_1(v) = \varepsilon \cdot \tilde{c}(v)$, for every $v \in V$, and $w_2 = w - w_1$
9: **return CVC**(V, E, w_2)

In this case, Lemma 6.4 can be replaced by the following.

LEMMA 8.1. *Let A be an assignment that was computed by Algorithm 3 for a graph G. Then, there exists a charging scheme (with respect to graph G and assignment A) such that* (i) $\$(u) \geq \alpha(u)\tilde{c}(u)$, *for every vertex u, and* (ii) $\sum_v \$(v) \leq (\Delta + 1)\ell(E)$.

Proof. We show that either condition 1 holds or condition 6 holds, where we refer to the conditions of Lemma 4.4:

1. $V(A) = \emptyset$. Also, the charging scheme $\$(v) = 0$ for every $v \in V$ is valid.
6. $V(A) \neq \emptyset$. Also, there exists a valid charging scheme $\$$ such that $\$(v) \geq \alpha(v)\tilde{c}(v)$ for every $v \in V$.

To do so we need to prove that the only possible transition from condition 1 is to condition 6. Consider a recursive call in which A' satisfies condition 1. In this case, $V(A) = \{u\}$. Since $L = \ell(E)$ it follows that $\ell(A(u)) = \deg(u)$, and therefore

$$\$(u) = (\Delta + 1)L = (\Delta + 1)\deg(u) \geq \alpha(u)\tilde{c}(u)$$

by Observation 7. ∎

8.3. A more careful analysis. We show that Algorithm 3 actually computes $(\Delta+1)$-approximate solutions on instances of PC-CVC with separable demands, where the maximum size of an edge is Δ. We do so by providing a more careful analysis for this special case of CVC with maximum edge size $\Delta + 1$. Specifically, we show that if an assignment A was computed by Algorithm 3, then there is a way to distribute $(\Delta + 1)L$ coins so that $\$(v) \geq \alpha(v)\tilde{c}(v)$.

LEMMA 8.2. *Let A be an assignment that was computed by Algorithm 3 given a PC-CVC instance with maximum edge size Δ. Then, there exists a charging scheme $\$$ (with respect to graph G and assignment A) such that* (i) $\$(u) \geq \alpha(u)\tilde{c}(u)$, *for every vertex u, and* (ii) $\sum_v \$(v) \leq (\Delta + 1)\ell(E)$.

Proof. The proof is almost the same as the proof of Lemma 8.1. The difference is that in this case we distribute $(\Delta+1)\ell(E)$ coins and not $(\Delta'+1)\ell(E) = (\Delta+2)\ell(E)$ coins. Hence, we need to show how to save $\ell(E)$ coins. (Recall that a PC-CVC instance with maximum edge size Δ can be seen as a PCVC instance with maximum edge size $\Delta' = \Delta + 1$.) The key observation is that there can be two cases: either e is assigned to z_e or to another vertex $v \in e$. If e is assigned to z_e, then it is enough to allot $\deg(z_e) = \ell(e)$ coins to z_e and to every other vertex $v \in e$. (According to the proof of Lemma 4.4 z_e should get $2\ell(e)$ coins.) z_e is satisfied since $\alpha(z_e) = 1$, and therefore $\$(z_e) = \deg(z_e) = \alpha(z_e)\tilde{c}(z_e)$. On the other hand, if e is assigned to a vertex $v \neq z_e$, then $A(z_e) = \emptyset$, and therefore z_e does not receive any coins. It follows that $2\ell(e)$ coins are given to v and at most $(\Delta - 1)\ell(e)$ coins are distributed between the other vertices of the edge e. ∎

The proof of the next theorem is similar to the proof of Theorem 6.2.

THEOREM 8.3. *Algorithm 3 computes $(\Delta + 1)$-approximate solutions in the case of PC-CVC with separable demands.*

Our result can be extended to PC-CVC with inseparable demands as shown in section 5.

THEOREM 8.4. *Algorithm 3 computes $(\Delta + 1)$-approximate solutions in the case of PC-CVC with inseparable demands.*

The arguments of Lemma 8.2 can be applied to PC-CVC with unit demands.

THEOREM 8.5. *Algorithm 1 computes Δ-approximate solutions in the case of PC-CVC with unit demands.*

Appendix A. A bad instance for a natural LP rounding algorithm. Consider the following LP formulation for partial vertex cover with unit demands,

unit weights, and no capacities:

$$\begin{array}{lll} & \min & \sum_{u\in V} x_u \\ \text{(A.1)} & \text{s.t.} & x_u + x_v \geq p_e \qquad\qquad \forall e=(u,v)\in E, \\ \text{(A.2)} & & \sum_{e\in E} p_e \geq L, \\ & & x_u, p_e \geq 0 \qquad\qquad \forall\, u\in V, e\in E, \end{array}$$

where x_u indicates whether u is chosen in the cover, for every vertex $u \in V$, and p_e indicates whether e is covered, for every edge $e \in E$. Constraint (A.1) enforces that if we decide to cover (u, v), either u or v must be chosen, while constraint (A.2) enforces that at least L edges are covered.

There is a natural way of reducing partial vertex cover to regular vertex cover using the above LP formulation: find an optimal fractional solution (x, p), randomly select a set of edges E' so that $\Pr[e \in E'] = p_e$, and finally find a vertex cover in $G' = (V, E')$. From now on, we refer to this approach as the *LP rounding algorithm.* Unfortunately, this approach does not yield a constant factor approximation; in fact, as we will show, it can achieve an approximation factor as bad as $O(n)$ with high probability.

Let G be the complete graph on n vertices, and let $L = nk - \binom{k+1}{2}$, where k is a parameter to be chosen shortly. An optimal partial cover for the instance (G, L) can be formed by selecting any k vertices, since the number of edges covered by k vertices is

$$k(n-k) + \binom{k}{2} = nk + \frac{k(k-1)-2k^2}{2} = nk - \frac{k^2-k}{2} = nk - \binom{k+1}{2}.$$

On the other hand, an optimal fraction solution sets

$$p_e = \frac{L}{\binom{n}{2}} = \frac{nk - \binom{k+1}{2}}{\binom{n}{2}} = \frac{2nk - k(k+1)}{n(n-1)} = \frac{2n-k-1}{n-1}\cdot\frac{k}{n} = \left(2 - \frac{k-1}{n-1}\right)\frac{k}{n}$$

for all $e \in E$ and $x_u = (2 - \frac{k-1}{n-1})\frac{k}{2n}$ for all $u \in V$.

We now show that an optimal vertex cover in $G' = (V, E')$ contains at least $\frac{n}{2} - 1$ vertices with high probability. We do so by proving that the probability that an independent set of size $\frac{n}{2} + 1$ exists in G' is very small. Let S be a subset of vertices of size $\frac{n}{2} + 1$, then the probability that S is an independent set ion G' is

$$\begin{aligned} \Pr[S \text{ is independent in } G'] &= (1-p_e)^{\binom{(n+1)/2}{2}} \\ &\leq (1-p_e)^{\frac{n^2}{8}} \\ &\leq e^{-p_e\frac{n^2}{8}} \\ &= e^{-(2-\frac{k-1}{n-1})\frac{kn}{8}} \\ &\leq e^{-\frac{kn}{8}}, \end{aligned}$$

where the second inequality follows from $1 - x \leq e^{-x}$.

Suppose we set k to be 8. Using a simple union bound argument, the probability that there exists an independent set of size $\frac{n}{2} + 1$ can be upper bounded as follows:

$$\Pr\left[\exists \text{ an IS of size } \frac{n}{2}+1\right] \leq \binom{n}{\frac{n}{2}+1}\cdot e^{-n} \leq 2^n e^{-n}.$$

Thus, with very high probability the vertex cover returned by the LP rounding algorithm has size at least $n/2$, whereas the optimal partial vertex cover for (G, L) costs only 8.

Appendix B. Algorithm PCVC-SEP without line 1.5. In this section we show that, in the absence of line 1.5, Algorithm 2 may fail to provide a 3-approximation if the problem instance contains medium or large edges.

Consider a graph that contains three edges whose endpoints are disjoint. (See Figure B.1.) We define $c(u_1) = c(u_2) = L - 2$, $c(v_1) = c(v_2) = L - 1$, and $c(u_3) = c(v_3) = L$. Also let $\ell(u_1, v_1) = \ell(u_2, v_2) = L - 1$ and $\ell(u_3, v_3) = L$. The weight of the vertices is as follows: $w(u_1) = w(u_2) = L - 2$, $w(v_1) = w(v_2) = L - 1 + \delta$, and $w(u_3) = w(v_3) = L+\delta$, where $\delta > 0$ is a very small constant. We note that the edges in this instance are either small or medium. When given this instance, Algorithm 2 will compute the solution A where $A(u_1) = \{(u_1, v_1)\}$, $A(u_2) = \{(u_2, v_2)\}$, and $A(v) = \emptyset$ for every $v \neq u_1, u_2$. This is because $w(u_1) = b(u_1)$ and $w(u_2) = b(u_2)$, while $w(v) > b(v)$ for every $v \neq u_1, u_2$. Observe that while $w(A) = 4(L - 2)$, there exists a solution that contains a single copy of u_3, and the weight of this solution is $L + \delta$.

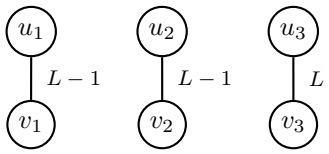


FIG. B.1. *A* PCVC *instance with medium edges.*

Acknowledgments. We thank Einat Or for helpful discussions. We also thank Roee Engelberg and the anonymous referees for their helpful comments and suggestions.

REFERENCES

[1] D. AMZALLAG, M. LIVSCHITZ, J. NAOR, AND D. RAZ, *Cell planning of 4g cellular networks: Algorithmic techniques, and results*, in 6th IEE International Conference on 3G & Beyond, Curran Associates, Inc., Red Hook, NY, 2005, pp. 501–506.

[2] V. BAFNA, P. BERMAN, AND T. FUJITO, *A 2-approximation algorithm for the undirected feedback vertex set problem*, SIAM J. Discrete Math., 12 (1999), pp. 289–297.

[3] J. BAR-ILAN, G. KORTSARZ, AND D. PELEG, *Generalized submodular cover problems and applications*, Theoret. Comput. Sci., 250 (2001), pp. 179–200.

[4] R. BAR-YEHUDA, *Using homogeneous weights for approximating the partial cover problem*, J. Algorithms, 39 (2001), pp. 137–144.

[5] R. BAR-YEHUDA, *One for the price of two: A unified approach for approximating covering problems*, Algorithmica, 27 (2000), pp. 131–144.

[6] R. BAR-YEHUDA, K. BENDEL, A. FREUND, AND D. RAWITZ, *Local ratio: A unified framework for approximation algorithms*, ACM Comput. Surveys, 36 (2004), pp. 422–463.

[7] R. BAR-YEHUDA AND S. EVEN, *A linear time approximation algorithm for the weighted vertex cover problem*, J. Algorithms, 2 (1981), pp. 198–203.

[8] R. BAR-YEHUDA AND S. EVEN, *A local-ratio theorem for approximating the weighted vertex cover problem*, Ann. Discrete Math., 25 (1985), pp. 27–46.

[9] R. BAR-YEHUDA AND D. RAWITZ, *On the equivalence between the primal-dual schema and the local ratio technique*, SIAM J. Discrete Math., 19 (2005), pp. 762–797.

[10] N. H. BSHOUTY AND L. BURROUGHS, *Massaging a linear programming solution to give a 2-approximation for a generalization of the vertex cover problem*, in Proceedings of the 15th Annual Symposium on Theoretical Aspects of Computer Science, Lecture Notes in Comput. Sci. 1373, Springer, Berlin, 1998, pp. 298–308.

[11] J. Chuzhoy and J. Naor, *Covering problems with hard capacities*, in Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science, IEEE, Washington, D.C., 2002, pp. 481–489.
[12] I. Dinur and S. Safra, *The importance of being biased*, in Proceedings of the 34th ACM Symposium on Theory of Computing, ACM, New York, 2002, pp. 33–42.
[13] R. Gandhi, E. Halperin, S. Khuller, G. Kortsarz, and A. Srinivasan, *An improved approximation algorithm for vertex cover with hard capacities*, in Procedings of the 30th Annual International Colloquium on Automata, Languages and Programming, Lecture Notes in Comput. Sci. 2719, Springer, Berlin, 2003, pp. 164–175.
[14] R. Gandhi, S. Khuller, S. Parthasarathy, and A. Srinivasan, *Dependent rounding in bipartite graphs*, in Proceedings of the 43rd IEEE Symposium on Foundations of Computer Science, IEEE, Washington, D.C., 2002, pp. 323–332.
[15] R. Gandhi, S. Khuller, and A. Srinivasan, *Approximation algorithms for partial covering problems*, J. Algorithms, 53 (2004), pp. 55–84.
[16] F. Grandoni, J. Könemann, A. Panconesi, and M. Sozio, *A primal-dual bicriteria distributed algorithm for capacitated vertex cover*, SIAM J. Comput., 38 (2008), pp. 825–840.
[17] S. Guha, R. Hassin, S. Khuller, and E. Or, *Capacitated vertex covering*, J. Algorithms, 48 (2003), pp. 257–270.
[18] J. Guo, R. Niedermeier, and S. Wernicke, *Parameterized complexity of generalized vertex cover problems*, in Proceedings of the 9th International Workshop on Algorithms and Data Structures, Lecture Notes in Comput. Sci. 3608, Springer, Berlin, 2005, pp. 36–48.
[19] E. Halperin and A. Srinivasan, *Improved approximation algorithms for the partial vertex cover problem*, in Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, Lecture Notes in Comput. Sci. 2462, Springer, Berlin, 2002, pp. 161–174.
[20] D. S. Hochbaum, ed., *Approximation Algorithms for NP-Hard Problem*, PWS Publishing Company, Pacitic Grove, CA, 1997.
[21] D. S. Hochbaum, *Solving integer programs over monotone inequalities in three variables: A framework of half integrality and good approximations*, European J. Oper. Res., 140 (2002), pp. 291–321.
[22] R. M. Karp, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, Plenum, New York, 1972, pp. 85–103.
[23] M. Kearns, *The Computational Complexity of Machine Learning*, MIT Press, Cambridge, MA, 1990.
[24] J. W. Lustbader, D. Puett, and R. W. Ruddon, eds., *Symposium on Glycoprotein Hormones: Structure, Function, and Clinical Implications*, Springer-Verlag, New York, 1993.
[25] J. Mestre, *A primal-dual approximation algorithm for partial vertex cover: Making educated guesses*, in Proceedings of the 8th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, Lecture Notes in Comput. Sci. 3624, Springer, Berlin, 2005, pp. 182–191.
[26] P. Slavík, *Improved performance of the greedy algorithm for partial cover*, Inform. Process. Letters, 64 (1997), pp. 251–254.
[27] A. Srinivasan, *Distributions on level-sets with applications to approximation algorithms*, in Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science, IEEE, Los Alamitos, CA, 2001, pp. 588–597.
[28] V. V. Vazirani, *Approximation Algorithms*, 2nd ed., Springer-Verlag, Berlin, 2001.
[29] L. A. Wolsey, *An analysis of the greedy algorithm for the submodular set covering problem*, Combinatorica, 2 (1982), pp. 385–393.