Capacitated Vertex Covering with Applications

Sudipto Guha * Refael Hassin † Samir Khuller ‡ Einat Or §

Abstract

In this paper we study the capacitated vertex cover problem, a generalization of the well known vertex problem. Given a graph $G = (V, E)$ with weights on the vertices, the goal is to cover all the edges by picking a cover of minimum weight from the vertices. When we pick a copy of a vertex, we pay the weight of the vertex and cover upto a pre-specified number of edges incident on this vertex (its capacity). The problem is NP-hard. We give a primal-dual based 2 approximation and study several generalizations, as well as the problem restricted to trees.

1 Introduction

Let $G = (V, E)$ be an undirected graph with vertex set $V = \{1, ..., n\}$ and edge set E. Suppose that w_v denotes the weight of vertex v and k_v denotes the capacity of vertex v (we assume that k_v is an integer). A *capacitated vertex cover* is a function $x : V \rightarrow \mathbb{N}_0$ such that there exists an orientation of the edges of G in which the number of edges directed into vertex $v \in V$ is at most $k_v x_v$. (These edges are said to be *covered* by or *assigned* to v.) The *weight* of the cover is $\sum_{v \in V} x_v w_v$. The MINIMUM CAPACITATED VERTEX COVER problem is that of computing a minimum weight capacitated cover. The problem generalizes the MINIMUM WEIGHT VERTEX COVER problem which can be obtained by setting $k_v = |V| - 1$ for every $v \in V$. The main difference is that in vertex cover, by picking a node v in the cover we can cover all edges incident to v, in this problem we can only cover a subset of at most k_v edges incident to node v.

The problem originated in research at Glycodata[1], a biotechnology company specializing in the areas of glycobiology and bioinformatics. Glycodata develops innovative platform technologies that enable drug discovery, drug development and rational re-design of known drugs. One of its projects related to rational re-design of known drugs involves Glycoproteins. A glycoprotein is a protein that has U attachment points, in which it binds to a glycan. The group, H, of glycans that may appear in each attachment point is known and hence a glycoprotein has $|H|^U$ variants. The goal of this project is to determine which are the building blocks of the glycans comprising the variants of the glycoprotein found in a given (liquid) solution. Such information may be important, for example, when the solution is a known drug (as EPO),since we can determine which variants in the solution contribute more to the activity of the drug, and by that improve the efficacy of the drug. Methods (e.g. HPLC and GC-MS) that identify the building blocks found in a solution exist. However, identifying the building blocks is not sufficient to determine the structure of any given variant found in the solution. It is therefore crucial to determine which of the building blocks are found in each variant, i.e., find a detailed description of the connectivity of the building blocks.

GMID (Glycomolecule ID) is a chip-based technology that is used to generate fingerprints which uniquely identify glycomolecules. It is able to answer in a single application a question of the form: "For a given building block A, and for each member B in a set S of building blocks, does the solution contain a molecule which contains both building blocks A and B? The size of the set S is restricted, because of the specific technology. When planning an experiment that would use the GMID method to obtain information about a given solution, the required information may be presented as a graph where the building blocks are its vertices, and an edge exists between two vertices if the question regarding their connectivity is required. The device is able to answer $|S| = K$ questions at once if they share a common vertex. The problem of minimizing the number of experiments (i.e., GMID uses) needed to cover the required information graph, is *precisely* a capacitated

*Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104. The work was done while the author was at AT&T Research, Florham Park, NJ 07932. E-mail : `sudipto@cis.upenn.edu`

†Department of Statistics and Operations Research, Tel-Aviv University, Tel-Aviv 69978, Israel. Email: `hassin@post.tau.ac.il`

‡Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, MD 20742. Research supported by NSF Awards CCR-9820965 and CCR-0113192. E-mail : `samir@cs.umd.edu`.

§Department of Statistics and Operations Research, Tel-Aviv University, Tel-Aviv 69978, Israel. Email: `eior@post.tau.ac.il`

[1] URL: http://www.glycodata.com

vertex cover problem.

The problem also generalizes another NP-hard problem, namely, covering the edges of a graph by a minimum number of edges and 2-edge paths [5]. This can be seen by setting $k_v = 2$ for every $v \in V$ and $w_v = 1$.

We denote by $\delta(v)$ the edges in E which are incident to v. We also denote by $d(v) = |\delta(v)|$ the degree of $v \in V$. For $S \subseteq V$ we denote by $G(S) = (V, E(S))$ the subgraph induced by S. We denote an edge with end-vertices i, j as a set $\{i, j\}$. We deal with orientation (direction) of edges. Orienting an edge $\{i, j\}$ from i to j is equivalent to replacing it by a directed edge (an arc) (i, j).

Since this problem generalizes vertex cover, perhaps the most studied problem in the area of approximation algorithms [1, 8], it raises several very interesting directions for future research. There are many interesting results known about vertex cover – for example, the bipartite case can be solved in polynomial time, fixed parameter tractability, structural results, special properties about the fractional LP solutions etc [8]. It would be of interest to investigate all these properties in the context of capacitated covering.

Our problem is also related to work on the capacitated facility location problem, for the model where multiple facilities can be opened at the same location, and each facility can only handle at most a specified demand. See Jain and Vazirani [10] and Chudak and Williamson [3] for recent work.

1.1 Summary of Results

The main results that we show are as follows. We give a primal-dual algorithm that yields a factor 2 approximation for the basic problem. We also consider a generalization where each edge has a "demand" of d_e which has to be assigned to an adjacent vertex. For this generalization we show a factor 3 approximation. For the case of hypergraphs (each edge in the hypergraph is a subset of f vertices and must be assigned to one of these f vertices), we can get an f^2 approximation using the LP rounding method described here, as well as a combinatorial algorithm with an f approximation using the primal-dual method (details omitted). We also show how to relate this problem to work done by Hakimi [7] on orientations of graphs. Finally, when the graph is a tree we show that the problem can be solved in polynomial time, but the more general version with edge demands the problem is NP-hard.

One can view Clarkson's greedy algorithm [4] for vertex cover as a primal-dual algorithm. In this (and other algorithms) some vertices are chosen in the final solution. The cost for these vertices is charged to the dual variables corresponding to the adjacent edges. Some edges are charged once and some edges are charged twice. For vertex-cover, the fact that some edges are charged only once does not (apparently) help in improving the approximation bound. For our proof, this savings is crucial and helps us improve the bound from 3 to 2. While the actual algorithm and proof are more complex, at a high level this is the key insight for the improved approximation factor.

Our algorithm has been implemented by Dan Wu[2].

2 Integer programming formulation and a simple LP rounding scheme

A linear integer program (IP) of the problem can be written as follows.

In this formulation, $y_{ev} = 1$ denotes that the edge $e \in E$ is covered by vertex v. Clearly, the values of x in a feasible solution correspond to a capacitated cover. While we do not really need the constraint $x_v \geq y_{ev}$ $v \in e \in E$ for the IP formulation, this constraint will play an important role in the relaxation. (In fact, without this constraint there is a large integrality gap between the best fractional and integral solutions.)

$$\boxed{\begin{array}{ll} \text{Minimize } \sum_v w_v x_v & \\ y_{eu} + y_{ev} \geq 1 & e = \{u, v\} \in E, \\ k_v x_v - \displaystyle\sum_{e \in \delta(v)} y_{ev} \geq 0 & v \in V, \\ x_v \geq y_{ev} & v \in e \in E, \\ y_{ev} \in \{0, 1\} & v \in e \in E, \\ x_v \in I\!N_0 & v \in V. \end{array}} \tag{2.1}$$

We suggest the following algorithm: Solve (2.1) by relaxing the requirement that the variables take integral values. We require that $y_{ev} \geq 0$ and $x_v \geq 0$. If $y_{ev} \geq \frac{1}{2}$ then we define the rounded value $y^*_{ev} = 1$ otherwise we define it as 0. For each edge $e = \{u, v\}$ either y_{eu} or y_{ev} is at least $1/2$, hence the edge can be assigned. (If both the rounded values, y^*_{eu} and y^*_{ev} are 1 then the edge can be assigned to either end.) Clearly, $y^*_{eu} \leq 2y_{eu}$ for the rounded value y^*.

We can now define x^*_v for $v \in V$, as $\lceil \frac{\sum_{e \in \delta(v)} y^*_{ev}}{k_v} \rceil$. We claim that this rounding gives a 4-approximation:

Let $y^*_v = \sum_{e \in \delta(v)} y^*_{ev} = ak_v + k'_v$ where $0 \leq k'_v < k_v$ and $a \geq 0$.

$$y^*_v = \sum_{e \in \delta(v)} y^*_{ev} \leq \sum_{e \in \delta(v)} 2y_{ev} \leq 2k_v x_v$$

$$\Rightarrow ak_v + k'_v \leq 2k_v x_v,$$

[2] http://www.glue.umd.edu/~dandan/CMSC858k/CVC.html

implying that $x_v \geq a/2 + (k'_v/2k_v)$.

Clearly, $x^*_v \leq (a+1)$. We will prove that $x^*_v \leq 4x_v$. It is sufficient to prove that $a+1 \leq 4(a/2 + (k'_v/2k_v))$. We need to show that $a+1 \leq 2a + 2k'_v/k_v$. If $a \geq 1$ we are done. If $a = 0$, then $x^*_v = 1$ while $x_v \geq 1/2$. The last case is when $a = 0$ and $k'_v = 0$ in which case $x^*_v = 0$ and the claim holds.

3 Primal-Dual Algorithm

We develop a primal-dual algorithm that gives a 2-approximation. While the algorithm is quite simple, the proof is somewhat subtle.

The dual problem of the relaxation of (2.1) is given in (3.2):

$$\boxed{\begin{array}{ll} \text{Maximize } \sum_{e \in E} \alpha_e & \\ k_v q_v + \sum\limits_{e \in \delta(v)} l_{ev} \leq w_v & v \in V, \\ q_v + l_{ev} \geq \alpha_e & v \in e \in E, \\ q_v \geq 0, & v \in V, \\ l_{ev} \geq 0, & v \in e \in E, \\ \alpha_e \geq 0, & e \in E \quad v \in V. \end{array}} \tag{3.2}$$

High Level Description of the Algorithm

Initially, no edges are assigned and all vertices are closed. As the algorithm runs, it declares certain vertices as open. When a vertex v is marked open, certain edges are assigned to it. In fact, when v is marked open, *all* unassigned edges that are incident to v are assigned to it. However, later on, if another vertex u that is adjacent to v is opened, an edge between u and v that was previously assigned to v *may* get re-assigned to u. In the end, the algorithm chooses the value of x^*_v to be $\lceil \frac{y_v}{k_v} \rceil$ where y_v is the number of edges assigned to v. The formal description of the algorithm is given in Figure 1.

Initially, all the dual variables α_e are 0. This is a dual feasible solution (with all $q_v = 0$ and $l_{ev} = 0$). We use E' to denote the set of unassigned edges. We use $\delta'(v)$ to denote the unassigned edges currently incident on vertex v. We use $d'(v)$ to denote the number of unassigned edges currently incident on vertex v. We now explain how vertices are marked open. This is done by increasing all the dual variables α_e for the *unassigned edges* $e \in E'$ simultaneously. In the dual program, there are two kinds of constraints - vertex constraints and edge constraints.

To maintain dual feasibility of the edge constraints $q_v + l_{ev} \geq \alpha_e$, as we increase α_e, we have to increase q_v or l_{ev}. If the vertex has a large number of unassigned edges incident to it, then we increase q_v, otherwise we increase l_{ev}. Formally, if $d'(v) > k_v$ then we increase q_v, otherwise we increase l_{ev}.

For each vertex constraint, $k_v q_v + \sum_{e \in \delta(v)} l_{ev} \leq w_v$, initially the left-hand side is 0 and the right-hand side is the weight of the vertex. While increasing the dual variables for the unassigned edges, we stop as soon as a vertex constraint is met with equality. (In Figure 1 this is vertex u in the main loop.) We declare this vertex as open. We assign to this vertex, u, all unassigned edges incident to it and have to stop increasing their dual variables. In addition to assigning edges in $\delta'(u)$ to u, we may also re-assign some of the previously assigned edges from $\delta(u)$ to u. We now elaborate on this point further.

For any vertex v, as soon as $d'(v) \leq k_v$, we *define* D_v to be the set of unassigned edges incident to vertex v. If a vertex has its initial degree at most k_v then this condition is true at the start of the algorithm. For other vertices, the initial degree may exceed k_v, but as edges are assigned, it may happen that at some stage $d'(v) = k_v$. If this event happens we define D_v to be the set of k_v edges from $\delta'(v)$ that are currently unassigned. Later on, $\delta'(v)$ may change but not D_v.

When a vertex v is declared open, if $d'(v) > k_v$ then we assign all unassigned edges in $\delta'(v)$ to v. If $d'(v) \leq k_v$ then we assign all edges in D_v to v. Note that some of these edges may have earlier been assigned to other vertices. (Only the edges in $\delta'(v)$ are previously unassigned.)

The pseudo-code description of the algorithm is given in Figure 1.

THEOREM 3.1. *Min_Capacitated_Cover returns a 2-approximation for* MINIMUM CAPACITATED COVER.

Proof. The algorithm opens a multi-set of vertices S as centers. The total cost of the solution can be represented by $w(S)$, the total weight of the subset S, counting multiple copies. Any vertex v that is declared open has the property that $w_v = k_v q_v + \sum_{e \in \delta(v)} l_{ev}$. We will charge the weight for v (all copies of v) to edges in $\delta(v)$. We will show (Lemma 3.1) that each edge gets a charge of at most $2\alpha_e$. Since the dual solution has value $\sum_e \alpha_e$, and is a lower bound on the optimal solution, we get the required bound. In other words $w(S) \leq 2\sum_e \alpha_e \leq 2w(OPT)$ where OPT is an optimal cover.

LEMMA 3.1. *We can charge the weight of each open vertex v (all copies) to edges in $\delta(v)$ such that each edge e gets a charge of at most $2\alpha_e$.*

Proof. Define a vertex to be a low degree vertex if *when it is declared open* $d'(v) \leq k_v$, otherwise it is defined to be a high degree vertex. We will discuss the charging mechanism for both low degree and high degree vertices.

Min_Capacitated_Cover

input

1. A graph $G = (V, E)$.

2. A capacity function $k : V \to \mathbb{N}_0$.

3. A cost function $w : V \to \mathbb{N}_0$.

output

A capacitated cover.

begin

$E' := E$ [E' *is the set of unassigned edges*].

$V' := V$ [V' *is the set of closed vertices*].

for every $v \in V$

    $\delta'(v) :=$ *edges in* E' *incident with* v. $d'_v := |\delta'(v)|$.

    If $d'_v > k_v$ *then* $D_v := \emptyset$; *otherwise* $D_v := \delta'(v)$.

    $w'_v := w_v$.

    end for

while $E' \neq \emptyset$

    $r_v := \frac{w'_v}{\min(k_v, d'_v)} \quad v \in V'$.

    $u := \arg\min(r_v : v \in V')$.

    $V' := V' \setminus \{u\}$.

    $w'_v := w'_v - r_u \min(k_v, d'_v) \quad v \in V'$.

    if $d'_u > k_u$

      then

        Assign the edges in $\delta'(u)$ *to* u.

      else [$d'_u \leq k_u$]

        Assign the edges in D_u *to* u.

        [*For* $D_u \setminus E'$ *this is a re-assignment.*]

      end if

    for every $\{u, v\} \in \delta'(u)$

        $E' := E' \setminus \{\{u, v\}\}$. $\quad \delta'(v) := \delta'(v) \setminus \{\{u, v\}\}$.

        $d'_v := d'_v - 1$.

        if $d'_v = k_v$

          then $D_v := \delta'(v)$.

          end if

        end for

end while

return

$x^*_v := \left\lceil \frac{|\mathit{edges\ assigned\ to\ v}|}{k_v} \right\rceil$.

end *Min_Capacitated_Cover*

Figure 1: Algorithm Min_Capacitated_Cover

Consider a low degree vertex v. We pick only one copy of the vertex. We will charge the weight of this vertex to *all* edges in the set D_v. All these edges are now assigned to v, regardless of having been assigned earlier. In fact, some of the edges in D_v may be assigned to other vertices later on and are thus charged again.

If $d(v) \leq k_v$ then $D_v = \delta(v)$ and $q_v = 0$, and if the vertex constraint is tight then $w_v = \sum_{e \in \delta(v)} l_{ev} = \sum_{e \in \delta(v)} \alpha_e$. We thus charge the cost of vertex v to all incident edges by charging $\alpha(e)$ to each $e \in \delta(v)$. If $d(v) > k_v$ then at some point of time $d'(v) = k_v$. At this point we fix the value of q_v and subsequently increase the l_{ev} variables. Note that $|D_v| = k_v$. When this vertex is declared open, we have that $w_v = k_v q_v + \sum_{e \in \delta(v)} l_{ev}$. For the edges not in D_v, note that $l_{ev} = 0$. Hence $w_v = k_v q_v + \sum_{e \in D_v} l_{ev}$. Since there are exactly k_v edges in D_v, we have $w_v = \sum_{e \in D_v}(q_v + l_{ev}) = \sum_{e \in D_v} \alpha_e$. Thus, the cost for vertex v is charged to all edges in D_v.

Now consider a high degree vertex v. Suppose $\delta'(v)$ is the set of unassigned edges incident to v when v is declared open. In our charging scheme these will be the only edges that will be charged by v, and previously assigned edges incident on v will not be charged. Some of these edges, a subset $R_v \subseteq \delta'(v)$, will be re-assigned to other vertices.

For a high degree vertex v, we have $l_{ev} = 0$ and $\alpha_e = q_v$ for each unassigned edge when the vertex is declared open. Thus $w_v = k_v q_v$. Since $\alpha_e = q_v$ the cost for a single copy of v needs to be charged to k_v edges. Let $\delta'(v) = p_v k_v + k'_v$ where $0 \leq k'_v < k_v$. If $|R_v| \geq k'_v$ then the number of edges assigned to v is at most $p_v k_v$. In this case the cost for p_v copies can be charged to any $p_v k_v$ edges in $\delta'(v)$. Each edge is charged at most $\frac{w_v}{k_v} = q_v = \alpha_e$. If $|R_v| < k'_v$ then we need $p_v + 1$ copies of v. The cost for these copies is charged to $(p_v + 1)k_v$ edges. We can charge all the edges in $\delta'(v)$ once. We still need to charge $k_v - k'_v$ edges. Since there are at least $p_v k_v$ edges that are not re-assigned, their other ends are not charged. Since $p_v \geq 1$, we have at least k_v edges that we can charge a second time. (If the other end is ever opened, if it is a high degree vertex then it does not re-assign these edges and does not charge them. If it is a low degree vertex then the edge is re-assigned and belongs to R_v. The edges in R_v are charged at most once by v.)

Finally, for any edge $e = \{u, v\}$ if only one end is open then the edge is charged at most twice. If both ends are open, and both are high degree, then only the end that the edge is assigned to can charge it. If both ends are low degree then it is charged at most once from each end. If one end is low degree and one end is high degree, then the edge is assigned to the low degree end and charged once from each end. This completes the proof of the lemma.

4 Approximation with d_e

Consider the case that each edge has demand d_e, and each vertex v satisfies that if r copies of it are open, then summing over the edges assigned to v, $\sum_e d_e \leq r k_v$. In this case we have a 3-approximation. The primal and dual linear programs in this case are:

$$\begin{array}{|lr|}\hline \text{Minimize } \sum_v w_v x_v & \\ y_{eu} + y_{ev} \geq 1 & e = \{u,v\} \in E, \\ k_v x_v - \sum_{e \in \delta(v)} y_{ev} d_e \geq 0 & v \in V, \\ x_v \geq y_{ev} & v \in e \in E, \\ y_{ev} \in \{0,1\} & v \in e \in E, \\ x_v \in \mathbb{N}_0 & v \in V. \\ \hline \end{array} \tag{4.3}$$

$$\begin{array}{|lr|}\hline \text{Maximize } \sum_{e \in E} \alpha_e & \\ k_v q_v + \sum_{e \in \delta(v)} l_{ev} \leq w_v & v \in V, \\ q_v d_e + l_{ev} \geq \alpha_e & v \in e \in E, \\ q_v \geq 0, & v \in V, \\ l_{ev} \geq 0, & v \in e \in E, \\ \alpha_e \geq 0, & e \in E \quad v \in V. \\ \hline \end{array} \tag{4.4}$$

We grow α_e in proportion to the d_e values. If $\sum_{e \in \delta'(e)} d_e > k_v$ raise q_v, otherwise raise l_{ev} appropriately. Once again, as soon as a vertex is open assign all unassigned edges adjacent to it by sufficiently many copies. We do not perform any re-assignments. Let r be the minimum integer such that for all adjacent unassigned edges e, $\sum_e d_e \leq r k_v$.

It is easy to observe that if $r > 1$ we have $r k_v \geq \sum_{e \in \delta'(v)} d_e \geq \frac{1}{2} r k_v$ for assigned edges e. In this case $\alpha_e = d_e q_v$ for all assigned edges. We will charge all edges $2\alpha_e$, and since

$$2 \sum_{e \in \delta'(v)} \alpha_e = \sum_{e \in \delta'(v)} (2 d_e q_v) \geq r k_v q_v = r w_v$$

we can pay for all copies.

If $r = 1$ we open one copy and each adjacent edge pays α_e; in this case as before we may charge an edge already assigned elsewhere. Over all these edges by dual feasibility and the growing process, $\sum_{e \in \delta'(e)} \alpha_e \geq w_v$. Thus as before each edge gets charged at most $3\alpha_e$.

Therefore we can claim the following,

THEOREM 4.1. *For the capacitated vertex cover problem with arbitrary demands on edges and arbitrary capacity and costs of vertices we have a factor 3 approximation.*

It appears that a re-assignment is feasible and that should reduce the cost to $2\alpha_e$ per edge e. However no simple re-assignment exists since the low degree vertex (under-saturated, filled to less than capacity) may get over-saturated and a high degree vertex (over-saturated) becomes under-saturated, thus disallowing any reassignment. This is illustrated in the example below.

4.1 Gap Example We observe that in the above primal dual method as long as we grow α_e in the intuitive fashion as described above and only select the open vertices in the final solution we can only hope for a factor 3 approximation.

Consider the chain of length three defined by vertices $ABCD$. Edges AB and CD have demand 1. BC has demand $k > 2$. Capacity of A, and D are 1. Capacity of B, and C are k. Cost of C is c. Cost of A, and D are $c/k + c\epsilon'$, and of B is $c(1+\epsilon)$ with $\frac{1}{k} > \epsilon' > \epsilon$. If both ϵ, ϵ' are small, the optimal solution is AB being assigned to A, BC to C and CD to D.

It is easy to verify that in our process we will only declare B and C to be open. Since C is cheaper, we can at best have a cost of $3c + c\epsilon$. Thus the best ratio we can hope is $3 - \frac{2+2k\epsilon'-k\epsilon}{k+2+2k\epsilon'} > 3 - \frac{4}{k+4}$.

5 Alternative IP Formulation

If x is a capacitated cover then an orientation of the edges corresponds to the way each edge is assigned to one of its vertices. can be determined by computing a feasible solution to the following system of constraints (which corresponds to a *transportation problem*):

$$\boxed{\begin{array}{ll} y_{eu} + y_{ev} = 1 & e = \{u, v\} \in E, \\ \sum_{e \in \delta(v)} y_{ev} \leq k_v x_v & v \in V. \end{array}} \tag{5.5}$$

We now describe an alternative IP formulation for CAPACITATED VERTEX COVER which involves only vertex variables:

$$\boxed{\begin{array}{ll} \text{Minimize} \sum_v w_v x_v & \\ \sum_{v \in S} k_v x_v \geq |E(S)| & \forall S \subseteq V, \\ x_v \in I\!N_0 & v \in V. \end{array}} \tag{5.6}$$

THEOREM 5.1. *A feasible solution of (5.6) is a capacitated cover.*

Proof. The proof results from a theorem that follows from Hakimi [7] (see [2] for extensions): Given bounds b_v $v \in V$, there exists an orientation of E such that the number of edges oriented to v is at most b_v if and only if $\sum_{v \in S} b_v \geq E(S)$ for every $S \subseteq V$. In our case, $b_v = k_v x_v$.

For completeness we give a direct proof: Call a vertex $v \in V$ *full* if exactly $k_v x_v$ edges are assigned to it. Otherwise call it *available*. Let x be a feasible solution to (4.3). We will show that the edges can be assigned so that the number of edges assigned to each $v \in V$ is at most $k_v x_v$.

For $i = 1, ..., n$ let $V_i = \{1, ..., i\}$ and assume, without loss of generality, that $\{1, 2\} \in E$. We prove by induction on i that $(x_1, ..., x_i)$ is a capacitated cover of the subgraph $G_i = (V_i, E(V_i))$ induced by V_i. For $i = 2$ this trivially holds since $x_1 + x_2 \geq 1$.

Assume that the above property holds for V_{i-1}. Since $x_1, ..., x_{i-1}$ is a capacitated cover of G_{i-1}, the edges $E(V_{i-1})$ can be assigned so that the number of edges assigned to each $j < i$ is at most $k_j x_j$. Denote the set of assigned edges by A_{i-1}.

The induction: Let $E_i = \{\{i, j\} \in E | j < i\}$. Assign $\{i, j\} \in E_i$ to j if j is available. Otherwise assign it to i. If the number of edges assigned to i is at most $k_i x_i$ then we have shown that $(x_1, ..., x_i)$ is a capacitated cover of V_i. Suppose now that the number of edges assigned to i exceeds $k_i x_i$.

Let N_i be the set of vertices in V_{i-1} such that for each $j \in N_i$ there exists a directed path from j to i of assigned edges. Call a vertex in N_i a *leaf* if there are no edges assigned to it. Clearly, a non-leaf vertex v in N_i must have $x_v > 0$.

We now claim that there exists an available vertex in N_i: Assume that the number of edges assigned to i is $k_i x_i + q$ for some $q > 0$ and that every $j \in N_i$ is full. Let $N_i^0 = N_i \cup \{i\}$ and let E_i^0 be the set of edges induced by N_i^0 in G. Then,

$$\sum_{j \in N_i} k_j x_j + k_i x_i = |E_i^0| - q < |E_i^0|.$$

contradicting the feasibility of x.

Let j be an available vertex in N_i and let P be a directed path from j to i. Reverse the orientation of the edges of P. The result is that for all $v \in V_{i-1}$ the property that the number of edges assigned to v is bounded by $k_v x_v$ is maintained, while q is reduced by 1. Repeating this process q times completes the proof.

6 Capacitated covers on trees

Suppose that the MINIMUM CAPACITATED VERTEX COVER has to be solved on a tree $T = (V, E)$. We note that the problem has been shown to be polynomially solvable when $k_v = K$ and $w_v = 1$ for all $v \in V$ [9]. We start by rooting the tree at an arbitrary vertex and renumbering the vertices so that a child of a vertex has a smaller index than its parent. We define T_v as the subtree rooted at v, $v = 1, ..., n$. Algorithm Min_Capacitated_Cover_for_Tree computes for every $v \in V$ two values defined as follows: Let e_v be the edge connecting v and its parent ($e_n = \emptyset$). W_v^{out} is the cost of a minimum capacitated cover of T_v, and W_v^{in} is the cost of a minimum capacitated cover of $T_v \cup \{e_v\}$ under the restriction that e_v is assigned to v.

THEOREM 6.1. *Algorithm Min_Capacitated_Cover_for_Tree (see Figure 2) computes the minimum cost of* CAPACITATED VERTEX COVER *for a tree.*

Proof. The proof is by induction on the hight (maximum number of edges between the root and a leaf) of the tree. We first observe that the algorithm returns an optimal solution for a tree of height 1.

Assume W_v^{in} and W_v^{out} are correct for all $v \in V$, such that the height of T_v is bounded by $h-1$. Consider a vertex $u \in V$ such that the hight of T_u is h. The computation of W_u^{out} and W_u^{in} involves the use of W_v^{in} and W_v^{out} for $v \in A_u$, and these values are correct by the hypothesis.

Consider first the computation of W_u^{out}. This is meant to be the cost of an optimal capacitated-cover of T_u. The contribution to W_u^{out} of $v \in A_u$ is W_v^{out} if $\{v, u\}$ is directed from v to u (that is, $v \in S$), and W_v^{in} otherwise (that is, $v \in F$). In addition, u covers F and this contributes $w_u \lceil |F|/k_u \rceil$.

Assuming that W_u^{out} is correct, we now justify the computation of W_u^{in}. The difference is that now u also covers an extra edge, the one connecting it to its parent. If u is available in the solution determining W_u^{out}, then this extra edge is covered at no additional cost, so that in this case $W_u^{in} = W_u^{out}$. Otherwise, if u is full, $W_u^{in} := W_u^{out} + min(w_u - \sum_{r \in S_k} C_v, min_{v \in F} C_v)$, reflecting the choice between covering the extra edge by increasing x_u by one (and then using the available capacity to cover additional edges from S), or by reassigning one of the edges $\{v, u\}$ that it covers, and paying for this C_v.

Given W_v^{out} and W_v^{in} for $v = 1, ..., n$, one can recursively obtain the assignment of the edges of T as follow: by the values of W_v^{in} and W_v^{out}, $\forall v \in A_n$ the values of C_v can be obtained $\forall v \in A_n$. By using the algorithm on this tree of height 1, one obtains the assignment of the edges and the knowledge whether to use W_v^{in} or W_v^{out} in the next iteration. By repeating this procedure for $u = n$ to $u = 1$ the assignment of the edges in the optimal solution is obtained.

REMARK 1. Consider the more general case of Section 4 in which for all $e \in E$ there is a demand d_e. The MINIMUM CAPACITATED VERTEX COVER problem is NP-hard even on trees. We prove this claim by a reduction from the PARTITION problem. Let $a_1, ..., a_n$ be given non-negative integers such that $\sum_i a_i = 3K$, and suppose we want to check whether there exists a subset of these numbers whose sum is exactly $2K$. Construct a tree of hight one, with root 0 and leaves $1, ..., n$. Let $w_0 = 2K - 1$, $w_u = 1$ for all $u = 1, ..., n$, $k_0 = 2K$, and $k_u = 1$ for $u = 1, ..., n$. Let e_u denote the edge between 0 and u, and set $d_{e_u} = a_u$, $u = 1, ..., n$. We claim that the answer to the partition problem is positive if and only if the cost to this capacitated vertex cover is $3K - 1$: If it is possible to assign exactly total capacity of $2K$ to the root and the rest demand of total size K to the leaves then the cost would be $(2K-1)+K = 3K-1$. If the demand assigned to the root is more than $2K$ then the cost will be at least $4K$ since two centers need to be established there. If less than $2K$ is assigned to the root then the cost would be at least $3K$.

References

[1] R. Bar-Yehuda and S. Even. A linear time approximation algorithm for the weighted vertex cover problem. *J. of Algorithms*, Vol 2:198–203, (1981).

[2] A. Frank, "On the orientation of graphs", *Journal of Combinatorial Theory Series B*, Vol 28:251–261, (1980).

[3] F. Chudak and D. Williamson, "Improved approximation algorithms for capacitated facility location problems", *Proc of 1999 Integer Programming and Combinatorial Optimization Conference*, (1999).

[4] K. Clarkson, "A modification to the greedy algorithm for the vertex cover problem", *IPL*, Vol 16:23–25, (1983).

[5] M. R. Garey, and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, (1978).

[6] M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. *SIAM Journal on Computing*, Vol 24:296–317, (1995).

[7] S. L. Hakimi, "On the degrees of the vertices of a directed graph", *Journal of the Franklin Institute*, Vol 279:290–308, (1965).

[8] D. S. Hochbaum (editor). Approximation Algorithms for NP-hard problems. *PWS Publishing Company*, (1996).

[9] M. Jaeger and J. Goldberg, "A polynomial algorithm for the equal capacity p-center problem on trees", *Transportation Science*, Vol 28:167–175, (1994).

[10] K. Jain and V. Vazirani, "Primal-dual approximation algorithms for metric facility location and k-median problems", *Proc. 40th Ann. IEEE Symp.. Found. Comput. Sci.*, pages 2–13, (1999).

```
Min_Capacitated_Cover_for_Tree
  input
  1. A tree T = (V, E), |V| = n, routed at n.
  The vertices are numbered so that for A_u  v ∈ A_u ⇒ v < u.
  [A_u is the set of children of u.]
  2. A capacity function k : V → N_0.
  3. A weight function w : V → N^+.
  output
  The cost of a capacitated cover.
  begin
  for v = 1 to n.
      W_v^out := 0, W_v^in := w_v.
      end for
  for u = 1 to n.
      for every v ∈ A_u.
              C_v = W_v^in − W_v^out.
              Sort i ∈ A_u in non-increasing order of C_v.
              end for
      while A_u ≠ ∅
          A := the first min(k_u, |A_u|) vertices in A_u.
          if Σ_{v∈A} C_v ≥ w_u
            then Assign all edges {v, u}, v ∈ A to u.
                 A_u := A_u \ A.
            else
                 Assign edges {v, u}, v ∈ A_u to v.
                 A_u := ∅.
            end if
          S := {v : {v, u} is assigned to v}.
          F := {v : {v, u} is assigned to u}.
          S_k := {min(k_v − 1, |S|) highest value vertices in S}.
          W_u^out := Σ_{v∈S} W_v^in + Σ_{v∈F} W_v^out + w_u ⌈|F|/k_u⌉.
          if |F| = 0 (mod k_u)
            then W_u^in := W_u^out + min(w_u − Σ_{r∈S_k} C_v, min_{v∈F} C_v).
            else  W_u^in := W_u^out.
            end if
          end for
  end while
  return W_n^out.
  end Min_Capacitated_Cover_for_Tree
```

Figure 2: Algorithm Min_Capacitated_Cover_for_Tree