A Low-Exponential Algorithm for Counting Vertex Covers

Mark K. Goldberg,* Thomas H. Spencer,† David A. Berque‡

1 Introduction

The counting version of the vertex cover problem, $VC_{\#}$, is defined as follows: given a graph G, compute the number of vertex covers of G whose size is k, for every $k = 1, \ldots, |V(G)|$. Our main result is an algorithm $avc_{\#}$ which, for graphs with maximum vertex degree 3, solves this problem in $\tilde{O}(2^{.2292n})$ time and for general graphs in $\tilde{O}(2^{.325n})$ time. Here $n = |V(G)|$ and $t = \tilde{O}(f(n))$ means $t = O(n^a f(n))$ for some constant a. To evaluate the complexity of $avc_{\#}$ we use a novel accounting technique; we believe that this technique can be applied to a wide class of recursive algorithms.

Standard techniques prove that $VC_{\#}$ is NP-hard even on graphs with maximum vertex degree 3. Since it is unlikely that there is an algorithm for $VC_{\#}$ of complexity $\tilde{O}(2^{o(n)})$, our objective has been the construction of a "low-exponential" algorithm for which the constant α is "sufficiently" small. The value of this constant is important in applications. The authors' own motivation for the development of an algorithm for $VC_{\#}$ is their work on the interactive software system *SetPlayer* (see [2][3]). Computing the exact number of cover sets in a given graph or hypergraph is one of the kernel functions of the system.

*Department of Computer Science, Rensselaer Polytechnic Institute; the work of this author supported in part by the NSA under grant MDA904-90-H-4027 and by the NSF under grants IRI-8900511 and CDA-8805910; this work was partially done while the author was visiting Los Alamos National Laboratory.

†Department of Mathematics and Computer Science University of Nebraska at Omaha; the work of this author supported in part by University Committee on Research, University of Nebraska at Omaha and by the NSF under grants CDA-8805910 and CCR-8810609.

‡Department of Mathematics and Computer Science, DePauw University; the work of this author supported in part by the NSA under grant MDA904-90-H-4027.

Thus, the value of α in the bound determines the maximum size sets that the system can deal with interactively.

Our algorithm uses the notion of *pivoting*. When $avc_{\#}$ pivots on a vertex v, it generates and recursively solves two subproblems: computing the number of vertex covers that contain v and the number that do not. Another important notion used is that of a *terminal graph* defined in Section 3. We prove that every terminal graph has a separator of constant size; this yields a polynomial algorithm for $VC_{\#}$ on terminal graphs. The algorithms are presented in Section 4; their computational complexity is analyzed in Section 5. In Section 2, we describe the technique for computing the number of a nodes in rooted trees that are generated by a recursive algorithm.

Obviously, an algorithm for $VC_{\#}$ yields an algorithm for the *Vertex Cover* problem (see [6]), or equivalently, for the problem of finding the maximum size of an independent set in a given graph. Recursive algorithms for these problems are described in [5],[7], [9], [10], and [13]. Each of the algorithms employs the principle of "dominance": one of the children problems is discarded if another subproblem is guaranteed to yield an independent set (resp. a cover set) at least as large (resp. small) as the first one. The dominance idea, first used in [13] is, in fact, crucial in speeding up all the algorithms. The algorithm in [10] is the fastest for general graphs; it runs in $\tilde{O}(2^{.276n})$ time. The bound for the running time of the algorithm in [5] is $\tilde{O}(2^{.166n})$ for degree-three graphs, but it becomes substantially worse when the degree is higher. Obviously, the dominance principle is not applicable in the case of $VC_{\#}$. However, we have improved the other components of the general recursive methodology: (a) the class of terminal graphs used by our algorithm is substantially wider than that of trees; and (b) a new technique for counting the number of the subproblems generated by the algorithm allows us to consider a more efficient type of reduction rule.

The undirected graphs in this paper are without multiedges and loops. The number of vertices and the number of edges of a given graph G are denoted by $n = n(G)$ and $m = m(G)$, respectively; $deg(v)$ is the degree of v; the *dual degree* of v, $t(v)$, is $\sum_{x \in N(v)} deg(x)$; $k(G) = 2m(G) - 2n(G)$, and $k(G)/n(G)$ is the density of G. The number of vertex covers of G is denoted $c(G)$. If $v \in V(G)$, then $G - v$ denotes the subgraph of G resulting after removal from G of v and all edges

incident to v; $N(v)$ denotes the set of vertices adjacent to v in G. A set $S \subseteq V$ is an α-separator for G if every connected component of the subgraph induced on $V - S$ contains at most $\alpha|V|$ vertices; $|S|$ is called the size of the separator.

2 Evaluation Technique

Every recursive algorithm yields a directed rooted tree whose nodes represent the subproblems generated by the algorithm. Often, a numerical value is assigned to each node of the tree so that (a) the value of every node is greater than the values of its children and (b) a node is a leaf iff its value is 0. This value can be interpreted as the size of the problem represented by that node. Usually, the rate with which the values decrease from a parent to its children may vary for different children of the node; additionally, the rate may differ for different levels of the tree. This lack of uniformity is the core of the difficulty in evaluating the number of the nodes of the tree. In this section, we develop machinery for overcoming this difficulty. In our model, the problem instance is labeled by two numbers, n and k. The rate at which n and k decrease depends on the value of k/n. As k/n decreases, the graph becomes less "dense", and the amount of progress that the algorithm makes may decrease. In our application, the children problems are never more dense than their parents.

Given a sequence $R = \{\rho_1, \ldots, \rho_m\}$ $(0 = \rho_0 < \rho_1 < \ldots < \rho_m)$ and a pair of positive numbers (n, k), we define $rank(n, k; R)$ or simply $rank(n, k)$, if R is understood, to be the largest i for which $\rho_i < k/n$. The following description models how a recursive algorithm generates a binary rooted tree, which we call a *bound_tree*. Every node v of the tree is labeled by a pair of integers $(n(v), k(v))$, so that $k(v) = 0$ iff v is a leaf of the tree. The label of the root, called the *root_label*, is given; the labels of all other nodes are computed using the *parameters* that consist of two sequences:

1. a sequence R of *ranks*; $R = \{0 = \rho_0 < \rho_1 < \rho_2 < \ldots < \rho_m\}$; and

2. a sequence of quadruples $\mathcal{P} = \{P_i\}$, called the *rates* of the tree. For each $i = 1, \ldots, m$, $P_i = (a_i^l, b_i^l, a_i^r, b_i^r)$ is a quadruple whose entries are positive numbers. We require that $\forall i, \quad b_i^l \geq a_i^l \rho_{i+1}$

and $b_i^r \geq a_i^r \rho_{i+1}$; it is easy to prove then that the rank of the label of a parent is not less than that of a child.

Let (n,k) be the label of a non-leaf node v of the bound-tree and let $i = rank(n,k)$. Then the label (n_l, k_l) (resp. (n_r, k_r)) of the left child (resp. the right child) of v is $(n - a_i^l, k - b_i^l)$ (resp. $(n - a_i^r, n_r - b_i^r)$).

It turns out that the number of nodes in the bound_tree with given parameters $(R, \mathcal{P})$ is bounded by the exponential function $c_i 2^{\theta_i k + \epsilon_i n}$, where $i = rank(n,k,R)$. We call the sequence of triples $\{(c_i; \theta_i, \epsilon_i)\}_{i=1}^m$, that define the exponential function, the *resolvent sequence*. This sequence can be constructed recursively.

Base. In the base case, $c_1 = 1$, $\epsilon_1 = 0$, and θ_1 is the positive root of the equation

$$2^{-zb_1^l} + 2^{-zb_1^r} = 2^{-z(b_1^l + b_1^r)}. \qquad (*)$$

Incremental Step. Assume $(c_{i-1}; \theta_{i-1}, \epsilon_{i-1})$ has been defined for some $i > 1$. For the quadruple $q_i = (a^l, b^l, a^r, b^r) \in \mathcal{P}$, let (θ_i, ϵ_i) be the positive solution of the following system of equations:

$$\begin{cases} x\rho_i + y = \theta_{i-1}\rho_i + \epsilon_{i-1}; \\ 2^{-xb^l - ya^l} + 2^{-xb^r - ya^r} = 1. \end{cases} \qquad (**)$$

Furthermore, we let $\sigma = l$ or r, and define d_i and c_i by

$$d_i = \begin{cases} \max_\sigma(-b_i^\sigma \rho_i(\theta_{i-1} - \theta_i) - (\epsilon_{i-1} - \epsilon_i)a_i^\sigma), & if\ \theta_{i-1} \leq \theta_i; \\ 0, & else. \end{cases}$$

$$c_i = c_{i-1} 2^{d_i}.$$

Theorem 1 *Let $T(n,k)$ be the number of nodes in the bound_tree with the root_label (n,k) and parameters $(R, \mathcal{P})$, and let $\{c_i; \epsilon_i, \theta_i\}_{i=1}^m$ be the resolvent sequence of $(R, \mathcal{P})$. Then, there is a constant A such that for every root_label (n,k),*

$$T(n,k) \leq A c_i 2^{\theta_i k + \epsilon_i n},$$

where $i = rank(n,k;R)$.

Proof. We prove the theorem by induction. To satisfy the statement for small values of n and k, we select a constant A sufficiently large. Obviously,

$$T(n,k) = T(n - a^l, k - b^l) + T(n - a^r, k - b^r).$$

Assume that the theorem holds for (n^l, k^l) and (n^r, k^r). If $rank(n^l, k^l) = rank(n^r, k^r) = rank(n, k) = i$, then

$$T(n,k) \le Ac_i \left(2^{\theta_i(k-b^l)+\epsilon_i(n-a^l)} + 2^{\theta_i(k-b^r)+\epsilon_i(n-a^r)}\right) =$$

$$Ac_i 2^{\theta_i k+\epsilon_i n}\left(2^{\theta_i(-b^l)+\epsilon_i(-a^l)} + 2^{\theta_i(-b^r)+\epsilon_i(-a^r)}\right) = Ac_i 2^{\theta_i k+\epsilon_i n}.$$

Thus the induction is straightforward in this case. Suppose, for the sake of concreteness that $rank(n^l, k^l) = rank(n^r, k^r) = i-1 = rank(n, k)-1$ (the other cases are similar and simpler.) Then, for $\sigma = l, r$, we have

$$T(n^\sigma, k^\sigma) \le Ac_{i-1} 2^{\theta_{i-1}k^\sigma+\epsilon_{i-1}n^\sigma}.$$

Using $\theta_i\rho_i + \epsilon_i = \theta_{i-1}\rho_i + \epsilon_{i-1}$ we perform the following transformation

$$\theta_{i-1}(k-b_i^\sigma)+\epsilon_{i-1}(n-a_i^\sigma) = \theta_i(k-b_i^\sigma)+(\theta_{i-1}-\theta_i)(k-b_i^\sigma)+(\theta_i\rho_i+\epsilon_i-\theta_{i-1}\rho_i)n-\epsilon_{i-1}a_i^\sigma$$

$$= \theta_i(k-b_i^\sigma) + \epsilon_i(n-a_i^\sigma) + (\theta_{i-1}-\theta_i)(k-\rho_i n - b_i^\sigma) - (\epsilon_{i-1}-\epsilon_i)a_i^\sigma.$$

Using

$$\frac{k-b_i^\sigma}{n-a_i^\sigma} \le \rho_i < \frac{k}{n} \quad \text{and} \quad -b_i < k - n\rho_i - b_i \le -a_i\rho_i,$$

we prove

$$(\theta_{i-1}-\theta_i)(k-n\rho_i-b_i^\sigma) \le \begin{cases} -a_i^\sigma\rho_i(\theta_{i-1}-\theta_i), & if\ \theta_{i-1} \ge \theta_i; \\ -b_i^\sigma(\theta_{i-1}-\theta_i), & else. \end{cases}$$

Since $-a_i^\sigma\rho_i(\theta_{i-1}-\theta_i)-(\epsilon_{i-1}-\epsilon_i)a_i^\sigma = 0$, the previous inequality implies $\theta_{i-1}(k-b_i^\sigma)+\epsilon_{i-1}(n-a_i^\sigma) \le \theta_i(k-b_i^\sigma)+\epsilon_i(n-a_i^\sigma)+d_i$. Finally, $T(n,k) \le Ac_{i-1}2^{\theta_i(k-b_i^l)+\epsilon_i(n-a_i^l)+d_i} + Ac_{i-1}2^{\theta_i(k-b_i^r)+\epsilon_i(n-a_i^r)+d_i} \le Ac_i 2^{\theta_i k+\epsilon_i n}$. ∎

Thus, if parameters $(R, \mathcal{P})$ are known, the resolvent sequence can be approximated by numerically solving a system of equations. We developed a program *Evaluate* which does these computations.

3 Terminal Graphs

Intuitively, a graph is terminal if it can be reduced to the empty graph by a series of operations that are designed to eliminate "easy-to-handle" parts of the graph. A path is called *straight* if all its vertices with the possible exceptions of the endpoints, have degree two. The graph obtained by applying procedure *Simplify* below to a given graph G is denoted G^*. If $G^* = \emptyset$ then G is called *terminal*.

function *Simplify*;
  repeat
    repeat remove all vertices of degree ≤ 1
    until all remaining vertices have degree ≥ 2;

    replace every straight path of length > 2 by a path of length 2 connecting the same vertices;

    for every collection of straight paths connecting the same pair of vertices, remove all but a shortest one;

  until G does not change;

We can think of *Simplify* as mapping some of the vertices in G to vertices in G^*. For a non-terminal graph G, $deg^*(x)$ denotes the degree of x in G^*. A vertex $x \in V(G^*)$ is called *kernel* if $deg^*(x) > 2$. Further, we identify a kernel vertex $x \in V(G^*)$ with those vertices in G that were mapped by *Simplify* onto x. Two kernel vertices x, y are called $*$-*adjacent* if they are connected in G^* by a straight path. If $u, w \in V(G)$ are $*$-adjacent kernel vertices, and S is the set of all vertices $v \in V(G)$ that are mapped into a straight path connecting them in G^*, then the subgraph of G induced by S is called a $*$-link of G.

A block H of G is called *central* if for every cut vertex $v \in H$, the size of every component of $G - v$ not containing $H - v$ is $\leq (n/2) - 1$. The notion of a central block generalizes that of a centroid in trees (see [8]); the next theorem is a generalization of Jordan's theorem [8] and is proved in a similar way, so we do not present the proof here.

Theorem 2 *Every graph G has a central block. If there is more than one central block, then their intersection is a cut vertex v for which every connected component of $G - v$ is of size $\leq (n/2) - 1$.* ∎

The notion of a bridge of a subgraph H in a graph G is usually used in planarity issues (see [4]); it turns out that it is important for our purpose as well. Let C be a cycle in a terminal graph G and let B be a bridge of C with two vertices of attachment u and v. Denote by $L^+(B)$ and $L^-(B)$ the two segments of C that connect u and v. Also, let $F^+(B)$ and $F^-(B)$ be the connected components of $G - \{u, v\}$ that contain $L^+(B) - \{u, v\}$ and $L^-(B) - \{u, v\}$ respectively. We assume that $|V(F^+)| \geq |V(F^-)|$; if $|V(F^+)| = |V(F^-)|$, the choice of the "+"or the "−"-subgraph is arbitrary. A cycle C is called *stretched* if

for every bridge B with two vertices of attachment

$$|V(F^+)| \geq |V(F^-)| \geq |V(B)|. \qquad (***)$$

Theorem 3 *Every terminal graph G contains a $2/3$-separator of size ≤ 2.*

Proof. If G has more than one central block, then their intersection is an $(1/2)$-separator of size one. Let now H be the only central block of G and let C be a stretched cycle in H. Then for every bridge B of C with two vertices of attachment, $|V(B)| \leq (n+2)/3 \leq n/2$. If B is a bridge of H with one vertex of attachment, then it is composed of blocks different from H. Since H is a central block, $|V(B)| \leq n/2$.

Let $B_1, \ \ldots, B_k$ be the set of bridges with two vertices of attachment and let $L_i^- = L^-(B_i)$ $(i = 1, \ldots, k)$. In the list $L_1^-, \ldots, L_k^-$, remove all repetitions and then remove every segment which is a proper part of another remaining segment in the list; finally renumber the remaining segments, so that they would follow in clock-wise order along C. Let $L_1, \ldots, L_p$ be the resulting list of segments, $B_1, \ldots, B_p$ be the list of corresponding bridges and let $\{w_i\}$ $(i = 1, \ldots, r)$ be the vertices of attachment[1] of the remaining bridges. It is obvious, that $\forall i, j = 1, \ldots, r, i \neq j$, $\{w_i, w_j\}$ is a cut set of G. If B is a block from the list and $|V(F^+(B))| \leq 2n/3$, then the attachment vertices of B is a $2/3$-separator. Thus, we assume that for every $i = 1, \ldots, p$, $|V(F^+(B_i))| > 2n/3$, which implies that

$$|V(F^-(B_i)| + |V(B_i)| \leq n/3. \qquad (****)$$

Now, select two adjacent blocks , say B_1 and B_2, with the vertices of attachment u_1, v_1 and u_2, v_2 respectively, and let x be a vertex lying in between v_1 and u_2. Let v_1, v_2 not be a $2/3$-separator. Then, starting with v_2 we move along C to find the last vertex of attachment w_j in C for which the size of the component of $G - \{v_1, w_j\}$ containing L_1^- is bigger than $2n/3$. If w_j, w_{j+1} are the vertices of attachment of a block in the list, then v_1, w_{j+1} is a $2/3$-separator, because of (**). Otherwise, there is a vertex y in between w_j and w_{j+1} for which v_1, y make a $2/3$-separator. ∎

[1] it can be that $r < 2p$, since some of the blocks may share their attachment vertices.

4 The Algorithm

As a recursive algorithm, $avc_{\#}$ is described completely if we show how it reduces non-terminal graphs to smaller graphs and how it acts on the terminal graphs. For the sake of simplicity, below we describe an algorithm for counting all cover sets in a given graph. This algorithm can be easily changed to produce a procedure for computing the number of cover sets of a given size; the modification will only change the polynomial factor in the running time function.

Lemma 1 *If C_n (resp. P_n) denotes a cycle of length n (resp. a path of length n), then $c(C_n) = F(n+1)$ (resp. $c(P_n) = F(n)$).*

Proof. Obviously, $c(P_0) = 2 = F(0)$ and $c(P_1) = 3 = F(1)$. Now let $n \geq 2$, v be an endpoint of P, and u be adjacent to v. Then $c(P)$ equals the number of cover sets of P that contain v plus the number of those that do not contain v. The latter must contain u, implying $c(P_n) = c(P_{n-1}) + c(P_{n-2})$ and $c(P_n) = F(n-1) + F(n-2) = F(n)$. The equality for cycles is proved similarly. ||

Procedure *Terminal* below uses two subroutines: *Find_Separator* and *Pivot*. The first finds a separator of size two in a terminal graph; the second is usual pivoting on two vertices which produces four graphs. *Find_Separator* can always be implemented as a search over all pairs of two elements, but it is also can be implemented following the proof of the theorem 3. In both cases, the procedure is polynomial.

Procedure *Terminal*;
/*Input: $G(V, E)$; Output: $c(G)$ */

if G is disconnected and $G_1, \ldots, G_p$ are the components of G **then** $c(G) = \prod_{i=1}^{p} c(G_i)$;
else
if G is a path or a cycle
use Lemma to compute $c(G)$;
else{
$x, y = Find\_Separator(G)$;
$G^1, G^2, G^3, G^4 = Pivot(G; x, y)$;
$c(G) = c(G^1) + c(G^2) + c(G^3) + c(G^4)$};

For the general case, if an input to $avc_{\#}$ is a disconnected graph, then the algorithm is recursively applied to each component, and the answers are used to compute the function for the whole graph. For a connected graph, $avc_{\#}$ first applies procedure *Simplify* to check if the graph is terminal. If it is, procedure *Terminal* computes the answer. For a non-terminal connected graph, $avc_{\#}$ pivots on a vertex whose selection is based on the reduced graph G^*. As a rule, the pivot is a vertex of maximum degree which has the maximum dual degree. In fact, if $k/n > 4$, $avc_{\#}$ always follows this rule. The only exception occurs if G^* is not biconnected and $k/n \leq 4$; in this case, $avc_{\#}$ pivots on a cut vertex v which minimizes the number of vertices in the largest connected component of $G - \{v\}$.

5 The Analysis.

It is easy to see that $avc_{\#}$ is polynomial on terminal graphs; for general graphs, the algorithm is exponential. Given a graph G, $T(G)$ denotes the running time of $avc_{\#}$ on G; then $T(n) = \max_{|V(G)|=n} T(G)$. The starting point of our analysis is the following simple lemma which reduces the problem to simplified graphs.

Lemma 2 *Let v be a pivot, G_l and G_r be respectively the left and the right children-graphs obtained by using v, $G^* =$ Simplify(G), and let G^*_l and G^*_r be the left and the right children-graphs obtained by using pivot v considered as a vertex in G^*. Then Simplify(G_l) =Simplify(G^*_l) and Simplify(G_r) =Simplify(G^*_r). For an arbitrary graph H, $k(H) \geq k(H^*)$.* ||

If the pivot is a cut vertex of G, then both G_l and G_r are disconnected, so the problem is reduced to those on their components. Alternatively, it may be the case that removing $\{v\} \cup N(v)$ disconnects the graph, so G_r is disconnected. In either case, $T(G)$ satisfies inequality $T(G) \leq \sum T(H_i)$, where the graphs H_i are disjoint. Since $T(n)$ is exponential, it is dominated by $T(\max_i |H_i|)$. This means that we need only consider the size of the largest connected component of G. Below G, G_l and G_r, respectively, denote a simplified graph, its simplified largest component of the left and right children. If v is the pivot, then quadruple $q(v) = (n-n^l, k-k^l, n-n^r, k-k^r)$, where n^l, k^l (resp. n^r, k^r) refer to G^l (resp. G^r) is called *the rate of* v. We say, that

quadruple $q = (x_1, x_2, x_3, x_4)$ dominates quadruple $p = (y_1, y_2, y_3, y_4)$ and write $q \geq p$, if $x_i \geq y_i$ $(i = 1, 2, 3, 4)$; two quadruples p and q are *incomparable* if neither dominates the other. If $p_1 = q(v_1)$, $p_2 = q(v_2)$ and $p_1 \geq p_2$, then pivoting on v_1 leads to a better estimate of the worst case running time.

Lemma 3 *Let G be a simplified graph, v be a pivot selected by $avc_\#$ $d = d(v)$, $t = t(v)$, and $a = 0$ (resp. $a = 1$) if $t - d$ is even (resp. $t - d$ is odd). Then (1) $nq(v) \geq (1, 2d-2, d+1, 2d-2)$; (2) if G is biconnected, then $q(v) \geq (1, 2d-2, d+1, t-d+a)$; and (3) if v is a cut vertex, then $q(v) \geq (4, 8, 6, 8)$.*

Proof: Parts (1) and (2) of the lemma have three inequalities in common that are readily proved. To prove the fourth inequalities of (1) and (2), let $G' = G - \{v \cup N(v)\}$, let l be the number of edges connecting $N(v)$ with $V - \{v \cap N(v)\}$, and also let d_2 be the number of degree two vertices in $N(v)$. Since $\sum_{x \in v \cup N(v)} deg(x) = t + d$, $m(G') = (t + d + l)/2$ and $k(G) - k(G') = t - d + l - 2$. If G is biconnected, then $l \geq 2$ because otherwise G would have a cut vertex. For a general simplified graph G, we notice that no degree two vertex in $N(v)$ is adjacent to another vertex in $N(v)$, which implies $l \geq d_2$. Together with $t \geq 2d_2 + 3(d - d_2) = 3d - d_2$, we have $k(G) - k(G') \geq 2d - d_2 + l - 2 \geq 2d - 2$. The proof of (1) and (2) is completed by applying lemma 6 to G'. Proving (3) can be done by a case analysis, which we omit here. ∎

To establish, for given k/n, which values of (d, t) are possible, we formulate a linear program LP, which provides an upper bound, for given d and t, on the average vertex degree in an n-vertex graph with $d = \max_{v \in V(G)} d(v)$ and $t = \max_{d(v)=d} t(v)$. Let n_d and $n_{i,j}$ denote, respectively, the number of vertices of degree d and the number of vertices of degree i adjacent to exactly j vertices of degree d; the variables for the LP are $x_d = n_d/n$ and $x_{ij} = n_{ij}/n$ $(0 \leq j \leq i < d)$.

LP: $\max(dx_d + \sum_{0 \leq j \leq i < d} i x_{ij})$,
Subject to

(1) $x_d,\ x_{i,j} \geq 0\ \forall 0 \leq j \leq i < 0$;
(2) $x_d + \sum_{0 \leq j \leq i < d} x_{ij} = 1$;
(3) $\sum_{0 \leq j \leq i < d} (d - i) j x_{ij} \geq (d^2 - t) x_d$;
(4) $\sum_{0 \leq j \leq i < 0} j x_{i,j} \leq d x_d$.

The objective function expresses the average of the degrees of the vertices, that is the value $2m(G)/n$. The first constraint is obvious; the second constraint expresses the number of vertices in a graph; the fourth constraint is easily proved if we consider the bipartite graph, for which one part consists of all vertices of degree d, and the other part takes all other vertices. To prove the third inequality, denote m_i to be the number of edges between the vertices of degree i and those of degree d. Then, $\sum_{0\le i<d}(d-i)m_i \ge (d^2-t)n_d$. On the other hand, $m_i = \sum_{0\le j\le i} jn_{ij}$, which together with the previous inequality implies the inequality (3) in LP. ∎

Let $\delta(d,t)$ denote the value of the objective function of LP and let rank-sequence R be defined as the reordered set of those values $\delta(d,t)$ for which $d = \lceil \delta(d,t) \rceil$. Obviously, if for a graph G, $k(G)/n(G) > \delta(d,t)$ and $d(G) = d$, then $t(G) \ge t+1$. Thus, for each rank $\rho \in R$, we define $(1, 2d-2, d+1, t+1+a-d)$ (see lemma 7) to be the rate corresponding to the ρ. The ordered set $\mathcal{Q}$ of these quadruples is defined to be the rate-sequence.

ρ_{i+1}	0.5000	1.0000	1.4285	1.6000	2.0000	2.2857	2.5000	
ϵ_i	0.0000	0.0308	0.1235	0.1488	0.1684	0.2442	0.2514	
θ_i	0.2500	0.1884	0.0957	0.0780	0.0657	0.0278	0.0247	
d	3	3	4	4	4	5	5	
t	7	9	12	14	16	19	21	
ρ_{i+1}	2.6667	3.0000	3.2174	3.4545	3.5556	3.7143	4.0000	4.1764
ϵ_i	0.2581	0.2642	0.3177	0.3181	0.3186	0.3190	0.3194	0.3574
θ_i	0.0220	0.0197	0.0019	0.0017	0.0016	0.0015	0.0014	-.0081
d	5	5	6	6	6	6	6	7
t	23	25	28	30	32	34	36	39

Table 1. Degrees, Dual Degrees, and Running Times as a Function of Density.

The table above presents the values of ρ, ϵ and θ's with corresponding values of d and t, where $3 \le d \le 7$; $2d \le t \le d^2$; they were computed by using a publicly available package LINDO[11]. We are now ready to state the following

Theorem 4 *If $k/n \le 4$, the running time of $avc_{\#}$, as a function of k*

and n is the $\tilde{O}(2^{\theta_i k+\epsilon_i n})$, where θ_i and ϵ_i are given by Table 1 above. If $k/n > 4$, then $avc_{\#}$, runs in $\tilde{O}(2^{0.3250n})$ time.

Proof. Specifically, we will prove by induction on n and k that $T(n,k) \leq A2^{\theta_i k+\epsilon_i n}$, where (θ_i, ϵ_i) are given in Table 1, for $0 \leq i \leq 13$, and $(\theta_{14}; \epsilon_{14}) = (0; 0.3250)$. For small n and k, we make the base case hold by choosing A to be big enough. At each step $avc_{\#}$ pivots on some vertex v whose rate is (a^l, b^l, a^r, b^r) so we need to show that $T(n,k) \leq T(n-a^l, k-b^l) + T(n-a^r, k-b^r)$.

Let us first consider the case where both child problems have the same rank as their parent. In this case we need to show that

$$A2^{\theta_i k+\epsilon_i n} \geq A2^{\theta_i(k-b^l)+\epsilon_i(n-a^l)} + A2^{\theta_i(k-b^r)+\epsilon_i(n-a^r)}.$$

This is equivalent to showing that

$$1 \geq 2^{-\theta_i b^l-\epsilon_i a^l} + 2^{-\theta_i b^r-\epsilon_i a^r}. \qquad (*)$$

If $k/n > 4$, $avc_{\#}$ pivots on a vertex v of degree seven (or more), so $a^l \geq 1$ and $a^r \geq 8$. Since $\theta_{14} = 0$, we don't care about the values of b^l and b^r. The theorem holds in this case since $1 \geq 2^{-.3250} + 2^{-8(.3250)}$.

Now let us consider the case where $k/n < 4$. Again $avc_{\#}$ pivots on a vertex v. In general, there are three incomparable possibilities for the rate of v. One quadruple comes from the possibility that $d = \lceil k/n \rceil + 2$; another from the possibility that $d > \lceil k/n \rceil + 2$; and the third comes from the fact v may be a cut vertex. The quadruples that come from the first case were used to calculate the θ_i and the ϵ_i, using the technique of section 2. Thus, $(*)$ holds with equality for these quadruples. The inequality $(*)$ can easily be verified for the other two quadruples using Table 1 and a calculator.

The case where a child problem has a different rank from its parent can be analyzed similarly, so we omit it here. ∎

References

[1] Berque, D., *Implicit Set Manipulation: Theory and Practice,* Ph.D. Thesis, Computer Science Department, RPI, (1991).

[2] Berque, D., R. Cecchini, M. Goldberg, R. Rivenburgh The *Set-Player* System: An Overview and A User Manual, *Tech. Rep., 91-17,* Comp. Sci. Dept., RPI, (1991).

[3] Berque, D., R. Cecchini, M. Goldberg, R. Rivenburgh The *Set-Player* System for Symbolic Computation on Power Sets, (to appear in J. of Symbolic Computation).

[4] Bondy, J. A., U.S.R. Murty, *Graph Theory with Applications,* North Holland, New York • Amsterdam • Oxford, 1970.

[5] Coppersmith, D., U. Vishkin, Solving NP-hard problems in "Almost Trees": vertex cover, *Discr. Appl. Math.,* **10** (1985), pp. 27-45.

[6] Garey, M.R., D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, W.M. Freeman and Co, 1979.

[7] Gurevich, Y., L. Stockmeyer, and U. Vishkin, Solving NP-hard problems on graphs that are almost trees and an application to facility location problems, *Journal of the ACM*, Vol 31, (1984), 459-473.

[8] Harary, F., *Graph Theory*, Addison-Wesley, Reading, Mass., 1969.

[9] Jian, T., An $O(2^{0.304n})$-Algorithm for Solving Maximum Independent Set Problem, *IEEE Transactions on Computers,* vol.C-35, No. 9, (1986), pp. 847-851.

[10] Robson, J.M., Algorithms for Maximum Independent Sets, *J. of Algorithms,* 7, (1986), 425-440.

[11] Schrage, *User's Manual for Linear, Integer, and Quadratic Programming with LINDO, Release 5.0,* The Scientific Press, South San Francisco, 1991.

[12] Schroeppel, R, A. Shamir, A $T = O(2^{n/2}), S = O(2^{n/4})$-algorithm for certain NP-Complete Problems, *SIAM J. Comp.,* Vol. 10, No. 3, (1981), pp. 456-464.

[13] Tarjan, R., A. Trojanowski, Finding a Maximum Independent Set, *SIAM J. Comput.,* 6 (1977), pp. 537-546.

[14] Valiant, L.G., The Complexity of Enumeration and Reliability problems, *SIAM J. of Computing*, Vol. 6, No.3, (1979), pp. 189-201.