ELSEVIER Artificial Intelligence 83 (1996) 167-188

Artificial Intelligence

# Optimization of Pearl's method of conditioning and greedy-like approximation algorithms for the vertex feedback set problem [★]

Ann Becker [1], Dan Geiger [*]

*Computer Science Department, Technion, Haifa 32000, Israel*

Received March 1994; revised January 1995

**Abstract**

We show how to find a small *loop cutset* in a Bayesian network. Finding such a loop cutset is the first step in the method of conditioning for inference. Our algorithm for finding a loop cutset, called MGA, finds a loop cutset which is guaranteed in the worst case to contain less than twice the number of variables contained in a minimum loop cutset. The algorithm is based on a reduction to the weighted vertex feedback set problem and a 2-approximation of the latter problem. The complexity of MGA is $O(m + n \log n)$ where $m$ and $n$ are the number of edges and vertices respectively. A greedy algorithm, called GA, for the weighted vertex feedback set problem is also analyzed and bounds on its performance are given. We test MGA on randomly generated graphs and find that the average ratio between the number of instances associated with the algorithm's output and the number of instances associated with an optimum solution is far better than the worst-case bound.

## 1. Introduction

Most inference algorithms for the computation of a posterior probability in general Bayesian networks have two conceptual phases. One phase handles operations on the graphical structure itself and the other performs probabilistic computations. For example, the clique tree algorithm requires us to first find a "good" clique tree and then perform

[★] Originally submitted as a Research Note. This research was supported by the fund for the promotion of research at the Technion.

[*] Corresponding author. E-mail: dang@cs.technion.ac.il.

[1] E-mail: anyuta@cs.technion.ac.il.

probabilistic computations on the clique tree [17]. Pearl's method of conditioning requires us first to find a "good" loop cutset and then perform a calculation for each loop cutset [19,20]. Finally, Shachter's algorithm requires us to find a "good" sequence of transformations and then, for each transformation, to compute some conditional probability tables [21].

In the three algorithms just mentioned the first phase is to find a good discrete structure, namely, a clique tree, a cutset, or a sequence of transformations. The goodness of the structure depends on a chosen parameter that, if selected appropriately, reduces the probabilistic computations done in the second phase. Finding a structure that optimizes the selected parameter is usually NP-hard and thus heuristic methods are applied to find a reasonable structure. Most methods in the past had no guarantee of performance and performed very badly when presented with an appropriate example. For example, the greedy algorithms of [23,24] for the method of conditioning may in the worst case perform as bad as a factor of $n/2$ where $n$ is the number of variables in a Bayesian network (as shown in [23]). That is to say, the size of the loop cutset found by these algorithms can include as many as $n/2$ variables instead of just 2—a disastrous outcome. Similar situations occur with other inference algorithms.

However, recently, among other results, Bar-Yehuda et al. [1] have developed an algorithm that finds a loop cutset that is guaranteed in the worst case to contain less than 4 times the number of variables contained in a minimum loop cutset. This guarantee is given only when the number of values of every variable in the network is the same. Note that this result means that the number of instances associated with a loop cutset $F$ found by their algorithm (e.g., $r^{|F|}$ if the number of values of every variable is $r$) is no more than the number of instances associated with a minimum loop cutset raised to the forth power. Note also that the problem of finding a minimum loop cutset was shown to be NP-hard in [24].

We offer a new algorithm for finding a loop cutset, called MGA, that finds a loop cutset which is guaranteed in the worst case to contain less than twice the number of variables contained in a minimum loop cutset. That is, the number of instances associated with a loop cutset found by our algorithm is no more than the number of instances associated with a minimum loop cutset raised to the second power. The complexity of MGA is $O(m + n \log n)$ where $m$ and $n$ are the number of edges and vertices respectively.[2] Unlike [1], our result holds even when the number of values changes from one variable to another. Like [1], our solution is based on a reduction to the weighted vertex feedback set problem, defined in the next section. We should emphasize that all these performance guarantees are for the worst case.

In Section 4 we test MGA on randomly generated graphs and find that the average ratio between the number of instances associated with the algorithms' output and the number of instances associated with an optimum solution is far better than the worst-case bound.

From a theoretical point of view, Bar-Yehuda et al. [1] note that as the number of variables grows to infinity the worst-case ratio between the size of a loop cutset found by any polynomial algorithm and the size of a minimum loop cutset cannot be less than

[2] Throughout, $\log x$ stands for $\log_e x$.

two unless a similar result is obtained for the *weighted vertex cover problem* (WVC).[3] Consequently, we conjecture that no polynomial algorithm for the loop cutset problem performs better in the worst case than the algorithm presented in this paper as graphs grow to infinity in size.

We should note that another application of MGA is in the area of constraint satisfaction [6,7]. Dechter and Pearl use a vertex feedback set in order to solve constraint satisfaction problems in such a way that the search complexity depends on the size of the vertex feedback set utilized by their algorithm.

The rest of the paper is organized as follows. In Section 2 we outline the method of conditioning, explain the related loop cutset problem and describe the reduction from the loop cutset problem to the *weighted vertex feedback set problem* (WVFS); Section 2 summarizes previous works. In Section 3 we provide two new approximation algorithms for the WVFS problem which is by itself an NP-complete problem [9, pp. 191-192]. Finally, in Section 4 we present experiments that test the average performance of our algorithms.

## 2. The loop cutset problem

Pearl's method of conditioning is one of the known inference methods for Bayesian networks. A short overview of the method of conditioning and definitions of Bayesian networks are needed. The reader is referred to [20,25] for more details.

Let $P(u_1,\ldots,u_n)$ be a probability distribution where each $u_i$ draws values from a finite set called the *domain* of $u_i$. A directed graph $D$ with no directed cycles is called a *Bayesian network of P* if there is a 1-1 mapping between $\{u_1,\ldots,u_n\}$ and vertices in $D$, such that $u_i$ is associated with vertex $i$ and $P$ can be written as follows:

$$P(u_1,\ldots,u_n) = \prod_{i=1}^{n} P(u_i \mid u_{i_1},\ldots,u_{i_{j(i)}}), \qquad (1)$$

where $i_1,\ldots,i_{j(i)}$ are the source vertices of the incoming edges to vertex $i$ in $D$.

Suppose now that some variables $\{v_1,\ldots,v_l\}$ among $\{u_1,\ldots,u_n\}$ are assigned specific values $\{\mathbf{v}_1,\ldots,\mathbf{v}_l\}$ respectively. The *updating problem* is to compute the probability $P(u_i \mid v_1=\mathbf{v}_1,\ldots,v_l=\mathbf{v}_l)$ for $i=1,\ldots,n$.

The concept of d-separation, defined below, is crucial for finding solutions to the updating problem. A *trail* in a Bayesian network is a subgraph whose underlying graph is a simple path. A vertex $b$ is called a *sink* with respect to a trail $t$ if there exist two consecutive edges $a \rightarrow b$ and $b \leftarrow c$ on $t$. A trail $t$ is *active by a set of vertices* $Z$ if (1) every sink with respect to $t$ either is in $Z$ or has a descendant in $Z$ and (2) every other vertex along $t$ is outside $Z$. Otherwise, the trail is said to be *blocked* (*d-separated*) by $Z$.

Verma and Pearl [26] proved that if $D$ is a Bayesian network of $P(u_1,\ldots,u_n)$ and all trails between a vertex in $\{r_1,\ldots,r_l\}$ and a vertex in $\{s_1,\ldots,s_k\}$ are blocked by

[3] The WVC problem is to find a set of vertices that contains an endpoint of every edge in a given undirected graph and which has a minimum weight among all such sets.

$\{t_1, \ldots, t_m\}$, then the corresponding sets of variables $\{u_{r_1}, \ldots, u_{r_l}\}$ and $\{u_{s_1}, \ldots, u_{s_k}\}$ are independent conditioned on $\{u_{t_1}, \ldots, u_{t_m}\}$. Furthermore, Geiger and Pearl [10] proved a converse to this theorem. Both results are presented and extended in [11].

Using the close relationship between blocked trails and conditional independence, Kim and Pearl [16] developed an algorithm UPDATE-TREE that solves the updating problem on Bayesian networks in which every two vertices are connected with at most one trail. These networks are called *singly-connected*. Pearl then solved the updating problem on multiply-connected Bayesian networks by selecting a set of vertices called a loop cutset such that once the corresponding variables are instantiated the remaining network is singly-connected [19]. More precisely, the algorithm can be described as follows:

First, a set of vertices $S$ is selected such that any two vertices in the network are connected with at most one *active* trail by $S \cup Z$, where $Z$ is any subset of vertices. Then, UPDATE-TREE is applied once for each combination of value assignments to the variables corresponding to $S$, and, finally, the results are combined. This algorithm is called the method of *conditioning* and its complexity grows exponentially with the size of $S$. The set $S$ is called a *loop cutset*. Note that when the domain size of the variables varies, then UPDATE-TREE is called a number of times equal to the product of the domain sizes of the variables whose corresponding vertices participate in the loop cutset. If we take the logarithm of the domain size (number of values) as the weight of a vertex, then finding a loop cutset such that the sum of its vertices weights is minimum optimizes Pearl's updating algorithm in the case where the domain sizes may vary.

We now give an alternative definition for a loop cutset $S$ and then provide an approximation algorithm for finding it. This definition is borrowed from [1]. The *underlying graph* $G$ of a directed graph $D$ is the undirected graph formed by ignoring the directions of the edges in $D$. A *cycle* in $G$ is a path whose two terminal vertices coincide. A *loop* in $D$ is a subgraph of $D$ whose underlying graph is a cycle. A vertex $v$ is a *sink* with respect to a loop $\Gamma$ if the two edges adjacent to $v$ in $\Gamma$ are directed into $v$. Every loop must contain at least one vertex that is not a sink with respect to that loop. Each vertex that is not a sink with respect to a loop $\Gamma$ is called an *allowed vertex with respect to* $\Gamma$. A *loop cutset* of a directed graph $D$ is a set of vertices that contains at least one allowed vertex with respect to each loop in $D$. The weight of a set of vertices $X$ is denoted by $w(X)$ and is equal to $\sum_{v \in X} w(v)$ where $w(x) = \log(|x|)$ and $|x|$ is the size of the domain associated with vertex $x$. A *minimum loop cutset* of a weighted directed graph $D$ is a loop cutset $F^*$ of $D$ for which $w(F^*)$ is minimum over all loop cutsets of $G$. The *loop cutset problem* is defined as finding a minimum loop cutset of a given weighted directed graph $D$.

The approach we take is to reduce the weighted loop cutset problem to the weighted vertex feedback set problem, as done by [1]. We now define the weighted vertex feedback set problem and then the reduction.

Let $G = (V, E)$ be an undirected graph, and let $w : V \to \mathbb{R}^+$ be a weight function on the vertices of $G$. A *vertex feedback set* of $G$ is a subset of vertices $F \subseteq V$ such that each cycle in $G$ passes through at least one vertex in $F$. In other words, a vertex feedback set $F$ is a set of vertices of $G$ such that by removing $F$ from $G$, along with all the edges incident with $F$, we obtain a set of trees (i.e., a forest). The weight of a set of vertices $X$ is denoted (as before) by $w(X)$ and is equal to $\sum_{v \in X} w(v)$. A *minimum vertex feedback set* of a weighted graph $G$ with a weight function $w$ is a vertex feedback set $F^*$ of $G$ for which $w(F^*)$ is minimum over all vertex feedback sets of $G$. The *weighted vertex feedback set problem* (WVFS) is defined as finding a minimum vertex feedback set of a given weighted graph $G$ having a weight function $w$.

In the next section we offer an algorithm, called MGA, for approximately solving the weighted vertex feedback set problem. The algorithm is guaranteed to output a weighted vertex feedback set whose weight is less than twice the minimum weight.

The reduction is as follows. Given a weighted directed graph $(D, w)$ (e.g., a Bayesian network), we define the *splitting* weighted undirected graph $D_s$ with a weight function $w_s$ as from [1]. Split each vertex $v$ in $D$ into two vertices $v_{\mathrm{in}}$ and $v_{\mathrm{out}}$ in $D_s$ such that all incoming edges to $v$ in $D$ become undirected incident edges with $v_{\mathrm{in}}$ in $D_s$, and all outgoing edges from $v$ in $D$ become undirected incident edges with $v_{\mathrm{out}}$ in $D_s$. In addition, connect $v_{\mathrm{in}}$ and $v_{\mathrm{out}}$ in $D_s$ by an undirected edge. Now set $w_s(v_{\mathrm{in}}) = \infty$ and $w_s(v_{\mathrm{out}}) = w(v)$. For a set of vertices $X$ in $D_s$, we define $\psi(X)$ as the set obtained by replacing each vertex $v_{\mathrm{in}}$ or $v_{\mathrm{out}}$ in $X$ by the respective vertex $v$ in $D$ from which these vertices originated.

Our algorithm can now be easily stated.

**Algorithm LC.**

```
Input: A Bayesian network $D$.
Output: A loop cutset of $D$.
(1) Construct the splitting graph $D_s$ with weight function $w_s$;
(2) Apply MGA on $(D_s, w_s)$ to obtain a vertex feedback set $F$;
(3) Output $\psi(F)$.
```

It is immediately seen that if MGA outputs a vertex feedback set $F$ whose weight is no more than twice the weight of a minimum vertex feedback set of $D_s$, then $\psi(F)$ is a loop cutset of $D$ with weight no more than twice the weight of a minimum loop cutset of $D$. This observation holds because there is an obvious one-to-one and onto correspondence between loops in $D$ and cycles in $D_s$ and because MGA never chooses a vertex that has an infinite weight.

## 3. Algorithms for the WVFS problem

Recall that the weighted vertex feedback set problem is defined as finding a minimum vertex feedback set of a given weighted graph $G$.

### 3.1. The greedy algorithm

We first analyze the simplest of all approximation algorithms for the weighted vertex feedback set problem—the greedy algorithm. This algorithm is especially interesting because of its simplicity. Assume we are given a weighted undirected graph $G$ with a weight function $w$. The algorithm starts with $G$ after removing all vertices with degree

0 or 1 and repeatedly chooses to insert a vertex $v$ into the constructed vertex feedback set if the ratio between $v$'s weight $w(v)$ and $v$'s degree $d(v)$ in the current graph is minimal across all vertices in the current graph. When $v$ is selected, it is removed from the current graph and then all vertices with degree 0 or 1 are repeatedly removed as well. This step is repeated until the graph is exhausted.

This algorithm and parts of its analysis are influenced by the work of Chvatal (1979) who analyzed the greedy algorithm for the weighted set cover problem (WSC) and by Lovász (1975) and Johnson (1974) who analyzed the unweighted version of this problem.

**Algorithn GA.**

```
Input: A weighted undirected graph $G(V,E,w)$.
Output: A vertex feedback set $F$.
$F \leftarrow \emptyset$; $i \leftarrow 1$;
Repeatedly remove all vertices with degree 0 or 1 from $V$ and their adjacent edges
from $E$ and insert the resulting graph into $G_i$.
While $G_i$ is not the empty graph do
    (1) Pick a vertex $v_i$ for which $w(v_i)/d(v_i)$ is minimum in $G_i$;
    (2) $F \leftarrow F \cup \{v_i\}$;
    (3) $V \leftarrow V \setminus \{v_i\}$;
    (4) $i \leftarrow i+1$;
    (5) Repeatedly remove all vertices with degree 0 or 1 from $V$ and their adjacent
        edges from $E$ and insert the resulting graph into $G_i$.
end
```

In the rest of this section we prove that the performance ratio of GA is bounded by $2\log d + 1$ where $d = \max_{v\in V} d(v)$ is the degree of the graph. Recall that the performance ratio of an approximation algorithm is the worst-case ratio between the weight of the algorithm's output and the weight of an optimal solution. In Section 4, we show experimentally that a slight variant of this algorithm when combined with the reduction algorithm LC convincingly outperforms the algorithms given by [23,24].

Note that the vertices in $F$ (the output of GA) are denoted by $\{v_1, v_2, \ldots, v_t\}$ where the $v_i$ are indexed in the order in which they are inserted into $F$ by GA and where $t = |F|$. Let $d_i(v)$ denote the degree of vertex $v$ in $G_i$—the graph generated in iteration $i$ of GA—and let $V_i$ be the set of vertices of $G_i$. An edge is *covered* by the algorithm if for some $i = 1, \ldots, t$, one of its endpoints is $v_i$ and the edge exists in $G_i$. Note that the set of vertex feedback sets of $G$ and $G_1$ is the same and that the degree of every vertex in $G_1$ is smaller or equal to the degree of that vertex in $G$.

Let $c_i = w(v_i)/d_i(v_i)$ and let $C(e) = c_i$ be the *edge weight* of an edge $e$ removed at iteration $i$. Note that for every $j \leqslant i$ we have $w(v_j)/d_j(v_j) \leqslant w(v_i)/d_j(v_i)$ because vertices are selected in increasing order of these ratios. Also note that for $j \leqslant i$, $d_j(v_i) \geqslant d_i(v_i)$ since the algorithm never adds edges. Thus,

$$c_j \equiv w(v_j)/d_j(v_j) \leqslant w(v_i)/d_i(v_i) \equiv c_i, \tag{2}$$

for $1 \leqslant j \leqslant i \leqslant |F|$, as originally claimed by [3] in the context of the WSC problem.

To analyze the performance ratio we use a lemma that bounds the number of edges in $G_i$ covered by the algorithm until its termination. We need the following definitions. Let $d_X(v)$ be the number of edges whose one endpoint is $v$ and the other is a vertex in $X$. A *linkpoint* is a vertex that has degree 2 and a *branchpoint* is a vertex that has a degree larger than 2. (A self-loop adds 2 to the degree of a vertex.)

**Lemma 1.** *Let $F = \{v_1, \ldots, v_t\}$ be a vertex feedback set produced by GA for a graph $G$, $F_i = \{v_i, \ldots, v_t\}$, and $F^*$ be any vertex feedback set of $G$. Also let $F_i^* = F^* \cap V_i$ where $V_i$ is the set of vertices of $G_i$—the graph produced in iteration $i$ of GA. Then,*

$$\sum_{j=i}^{t} d_j(v_j) \leqslant 2 \sum_{v \in F_i^*} d_i(v). \tag{3}$$

**Proof.** Let $\overline{F}_i^* = \overline{F}^* \cap V_i$. We will prove two inequalities. First,

$$\sum_{v \in V_i} (d_i(v) - 2) + 2|F_i^*| \leqslant 2 \sum_{v \in F_i^*} d_i(v) \tag{4}$$

and then

$$\sum_{j=i}^{t} d_j(v_j) \leqslant \sum_{v \in V_i} (d_i(v) - 2) + 2|F_i^*|. \tag{5}$$

According to our notations,

$$\sum_{v \in V_i} (d_i(v) - 2) = \sum_{v \in \overline{F}_i^*} (d_{\overline{F}_i^*}(v) - 2) + \sum_{v \in \overline{F}_i^*} d_{F_i^*}(v) + \sum_{v \in F_i^*} (d_i(v) - 2).$$

Furthermore, the graph induced by $\overline{F}_i^*$ is a forest and since the number of edges in a forest is smaller than the number of vertices, we have $\sum_{v \in \overline{F}_i^*} d_{\overline{F}_i^*}(v)/2 \leqslant |\overline{F}_i^*|$. Thus $\sum_{v \in \overline{F}_i^*} (d_{\overline{F}_i^*}(v) - 2) \leqslant 0$. Consequently,

$$\sum_{v \in V_i} (d_i(v) - 2) + 2|F_i^*| \leqslant \sum_{v \in \overline{F}_i^*} d_{F_i^*}(v) + \sum_{v \in F_i^*} d_i(v) \leqslant 2 \sum_{v \in F_i^*} d_i(v).$$

The proof of Eq. (5) is constructive. We repeatedly apply the following procedure on $G_i$ selecting in each step a vertex $v_j \in F_i$ and showing that there are terms in the right-hand side (RHS) of Eq. (5) that can contribute $d_j(v_j)$ to the RHS and have not been used for any other $v \in F_i$. Set $H = G_i$ and for $k = i, \ldots, t$ do as follows:

Pick the vertex $v_k$. If $v_k$ is a linkpoint in $H$ then follow the two paths $p_1$ and $p_2$ in $H$ emanating from $v_k$ until the first branchpoint on each side is found. There are three cases to consider. Either two distinct branchpoints $b_1$ and $b_2$ are found, one branchpoint $b_1$ (in which case $p_1$ and $p_2$ define a cycle) or none (if the cycle is isolated). In the first case the two edges on $p_1$ and $p_2$ whose endpoints are $b_1$ and $b_2$, respectively, are associated with the terms $d_k(b_1) - 2 > 0$ and $d_k(b_2) - 2 > 0$ in the RHS and so, since for every vertex $v$ in $G_k$, $d_k(v) \leqslant d_i(v)$, each of these terms can contribute 1 to the sum $\sum_{v\in V_i}(d_i(v)-2)$. In the second case, similarly, the two edges on $p_1$ and $p_2$ whose endpoints is $b_1$ are associated with the term $d_k(b_1)-2>0$ and so, if $d_k(b_1)>3$, this term can contribute 2 to the sum $\sum_{v\in V_i}(d_i(v)-2)$. If $d_k(b_1)=3$ we continue to follow the third path from $b_1$ (i.e., not $p_1$ or $p_2$) until another branchpoint $b_2$ is found and the last edge on that path is associated with $d_k(b_2)-2$ which can contribute the extra missing 1 to the RHS. Finally, if no branchpoint is found, then on the cycle in which $v_k$ resides there must exist a vertex from $F_i^*$ that resides on no other cycles of $H$ (hence the term $2|F_i^*|$ in the RHS). Now, if $v_k$ is a branchpoint, then the term $d_k(v_k)-2$ appears in both sides of the inequality. In this case, sequentially remove $d_k(v_k)-2$ of the $d_k(v_k)$ edges adjacent to $v_k$ such that after each removal the vertices with degree 0 or 1 are removed from $H$ as well. Thus, $v_k$ remains a linkpoint in which case the procedure for a linkpoint is applied. Finally, remove $v_k$, and repeatedly remove all the vertices with degree 0 or 1 from $H$. Repeat until $F_i$ is exhausted. □

Note that the above proof does not use the weight function of $G$ and thus the stated inequality holds even when in step (1) of GA's main loop the vertex $v_i$ is selected arbitrarily regardless of its current degree or weight. We will use this observation in the next section.

Let $F^*$ be a minimum weight feedback set of $G(V,E,w)$ and let $\overline{F}^*=V\setminus F^*$. Denote $F_i^*=F^*\cap V_i$ and $\overline{F}_i^*=\overline{F}^*\cap V_i$. Lemma 1 holds for every vertex feedback set $F^*$ and therefore, in particular, when $F^*$ has minimum weight. We now show that $w(F)\leqslant(2\log d+1)\cdot w(F^*)$.

$$\begin{aligned} w(F) &= \sum_{i=1}^{t} w(v_i) = \sum_{i=1}^{t} c_i\cdot d_i(v_i) \\ &= c_1\sum_{i=1}^{t} d_i(v_i) + \sum_{i=2}^{t}(c_i-c_{i-1})\sum_{j=i}^{t} d_j(v_j). \end{aligned} \tag{6}$$

Since $c_i\geqslant c_{i-1}$ and due to Eq. (3), we get

$$\begin{aligned} w(F) &\leqslant 2c_1\sum_{v\in F_1^*} d_1(v) + \sum_{i=2}^{t} 2(c_i-c_{i-1})\sum_{v\in F_i^*} d_i(v) \\ &= \sum_{i=1}^{t} 2c_i\sum_{v\in F_i^*} d_i(v) - \sum_{i=1}^{t-1} 2c_i\sum_{v\in F_{i+1}^*} d_{i+1}(v). \end{aligned} \tag{7}$$

Thus,

$$\begin{aligned} w(F) &\leqslant \sum_{i=1}^{t} 2c_i\sum_{v\in F_i^*\setminus F_{i+1}^*} d_i(v) + \sum_{i=1}^{t} 2c_i\sum_{v\in F_{i+1}^*} d_i(v) - \sum_{i=1}^{t-1} 2c_i\sum_{v\in F_{i+1}^*} d_{i+1}(v) \\ &= 2\left[\sum_{i=1}^{t-1}\left(c_i\sum_{v\in F_i^*\setminus F_{i+1}^*} d_i(v) + c_i\sum_{v\in F_{i+1}^*}(d_i(v)-d_{i+1}(v))\right) + c_t\sum_{v\in F_t^*} d_t(v)\right]. \end{aligned}$$

However, since the last sum on the right-hand side merely counts the edge weights according to the iteration they are assigned a weight, we get,

$$w(F) \leqslant 2 \sum_{v \in F^*} \sum_{e \in \Gamma_1(v)} C(e), \tag{8}$$

where $\Gamma_1(v)$ denotes the set of edges in $G_1$ for which at least one endpoint is $v$.

We now show that for every $v \in F^*$,

$$w(v) \left[H(d(v)) - 1/2\right] \geqslant \sum_{e \in \Gamma_1(v)} C(e), \tag{9}$$

where $H(m) = \sum_{i=1}^{m} 1/i$, using the following argument which is similar to the one used in [3].

Let $s$ be the largest superscript such that $d_s(v) > 0$. Consequently,

$$\begin{aligned}\sum_{e \in \Gamma_1(v)} C(e) &= \sum_{i=1}^{s} (d_i(v) - d_{i+1}(v)) \cdot (w(v_i)/d_i(v_i)) \\ &\leqslant w(v) \sum_{i=1}^{s} (d_i(v) - d_{i+1}(v))/d_i(v),\end{aligned}$$

where the inequality is due to Eq. (2). Furthermore, it can be shown by induction that

$$\frac{n_1 - n_2}{n_1} \leqslant H(n_1) - H(n_2),$$

whenever $n_1$ and $n_2$ are positive integers satisfying $n_2 \leqslant n_1$. Thus, and since $d_s(v) \geqslant 2$,

$$\sum_{e \in \Gamma_1(v)} C(e) \leqslant w(v) \left[ \sum_{i=1}^{s-1} [H(d_i(v)) - H(d_{i+1}(v))] + H(d_s(v)) - 1/2 \right].$$

Since the right-hand side is equal to $w(v)[H(d_1(v)) - 1/2]$, Eq. (9) follows. Combining Eqs. (8) and (9) yields,

$$w(F) \leqslant 2 \sum_{v \in F^*} [H(d(v)) - 1/2] \cdot w(v) \leqslant [2H(d) - 1] \cdot w(F^*).$$

Thus,

**Theorem 2.** *The performance ratio of GA is bounded by* $2H(d) - 1$.

Notably, since $H(d) < \log d + 1$ (for $d > 1$), the performance ratio of GA is bounded by $2 \log d + 1$.

We now describe a sequence of graphs for which GA achieves a performance ratio $2H(d) - 2$. Consequently, the upper bound given by Theorem 2 is rather tight.

Let $B(L, R, E')$ be a complete bipartite graph where $R = \{u_1, \dots, u_r\}$ and where $L$ also consists of $r$ vertices. We slightly modify $B(L, R, E')$ as follows to obtain a graph

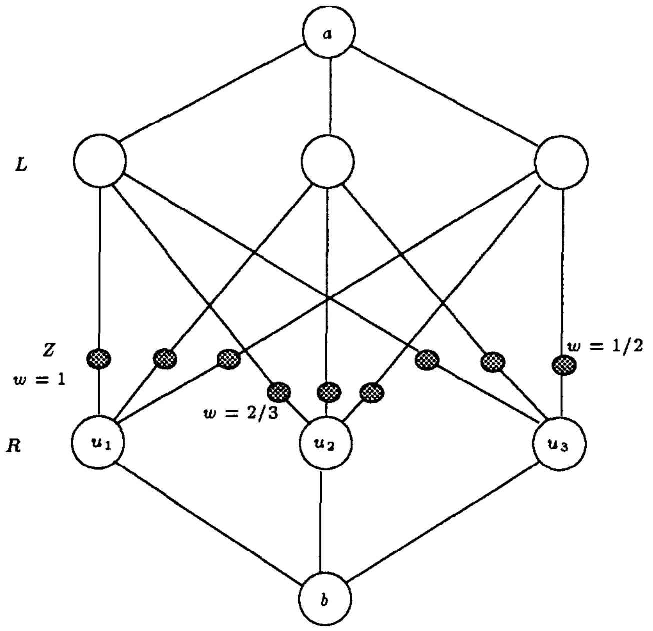

Fig. 1. The graph $G$ for $r = 3$.

$G$. We add a new vertex $a$ which is connected to each vertex in $L$ with an edge and we add a new vertex $b$ which is connected to each vertex in $R$ with an edge. Furthermore, each edge $(l, u_i)$ between a vertex $l$ in $L$ and a vertex $u_i$ in $R$ is replaced with a chain of two edges and a new middle vertex whose weight is $2/(i+1)$. The set of these middle vertices is denoted by $Z$. The weight of every vertex in $L$ is 1 and the weight of all other vertices in $G$ is infinitely large. Fig. 1 shows $G$ for $r = 3$.

Consider the behavior of GA on $G$. Suppose the algorithm chooses the vertices in $Z$ for the vertex feedback set according to increasing order of weight-to-degree ratios. The weight of this vertex feedback set is $2r(H(d) - 1)$ where $d = r + 1$ is the degree of $G$. However, $L$ itself is a minimum vertex feedback set of $G$ having a weight $r$. It follows that the performance ratio achieved by GA for these graphs is $2H(d) - 2$.

It may seem that GA can be improved if in each step the algorithm will remove every vertex that does not reside anymore on a cycle in the current graph rather than remove only those vertices that have degrees 0 or 1. However, even for this slightly modified algorithm, which we shall call GA$'$, the conclusions given by Theorem 2 and the above example still hold, namely, the worst-case performance can hardly be affected by this change. In fact, there are examples in which GA is superior and examples where GA$'$ is superior. However, on the average, GA$'$ performs slightly better than GA.

Finally we note that the output $F$ of GA need not be a minimal vertex feedback set of $G$, that is, there may exist a vertex $v$ in $F$ such that $F \setminus \{v\}$ is still a vertex feedback set of $G$. If we had removed such redundant vertices from $F$, as we will in the next section, this modification would not by itself have improved the (worst-case) performance ratio of GA below $2H(d)-2$ because the output $Z$ of GA for the above example is in fact a minimal vertex feedback set that achieves the stated performance ratio. This discussion applies to GA$'$ as well.

## *3.2. The modified greedy algorithm*

We now present a modified greedy algorithm, called MGA, whose performance ratio is bounded by the constant 2. The changes we introduce into GA are quite minor and so it is interesting that such a vast improvement in the performance ratio is obtained. A similar phenomenon is reported in the context of the weighted vertex cover problem [4].

MGA has two phases. In the first phase MGA repeatedly chooses to insert a vertex $v$ into the constructed vertex feedback set if the ratio between $v$'s weight $w(v)$ and $v$'s degree $d(v)$ in the current graph is minimal across all vertices in the current graph. When $v$ is selected, it is removed from the current graph and then all vertices with degree 0 or 1 are repeatedly removed as well. For every edge removed in this process, a weight of $w(v)/d(v)$ is subtracted from its endpoint vertices. These steps are repeated until the graph is exhausted. The only difference between this phase and the plain greedy algorithm is the revision of some weights in each step instead of just revising the current degrees. The second phase removes redundant vertices from the constructed vertex feedback set.

```
Algorithm MGA.
  Input: A weighted undirected graph $G(V,E,w)$.
  Output: A vertex feedback set $F$.
  $F' \leftarrow \emptyset$; $i \leftarrow 1$;
  Repeatedly remove all vertices with degree 0 or 1 from $V$ and their adjacent edges
  from $E$ and insert the resulting graph into $G_i$.
  While $G_i$ is not the empty graph do
    (1) Pick a vertex $v_i$ for which $w(v_i)/d(v_i)$ is minimum in $G_i$;
    (2) $F' \leftarrow F' \cup \{v_i\}$;
    (3) $V \leftarrow V \setminus \{v_i\}$;
    (4) $C = w(v_i)/d(v_i)$;
    (5) $i \leftarrow i+1$;
    (6) Repeatedly remove all vertices with degree 0 or 1 from $V$ and their adjacent
        edges from $E$ and insert the resulting graph into $G_i$.
        For every edge $e=(u_1,u_2)$ removed in this process do
          $C(e) \leftarrow C$
          $w(u_1) \leftarrow w(u_1) - C(e)$
          $w(u_2) \leftarrow w(u_2) - C(e)$.
  end
  $F \leftarrow F'$
  For $i = |F'|$ to 1 do {Phase 2}
    If every cycle in $G$ that intersects with $\{v_i\}$ also intersects with $F \setminus \{v_i\}$ then,
    $F \leftarrow F \setminus \{v_i\}$.
  endfor
end
```

Clearly $F'$ computed at the first phase of MGA is a vertex feedback set of $G$ and $F$ created from $F'$ by removing all redundant vertices is a *minimal vertex feedback set* of $G$, that is, if a vertex is removed from $F$, then $F$ ceases to be a vertex feedback set of $G$. Furthermore, as a result of removing redundant vertices the inequality given by Eq. (3), proven to hold for GA, becomes

$$\sum_{v\in F_i} d_i(v) \leqslant 2 \sum_{v\in F_i^*} d_i(v), \tag{10}$$

where $F_i = F\cap V_i$, $F_i^* = F^*\cap V_i$ and $V_i$ are the vertices in $G_i$. The proof of this inequality is postponed to Section 3.3. From the description of the algorithm we have for every vertex $v$ in $G_1$,

$$\sum_{e\in\Gamma_1(v)} C(e) \leqslant w(v) \tag{11}$$

and if $v \in F$, equality must hold. By analogy with the previous section, Eqs. (10) and (11), which replace Eqs. (3) and (9), suggest that the bound on the performance ratio drops from $2\log d + 1$ for GA to 2 for MGA, as shown next.

**Theorem 3.** *Algorithm MGA always outputs a vertex feedback set whose weight is no more than twice the weight of a minimum vertex feedback set.*

**Proof.** As in Section 3.1, $F^*$ denotes a minimum vertex feedback set of $G(V, E, w)$ and $\overline{F}^* = V\setminus F^*$. Recall that the vertices in the constructed set $F'$ are $\{v_1, v_2, \ldots, v_t\}$ where $v_i$ are indexed in the order in which they are inserted into $F'$ by MGA and $t = |F'|$. Also, $w_i(v)$ and $d_i(v)$ denote the weight and degree, respectively, of vertex $v$ in $G_i$—the graph generated in iteration $i$ of step (5) of MGA—and $V_i$ denotes the set of vertices of $G_i$.

As in the previous subsection for every $j \leqslant i$ we have $w_j(v_j)/d_j(v_j) \leqslant w_j(v_i)/d_j(v_i)$ and also $w_j(v_i)/d_j(v_i) \leqslant w_i(v_i)/d_i(v_i)$ due to the way that the current weights and degrees are updated in the algorithm. Thus,

$$c_j \equiv w_j(v_j)/d_j(v_j) \leqslant w_i(v_i)/d_i(v_i) \equiv c_i, \tag{12}$$

for $1 \leqslant j \leqslant i \leqslant |F'|$.

We also have

$$\sum_{e\in\Gamma_1(v_i)} C(e) = c_i\cdot d_i(v_i) + \sum_{j=1}^{i-1} c_j\cdot (d_j(v_i) - d_{j+1}(v_i)), \tag{13}$$

because the right-hand side simply groups edges according to the iteration in which they are assigned a weight.

Let $\alpha_i = 1$ if $v_i \in F$ and $\alpha_i = 0$ if $v_i \notin F$. That is, $\alpha_i$ is 1 if $v_i$ is not removed from $F'$ in the second phase of MGA and 0 otherwise. We now prove that $w(F) \leqslant 2\cdot w(F^*)$.

$$w(F) = \sum_{i=1}^{t} \alpha_i\cdot w(v_i) = \sum_{i=1}^{t} \alpha_i \sum_{e\in\Gamma_1(v_i)} C(e).$$

Now, due to Eq. (13),

$$w(F) = \sum_{i=1}^{t} \alpha_i \cdot \left[ c_i \cdot d_i(v_i) + \sum_{j=1}^{i-1} c_j \cdot (d_j(v_i) - d_{j+1}(v_i)) \right].$$

Hence,

$$w(F) = c_1 \sum_{i=1}^{t} \alpha_i \cdot d_1(v_i) + \sum_{i=2}^{t} (c_i - c_{i-1}) \sum_{j=i}^{t} \alpha_j \cdot d_i(v_j).$$

Furthermore,

$$\sum_{j=i}^{t} \alpha_j \cdot d_i(v_j) = \sum_{v \in F_i} d_i(v). \tag{14}$$

Now, since $c_i \geqslant c_{i-1}$ and due to Eqs. (10) and (14), we get,

$$w(F) \leqslant 2c_1 \sum_{v \in F_1^*} d_1(v) + \sum_{i=2}^{t} 2(c_i - c_{i-1}) \sum_{v \in F_i^*} d_i(v).$$

This equation is identical to Eq. (7). Consequently, as in the derivation of Eq. (8), we get,

$$w(F) \leqslant 2 \sum_{v \in F^*} \sum_{e \in \Gamma_1(v)} C(e). \tag{15}$$

Now, Eqs. (11) and (15) yield the claimed inequality, $w(F) \leqslant 2\sum_{v \in F^*} w(v) = 2w(F^*)$. □

Interestingly, if the second phase is removed from MGA (making MGA even closer to GA), then the performance ratio becomes 4 rather than 2. To prove this claim we use the inequality

$$\sum_{j=i}^{t} d_j(v_j) \leqslant 2 \sum_{v \in F_i^*} d_i(v),$$

which holds for the first phase of MGA due to the proof of Lemma 1 because the only difference between this phase of MGA and GA is the way the weights are altered, a fact not used in the proof of this inequality. The other inequality is

$$\sum_{j=i}^{t} (d_i(v_j) - d_j(v_j)) \leqslant \sum_{j=i}^{t} d_j(v_j),$$

which holds because the number of edges adjacent to $F'$ which are removed but not covered by the first phase of MGA is smaller than the total number of edges covered by the algorithm. Consequently, we have

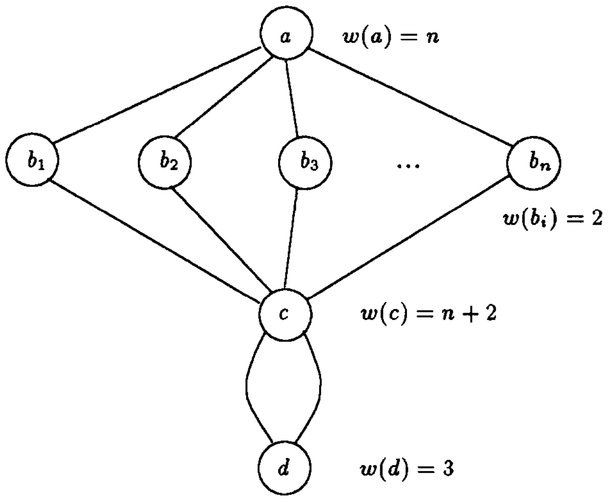

Fig. 2. The graph $H$.

$$\sum_{j=i}^{t} d_i(v_j) \leqslant 4 \sum_{v \in F_i^*} d_i(v). \tag{16}$$

Now, by using Eq. (16) instead of Eq. (10) in the proof of Theorem 3, the bound on the performance ratio is shown to be 4.

We now show that 4 is a tight bound for the MGA algorithm without the second phase. Consider the graph $H$ given in Fig. 2. Suppose that the algorithm first selects vertices $B_1, B_2, \ldots, B_{n-2}$, then it selects $A$, and finally it selects $C$. The weight of this vertex feedback set is $4n - 2$. However $C$ itself is a vertex feedback set. Thus, the performance ratio achieved by this algorithm is $(4n-2)/(n+2)$ and when $n \to \infty$ the ratio approaches 4.

Consequently, the vast improvement in the worst-case performance of MGA compared to GA stems from changing the vertices' weights in each step rather than from removing redundant vertices.

The complexity of the first phase of MGA is $O(|E| + |V| \log |V|)$ using a Fibonacci heap (e.g., [8]) because finding and deleting a vertex with minimum ratio $w(v)/d(v)$ from the heap is done $|V|$ times at the cost of $O(\log |V|)$ and decreasing a weight from a vertex in the heap is done $|E|$ times at an amortized cost of $O(1)$.

A naive implementation of the second phase can be described as follows. For $i = |F'|$ to 1 do: remove $F \setminus \{v_i\}$ and its adjacent edges from $G$, if the resulting graph contains no cycle, then $v_i$ is removed from $F$ (because every cycle in $G$ that intersects with $\{v_i\}$ also intersects with $F \setminus \{v_i\}$). The complexity of this implementation is $O(|E||V|)$. To implement the second phase more efficiently we observe the following two propositions:

**Proposition 4.** *If every cycle in $G_i$ that intersects with $\{v_i\}$ also intersects with $F \setminus \{v_i\}$, then every cycle in $G$ that intersects with $\{v_i\}$ also intersects with $F \setminus \{v_i\}$.*

Proposition 4 holds because every cycle in $G$ that is not a cycle in $G_i$ intersects with $F \setminus \{v_i\}$. This proposition implies that in order to decide whether to remove $v_i$ from $F$ it suffices to check all cycles in $G_i$ rather than all cycles in $G$.

**Proposition 5.** *Let $B$ be any minimal vertex feedback set of the graph $G_{i+1}$. If $B$ is a vertex feedback set of $G_i$, then $B$ is a minimal vertex feedback set of $G_i$. If $B$ is not a vertex feedback set of $G_i$, then $B \cup \{v_i\}$ is a minimal vertex feedback set of $G_i$.*

Proposition 5 holds because the only cycles contained in $G_i$ which are not contained in $G_{i+1}$ are cycles that pass through $v_i$. So, $B \cup \{v_i\}$ is a vertex feedback set of $G_i$. Thus, if $B$ is a vertex feedback set of $G_i$, then $B$ is minimal lest $B$ were not minimal in $G_{i+1}$. And if $B$ is not a vertex feedback set, then $B \cup \{v_i\}$ is minimal because no vertex in $B$ can be eliminated lest $B$ were not minimal in $G_{i+1}$.

Note that a cycle in $G_i$ intersects with $F' \setminus \{v_i\}$ iff it intersects with $F_i' \setminus \{v_i\}$ (where $F'$ is the vertex feedback set found in the first phase of MGA). The algorithm thus starts with $F_t = F' \cap V_t$ which is a minimal vertex feedback set of $G_t$ and continues backwards with $F_i$, $i = t, \ldots, 1$. In order to find a minimal vertex feedback set $F_i \subseteq F' \cap V_i$ of $G_i$, assuming we already found such minimal vertex feedback sets $F_k$ of $G_k$, $k = i+1, \ldots, t$, using Proposition 5, it remains to determine whether $F_{i+1}$ is a vertex feedback set of $G_i$. If it is, then it is minimal (so $v_i$ is removed from $F'$), and if it is not then $F_i = F_{i+1} \cup \{v_i\}$ is a minimal vertex feedback set. To facilitate the test, we construct a graph $H_i$ induced from $G_i$ by $V_i \setminus (F_{i+1} \cup \{v_i\})$. This graph is a forest and $v_i$ resides on some cycle in $G_i$ on which no vertex from $F_{i+1}$ resides if and only if $v_i$ has two neighbors in the same connected component of $H_i$.

We construct the graphs $H_i$, $i = t, \ldots, 1$, efficiently as follows. The graph $H_t$ consists of the vertices $V_t \setminus \{v_t\}$ and the edges adjacent to these vertices. To construct $H_i$ when we have already constructed $H_{i+1}$ and the minimal vertex feedback set $F_{i+1}$, we sequentially add to $H_{i+1}$ the set of vertices $V_i \setminus (V_{i+1} \cup \{v_i\})$ and their adjacent edges. Now, if $v_i$ does not have two neighbors in the same tree of $H_i$, then we add $v_i$ to $H_i$. (If $v_i$ has two neighbors in the same tree, then $v_i \in F_i$ and therefore, by our definition of $H_i$, $v_i$ is not included in $H_i$ nor in $H_{i-1}, \ldots, H_1$.) We use a simple version of the union-find algorithm. First we call $|V|$ times the MAKE_SET operation creating a collection $\mathcal{C}$ of singleton sets—one for each vertex $v$ in $V$. When we add a vertex $v$ and its adjacent edges to $H_i$ we unify the sets $\{v\}$ with the set $\bigcup_{u \in \Gamma_i(v)} \mathrm{FIND}(u)$ where $\Gamma_i(v)$ are the neighbors of $v$ in $G_i$ and $\mathrm{FIND}(u)$ is the set in $\mathcal{C}$ that contains $u$. Consequently, $\mathcal{C}$ maintains the connected components of $H_i$. Finally, in order to check if $v_i$ has two neighbors in the same tree of $H_i$ we check if any of the two neighbors of $v_i$ are in the same set in $\mathcal{C}$. We need to do at most $|V|$ union operations at an amortized cost of $O(\log |V|)$. When we check whether $v_i$ resides on some cycle we need $O(|E|)$ find operations at the cost of $O(1)$ [5, p. 445]. Therefore, the complexity of the second phase of MGA is also just $O(|E| + |V| \log |V|)$.

### 3.3. A theorem about minimal vertex feedback sets

In this section we prove a theorem that relates the number of edges adjacent to any minimal weighted vertex feedback set to the number of edges adjacent to any minimum weighted vertex feedback set. This theorem proves Eq. (10) which has been used in the analysis of the modified greedy algorithm.

Let $G$ be a weighted graph for which every vertex has a degree strictly greater than 1, $F$ be a minimal vertex feedback set of $G$ and $F^*$ be an arbitrary vertex feedback set of $G$ (possibly a minimum weight vertex feedback set). Let $d(v)$ be the degree of vertex $v$ and $d_X(v)$ be the number of edges whose one endpoint is $v$ and the other is in a set of vertices $X$.

**Theorem 6.** *Let $G$, $F$ and $F^*$ be defined as above. Then,*

$$\sum_{v\in F} d(v) \leqslant 2 \sum_{v\in F^*} d(v).$$

Note that $F_i$ in Eq. (10) is a minimal vertex feedback set of $G_i$ and therefore Theorem 6 proves Eq. (10). Also note that this theorem does not imply nor is implied by Lemma 1 albeit their similarities.

To prove this theorem we divide $\sum_{v\in F} d(v)$ into the sum $2|F| + \sum_{v\in F}(d(v) - 2)$ and provide an upper bound for each term.

**Lemma 7.** *Let $G$, $F$ and $F^*$ be defined as above. Then,*

$$2|F| \leqslant \sum_{v\in \overline{F}} d(v) - 2|\overline{F} \cap \overline{F}^*| + 2|F \cap F^*|. \tag{17}$$

**Proof.** First note that for every set of vertices $B$ in $G$,

$$\begin{aligned}\sum_{v\in \overline{F}} d(v) - 2|\overline{F} \cap \overline{F}^*| = \sum_{v\in \overline{F}\cap B} d(v) + \sum_{v\in \overline{F}\setminus B} d(v) - 2|\overline{F} \cap \overline{F}^* \cap B| \\ -2|(\overline{F} \cap \overline{F}^*) \setminus B|. \end{aligned}\tag{18}$$

However, the degree of every vertex in $G$ satisfies $d(v) \geqslant 2$ and therefore $\sum_{v\in \overline{F}\setminus B} d(v) \geqslant 2|(\overline{F} \cap \overline{F}^*) \setminus B|$. Consequently,

$$\sum_{v\in \overline{F}} d(v) - 2|\overline{F} \cap \overline{F}^*| \geqslant \sum_{v\in \overline{F}\cap B} d(v) - 2|\overline{F} \cap \overline{F}^* \cap B|. \tag{19}$$

Thus, and since $|F \cap F^*| \geqslant |F \cap F^* \cap B|$ and $d_B(v) \leqslant d(v)$, to prove the lemma it suffices to show that

$$2|F| \leqslant \sum_{v\in \overline{F}\cap B} d_B(v) - 2|\overline{F} \cap \overline{F}^* \cap B| + 2|F \cap F^* \cap B|, \tag{20}$$

or equivalently,

$$2|F| \leqslant \sum_{v\in \overline{F}\cap B} (d_B(v) - 2) + 2|F^* \cap B| \tag{21}$$

holds for some set of vertices $B$. We now define a set $B$ for which this inequality can be proven. Since $F$ is minimal, each vertex in $F$ can be associated with a cycle in $G$ that contains no other vertices of $F$. We define a graph $H$ that consists of the union of these cycles—one cycle per each vertex. Note that every vertex in $F$ is a linkpoint in $H$, i.e., a vertex with degree 2. Let $B$ be the set of vertices of $H$.

The proof of Eq. (21) is constructive. We repeatedly apply the following procedure on $H$ selecting in each step a vertex $v \in F$ and showing that there are terms in the right-hand side (RHS) of Eq. (21) that can contribute 2 to the RHS and have not been used for any other $v \in F$.

Pick a vertex $v \in F$ and follow the two paths $p_1$ and $p_2$ in $H$ emanating from $v$ (which is a linkpoint) until the first branchpoint on each side is found. There are three cases to consider. Either two distinct branchpoints $b_1$ and $b_2$ are found, one branchpoint $b_1$ (in which case $p_1$ and $p_2$ define a cycle) or none (if the cycle is isolated). In the first case the two edges on $p_1$ and $p_2$ whose endpoints are $b_1 \in \overline{F}$ and $b_2 \in \overline{F}$, respectively, are associated with the terms $d_B(b_1) - 2 > 0$ and $d_B(b_2) - 2 > 0$ in the RHS and so each of these terms can contribute 1 to the sum $\sum_{v\in\overline{F}\cap B}(d_B(v) - 2)$. In the second case, similarly, the two edges on $p_1$ and $p_2$ whose endpoint is $b_1 \in \overline{F}$ are associated with the term $d_B(b_1) - 2 > 0$ and so, if $d_B(b_1) > 3$, this term can contribute 2 to the sum $\sum_{v\in\overline{F}\cap B}(d_B(v) - 2)$. If $d_B(b_1) = 3$ we continue to follow the third path from $b_1$ (i.e., not $p_1$ or $p_2$) until another branchpoint $b_2 \in \overline{F}$ is found and the last edge on that path is associated with $d_B(b_2) - 2$ which can contribute the extra missing 1 to the RHS. Finally, if no branchpoint is found, then on the isolated cycle on which $v$ resides there exists a vertex from $F^*$ that resides on no other cycles of $H$. Thus, the third case could not occur more than $|F^* \cap B|$ times. Now remove the paths $p_1$ and $p_2$ from $H$ obtaining a graph in which still each vertex in $F$ resides on a cycle that contains no other vertices of $F$. Continue the process until $F$ is exhausted. □

**Lemma 8.** *Let $G$, $F$ and $F^*$ be defined as above. Then,*

$$\sum_{v\in F}(d(v) - 2) \leqslant \sum_{v\in F\cap\overline{F}^*} d_{F^*}(v) + \sum_{v\in F\cap F^*}(d(v) - 2) - \sum_{v\in\overline{F}\cap\overline{F}^*}(d_{\overline{F}^*}(v) - 2).$$

**Proof.** First note that

$$\sum_{v\in F}(d(v) - 2) = \sum_{v\in F\cap\overline{F}^*} d_{F^*}(v) + \sum_{v\in F\cap F^*}(d(v) - 2) + \sum_{v\in F\cap\overline{F}^*}(d_{\overline{F}^*}(v) - 2). \tag{22}$$

We now claim that $\sum_{v\in F\cap\overline{F}^*}(d_{\overline{F}^*}(v) - 2) + \sum_{v\in\overline{F}\cap\overline{F}^*}(d_{\overline{F}^*}(v) - 2)$ is less or equal than 0 which concludes this proof. The graph induced by $\overline{F}^*$ is a forest and since the number of edges in a forest is smaller than the number of vertices, we have, $\sum_{v\in\overline{F}^*} d_{\overline{F}^*}(v)/2 \leqslant |\overline{F}^*|$. Thus $\sum_{v\in\overline{F}^*}(d_{\overline{F}^*}(v) - 2) \leqslant 0$ which is equivalent to the stated claim. □

**Proof of Theorem 6.** Using the bounds given by Lemmas 7 and 8 we have,

$$\sum_{v\in F} d(v) \leqslant \sum_{v\in F\cap F^*} (d(v)-2) + 2|F\cap F^*| - \sum_{v\in \overline{F}\cap \overline{F}^*} (d_{\overline{F}^*}(v)-2)$$
$$-2|\overline{F}\cap \overline{F}^*| + \sum_{v\in F\cap \overline{F}^*} d_{F^*}(v) + \sum_{v\in \overline{F}} d(v).$$

Thus,

$$\sum_{v\in F} d(v) \leqslant \sum_{v\in F\cap F^*} d(v) - \sum_{v\in \overline{F}\cap \overline{F}^*} d_{\overline{F}^*}(v) + \sum_{v\in F\cap \overline{F}^*} d_{F^*}(v) + \sum_{v\in \overline{F}} d(v).$$

Now,

$$\sum_{v\in \overline{F}} d(v) - \sum_{v\in \overline{F}\cap \overline{F}^*} d_{\overline{F}^*}(v) = \sum_{\overline{F}\cap F^*} d(v) + \sum_{v\in \overline{F}\cap \overline{F}^*} d_{F^*}(v)$$

and therefore,

$$\sum_{v\in F} d(v) \leqslant \sum_{v\in \overline{F}^*} d_{F^*}(v) + \sum_{v\in F^*} d(v) \leqslant 2\sum_{v\in F^*} d(v),$$

which concludes the proof of the theorem. □

## 4. Experimental results

Below we denote by A1 the algorithm described in [24] and by A2 the algorithm described in [23]. These algorithms find loop cutsets. We performed six experiments. In the first two experiments we tested how the outputs of the four algorithms, A1, A2, GA′ and MGA, compare to a minimum loop cutset. In two additional experiments we checked how the algorithms' outputs compare to each other when given larger graphs for which a minimum loop cutset is hard to obtain. In these four experiments we have chosen all variables to be binary. The final two experiments compare the performance of these algorithms when the number of values of each vertex is randomly chosen between 2 and 6, 2 and 8, and between 2 and 10. Each instance of the six experiments is based on 100 random graphs generated as described by [24].

In the first experiment each of the 100 graphs generated had 15 vertices and 25 edges. MGA made only one mistake producing 6 vertices instead of the minimum of 5 vertices. GA′ made 4 mistakes each by one vertex off. A2 made 7 mistakes one of which was two vertices off the minimum and the other six mistakes were one vertex off. A1 made 11 mistakes one of which was 2 vertices off and the other 10 mistakes were one vertex off. The minimum loop cutsets were between 3 and 6 vertices. Note that the ratio between the number of instances associated with a loop cutset found by MGA in this experiment and the number of instances associated with a minimum loop cutset is 1.01 which is far less than the theoretical ratios guaranteed by Theorem 3 for this experiment which lie between 8 when the minimum loop cutset contains 3 binary variables and 64 when the minimum loop cutset contains 6 binary variables.

In the second experiment we generated 100 networks each with 25 vertices and 25 edges and tested how the output of the four algorithms compare to a minimum loop

Table 1

| $\|V\|$ | $\|E\|$ | A2 | GA′ | Eq. | GA′ | MGA | Eq. |
|---|---|---|---|---|---|---|---|
| 25 | 25 | 0 | 1 | 99 | 0 | 4 | 96 |
| 25 | 50 | 1 | 8 | 91 | 0 | 8 | 92 |
| 25 | 75 | 0 | 15 | 85 | 1 | 7 | 92 |
| 55 | 55 | 1 | 2 | 97 | 0 | 9 | 91 |
| 55 | 75 | 4 | 10 | 86 | 1 | 18 | 83 |
| 55 | 105 | 2 | 17 | 81 | 6 | 21 | 83 |
| | | 8 | 53 | 539 | 8 | 67 | 525 |

cutset when the graphs have a small number of loops. This case is interesting because the conditioning inference algorithm is most appropriate for these networks. MGA made no mistakes while the other three algorithms made between 4 and 5 mistakes each by one vertex (the minimum loop cutsets contained between 2 and 4 vertices).

Next we tested larger graphs. The first portion of Table 1 compares between GA′ and A2 showing that GA′ performs better than A2 in 53 of the 61 graphs (87%) in which the algorithms disagree (out of 600 graphs tested). Each line in the table is based on 100 randomly generated graphs. The output columns show the number of graphs for which the two algorithms had an output of the same size and the number of graphs each algorithm performed better than the other. Thus even our simple greedy algorithm GA′ performs better than A2. The reason for this is the reduction from the loop cutset problem to the weighted vertex feedback set problem which allows the algorithm to select vertices that have parents while A2 unjustifiably does not select such vertices (unless they have no pair of parents residing on the same loop). Similar empirical results and the same explanation applies to A1. The second portion of the table shows that MGA performs better than GA′ in 67 of the 75 graphs (89%) in which the algorithms disagreed. Comparing MGA and A2 in the same fashion (600 graphs) showed that MGA performed better than A2 in 109 of the 116 graphs in which the algorithms disagreed. Similarly, MGA performed better than A1 in 135 of the 137 graphs in which these algorithms disagreed.

Finally, we repeated some of the experiments except that now each vertex was associated with a random number of values (between 2 and 6, 2 and 8, and 2 and 10). The results are summarized in Table 2. The two algorithms, A1 and MGA, output loop cutsets of the same size in 55% of the graphs and when the algorithms disagreed, then in 81% of these graphs MGA performed better than A1. The ratio obtained between the number of instances of the algorithms solution and a minimum solution was 1.22 for MGA and 1.44 for A1 (using the 300 graphs in the table for which the number of vertices is 15 and number of edges 25). Not surprisingly, the averaged number of instances by which the two algorithms' outputs differ, when the algorithms disagree, grows as the graphs being tested become larger.

To repeat this experiment with A2 required us to make a small change in A2 because it is not designed to run with vertices having different number of values. We adopted the approach of A1 which selects vertices (with at most one parent) according to their degree and if there are several candidates the one with the least number of values is selected for the loop cutset. Combining this idea with the A2 algorithm defines an

Table 2

| $\|V\|$ | $\|E\|$ | Values | A1 | MGA | Eq. |
|---|---|---|---|---|---|
| 15 | 25 | 2-6 | 1 | 17 | 82 |
| 15 | 25 | 2-8 | 2 | 17 | 81 |
| 15 | 25 | 2-10 | 2 | 19 | 79 |
| 55 | 105 | 2-6 | 13 | 58 | 29 |
| 55 | 105 | 2-8 | 17 | 51 | 32 |
| 55 | 105 | 2-10 | 15 | 55 | 30 |
| | | | 50 | 217 | 333 |

algorithm we call the weighted A2 algorithm (WA2). The results obtained were that MGA performed better than WA2 in 175 of the 224 graphs in which the algorithms disagreed (out of 600). The ratio obtained between the number of instances of the algorithms' solution and a minimum solution was 1.22 for MGA and 1.33 for WA2.

## 5. Discussion

We have presented simple algorithms that given a Bayesian network output a loop cutset whose instance size is less than twice the optimal size in a logarithmic scale (in the worst case). Furthermore, we have experimentally shown that on the average our algorithms perform much better than in the worst case. Consequently, before running the probabilistic computation of the method of conditioning, we can evaluate with high precision the optimal complexity of its running time. Furthermore the approximation algorithms for the weighted vertex feedback set have applications in areas of computer science other than AI.

The leading inference algorithm for Bayesian networks is the clique tree algorithm [17] which has been further developed in [14,15]. In fact, Shachter et al. [22] have recently shown that the weight of the largest clique is bounded by the weight of the union of the loop cutset and the largest parent set of a vertex in a Bayesian network, implying that the clique tree algorithm is superior to the conditioning algorithm.

One possible change to the method of conditioning makes this inference methodology quite useful in certain circumstances. Horvitz et al. [13] show how to rank the instances of a loop cutset according to their prior probabilities assuming all variables in the cutset are marginally independent. The conditioning algorithm can then be run according to this ranking and the answer to a query be given as an interval that shrinks towards the exact solution as more instances of the loop cutset are considered [12,13]. So if the maximal clique is too large to store (and therefore, according to [22], the loop cutset is also often too large to handle), one can still perform approximate inferences using the conditioning algorithm.

## Remark

While this work was at its final stages of preparation we became aware of a different method for the WVFS problem that achieves a performance ratio of 2 [2]. A quick examination of our own work in light of this information revealed that our method also achieves a performance ratio of 2.

An early version of this paper was presented at the Tenth Uncertainty in Artificial Intelligence Conference, Seattle, WA, July 1994.

## References

[1] R. Bar-Yehuda, D. Geiger, J. Naor and R. Roth, Approximation algorithms for the vertex feedback set problems with applications to constraint satisfaction and Bayesian inference, in: *Proceedings 5th Annual ACM-SIAM Symposium On Discrete Algorithms*, Arlington, VI (1994).

[2] P. Berman, Personal communication, Pennsylvania State University, University Park, PA (1994).

[3] V. Chvatal, A greedy heuristic for the set-covering problem, *Math. Oper. Res.* **4** (1979) 233-235.

[4] K.L. Clarkson, A modification of the greedy algorithm for vertex cover, *Inf. Process. Lett.* **16** (1983) 23-25.

[5] T.H. Cormen, C.E. Leiserson and R.L. Rivest, *Introduction to Algorithms* (MIT Press, London, 1990).

[6] R. Dechter, Enhancement schemes for constraint processing: backjumping, learning, and cutset decomposition, *Artif. Intell.* **41** (1990) 273-312.

[7] R. Dechter and J. Pearl, The cycle cutset method for improving search performance in AI, in: *Proceedings Third IEEE Conference on AI Applications*, Orlando, FL (1987) 224-230.

[8] M.L. Fredman and R.E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, *J. ACM* **34** (1987) 596-615.

[9] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (Freeman, San Francisco, CA, 1979).

[10] D. Geiger and J. Pearl, On the logic of causal models, in: R.D. Shachter, T.S. Levitt, L.N. Kanal and J.F. Lemmer, eds., *Proceedings of Uncertainty in Artificial Intelligence* **4** (North-Holland, New York, 1990) 3-14.

[11] D. Geiger, T.S. Verma and J. Pearl, Identifying independence in Bayesian networks, *Networks* **20** (1990) 507-534.

[12] E.J. Horvitz, Computation and action under bounded resources, Ph.D. dissertation, Stanford University, Stanford, CA (1990).

[13] E.J. Horvitz, H.J. Suermondt and G.H. Cooper, Bounded conditioning: flexible inference for decisions under scarce resources, in: *Proceedings 5th Conference on Uncertainty in Artificial Intelligence*, Windsor, Ont. (1989) 182-193.

[14] F.V. Jensen, S.L. Lauritzen and K.G. Olesen, Bayesian updating in causal probabilistic networks by local computations, *Comput. Stat. Quart.* **4** (1990) 269-282.

[15] F.V. Jensen, K.G. Olesen and S.K. Andersen, An algebra of Bayesian belief universes for knowledge-based systems, *Networks* **20** (1990) 637-659.

[16] H. Kim and J. Pearl, A computational model for combined causal and diagnostic reasoning in inference systems, in: *Proceedings IJCAI-83*, Karlsruhe (Morgan Kaufmann, San Mateo, CA, 1983) 190-193.

[17] S.L. Lauritzen and D.J. Spiegelhalter, Local computations with probabilities on graphical structures and their application to expert systems (with discussion), *J. Roy. Stat. Soc. Ser. B* **50** (1988) 157-224.

[18] R. Motwani, Lecture notes on approximation algorithms, Report STAN-CS-92-1435, Computer Science Department, Stanford University, Stanford, CA (1992).

[19] J. Pearl, Fusion, propagation and structuring in belief networks, *Artif. Intell.* **29** (1986) 241-288.

[20] J. Pearl, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference* (Morgan Kaufmann, San Mateo, CA, 1988).

[21] R.D. Shachter, Evaluating influence diagrams, *Oper. Res.* **34** (1986) 871-882.

[22] R.D. Shachter, S.K. Andersen and P. Szolovits, Global conditioning for probabilistic inference in belief networks, in: *Proceedings Tenth Conference on Uncertainty in Artificial Intelligence*, Seattle, WA (1994) 514-522.

[23] J. Stillman, On heuristics for finding loop cutsets in multiply connected belief networks, in: *Proceedings Sixth Conference on Uncertainty in Artificial Intelligence*, Cambridge, MA (1990) 265-272.

[24] H.J. Suermondt and G.F. Cooper, Probabilistic inference in multiply connected belief networks using loop cutsets, *Int. J. Approx. Reasoning* **4** (1990) 283-306.
[25] H.J. Suermondt and G.F. Cooper, Initialization for the method of conditioning in Bayesian belief networks, *Artif. Intell.* **50** (1991) 83-94.
[26] T. Verma and J. Pearl, Causal networks: Semantics and expressiveness, in: R.D. Shachter, T.S. Levitt, L.N. Kanal and J.F. Lemmer, eds., *Proceedings of Uncertainty in Artificial Intelligence* **4** (North-Holland, New York, 1990) 69-76.