Approximation Algorithms for the Vertex Feedback Set Problem with Applications to Constraint Satisfaction and Bayesian Inference

Reuven Bar-Yehuda * Dan Geiger Joseph (Seffi) Naor †
Ron M. Roth

Computer Science Department
Technion
Haifa 32000, ISRAEL

Abstract

A *vertex feedback set* of an undirected graph is a subset of vertices that intersects with the vertex set of each cycle in the graph. Given an undirected graph G with n vertices and weights on its vertices, polynomial-time algorithms are provided for approximating the problem of finding a vertex feedback set of G with a smallest weight. When the weights of all vertices in G are equal, the performance ratio attained by these algorithms is $4-(2/n)$. This improves a previous algorithm which achieved an approximation factor of $\sqrt{\log n}$ for this case. For general vertex weights, the performance ratio becomes $\min\{2\Delta^2, 4\log_2 n\}$ where Δ denotes the maximum degree in G. For the special case of planar graphs this ratio is reduced to 10. An interesting special case of weighted graphs where a performance ratio of $4-(2/n)$ is achieved is the one where a prescribed subset of the vertices, so called *blackout* vertices, is not allowed to participate in any vertex feedback set.

It is shown how these algorithms can improve the search performance for constraint satisfaction problems. An application in the area of Bayesian inference of graphs with blackout vertices is also presented.

*Part of this research was done while the author was visiting SUNY at Buffalo. This research was supported by the fund for the promotion of research at the Technion.

†Part of this research was done while the author was visiting DIMACS, Rutgers University, NJ.

1 Introduction

Let $G = (V, E)$ be an undirected graph, and let $w : V(G) \to I\!R^+$ be a weight function on the vertices of G. A *cycle* in G is a path whose two terminal vertices coincide. A *vertex feedback set* of G is a subset of vertices $F \subseteq V(G)$ 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 forest. A *minimum vertex feedback set of a weighted graph* (G, w) is a vertex feedback set of G of minimum weight. The weight of a minimum vertex feedback set will be denoted by $\mu(G, w)$.

The *Weighted Vertex Feedback Set (WVFS) Problem* is defined as finding a minimum vertex feedback set of a given weighted graph (G, w). The special case where w is the constant function 1 is called the *Unweighted Vertex Feedback Set (UVFS) Problem.* Given a graph G and an integer k, the problem of deciding whether $\mu(G, 1) \leq k$ is known to be NP-Complete [GJ79, pp. 191–192]. Hence, it is natural to look for efficient approximation algorithms for the vertex feedback set problem, particularly in view of the recent applications of such algorithms in artificial intelligence as we show in the sequel.

Suppose A is an algorithm that finds a vertex feedback set F_A for any given undirected weighted graph (G, w). We denote the sum of weights of the vertices in F_A by $w(F_A)$. The *performance ratio of A for (G, w)* is defined by $R_A(G, w) = w(F_A)/\mu(G, w)$. When $\mu(G, w) = 0$ we define $R_A(G, w) = 1$ if $w(F_A) = 0$ and $R_A(G, w) = \infty$ if $w(F_A) > 0$. The *performance ratio $r_A(n, w)$ of A for w* is the supremum of $R_A(G, w)$ over all graphs G with n vertices and for the same weight function w. When w is the constant function 1, we call $r_A(n, 1)$ the *unweighted performance ratio of A*. Finally, the *performance ratio $r_A(n)$ of A* is the supremum of $r_A(n, w)$ over all weight functions w defined over graphs with n vertices.

An approximation algorithm for the UVFS Problem that achieves an unweighted performance ratio of $2 \log_2 n$ is essentially contained in a lemma due to Erdős and Pósa [EP62]. This result was improved by Monien and Schulz [MS81], where they achieved a performance ratio of $\sqrt{\log n}$. In Section 2 we provide an approximation algorithm for the UVFS Problem that achieves an unweighted performance ratio of at most $4 - (2/n)$. Our algorithm draws upon a theorem by Simonovits [Si67] and our analysis uses a result by Voss [Vo68].

In Section 3 we present two algorithms for the WVFS Problem. We first devise a primal-dual algorithm which is based on formulating the WVFS Problem as an instance of the *set cover* problem. The algorithm has a performance ratio of 10 for weighted planar graphs and a performance ratio of $4 \log_2 n$ for general weighted graphs. This ratio is achieved by extending the Erdős-Pósa Lemma to weighted graphs. The second algorithm presented in Section 3 achieves a performance ratio of $2\Delta^2(G)$ for general weighted graphs, where $\Delta(G)$

is the maximum degree of G. This result is in particular interesting for low degree graphs. We conjecture that a constant performance ratio is nevertheless attainable by a polynomial time algorithm for all weighted graphs.

In Section 4 we consider a special case of the WVFS problem, where a prescribed subset of the vertices, called *blackout vertices*, is not allowed to participate in any vertex feedback set. We further assume that the blackout vertices induce a forest. We show that the bounds previously achieved can be extended to this case too.

Our interest in graphs with blackout vertices is motivated by the *loop cutset* problem and its application to the updating problem in Bayesian inference in Section 5. Let D be a directed graph. A *loop* in D is defined as a subgraph whose underlying undirected graph is a cycle. Given a weighted directed graph (D, w), the loop cutset problem is defined as follows: find a minimum weight set of vertices F in D such that the vertex set of every loop Γ in D intersects with F at a vertex which has at least one outgoing edge in Γ. The performance ratios obtained for this problem are similar to those obtained for the vertex feedback set problem.

Another application of approximation algorithms for the UVFS Problem in artificial intelligence due to Dechter and Pearl is as follows [DP90]. We are given a set of variables $x_1, x_2, \ldots, x_n$, where each x_i takes its values from a finite domain D_i. Also, for every $i < j$ we are given a constraint subset $R_{i,j} \subseteq D_i \times D_j$ which defines the allowable pairs of values that can be taken by the pair of variables (x_i, x_j). Our task is to find an assignment for all variables such that all the constraints $R_{i,j}$ are satisfied. With each instance of the problem we can associate an undirected graph G whose vertex set is the set of variables, and for each constraint $R_{i,j}$ which is *strictly* contained in $D_i \times D_j$ (i.e., $R_{i,j} \neq D_i \times D_j$) there is an edge in G connecting x_i and x_j. The resulting graph G is called a *constraint network* and it is said to represent a *constraint satisfaction problem*.

A common method for solving a constraint satisfaction problem is by backtracking, that is, by repeatedly assigning values to the variables in a predetermined order and then backtracking whenever reaching a dead end. This approach can be improved as follows. First, find a vertex feedback set of the constraint network. Then, arrange the variables so that variables in the vertex feedback set precede all other variables, and apply the backtracking procedure. Once the values of the variables in the vertex feedback set are determined by the backtracking procedure, the algorithm switches to a polynomial-time procedure SOLVE-TREE that solves the constraint satisfaction problem in the remaining forest. If SOLVE-TREE succeeds, a solution is found; otherwise, another backtracking phase occurs.

The complexity of the above modified backtracking algorithm grows exponentially with the size of the vertex feedback set: If a vertex feedback set contains k variables, each having a domain of size 2, then the procedure SOLVE-TREE might be invoked up to 2^k times. A procedure SOLVE-TREE that runs in polynomial-time was developed by Dechter and Pearl, who also proved the optimality of their tree algorithm [DP88]. Consequently, our approximation algorithm for finding a small vertex feedback set reduces the complexity of solving constraint satisfaction problems through the modified backtracking algorithm. Furthermore, if the domain size of the variables varies, then SOLVE-TREE is called a number of times which is bounded from above by the product of the domain-sizes of the variables whose corresponding vertices participate in the vertex feedback set. If we take the logarithm of the domain size as the weight of a vertex, then solving the WVFS problem with these weights optimizes the complexity of the modified backtracking algorithm in the case where the domain size is allowed to vary.

2 The Unweighted Vertex Feedback Set Problem

In this section we consider the approximation of the UVFS Problem described in Section 1. Namely, given an undirected graph G, find a small vertex feedback set for $(G,1)$. Throughout this section a graph means an undirected graph with at least one vertex and with possibly parallel edges and self-loops.

2.1 Definitions

Let G be an undirected graph with a set of vertices $V(G)$ and a set of edges $E(G)$ and let v be a vertex in G. A *neighbor* of v is a vertex $u \in V(G)$ which is connected to v by an edge in $E(G)$. The degree $\Delta_G(v)$ of v in G is the number of edges that are incident with v in G. A self-loop at a vertex v contributes 2 to the degree of v. The degree of G, denoted $\Delta(G)$, is the largest among all degrees of vertices in G.

A vertex in G of degree 1 is called an *endpoint*. A vertex of degree 2 is called a *linkpoint* and a vertex of any higher degree is called a *branchpoint*. A graph G is called *rich* if every vertex in G is a branchpoint. A graph is called a *singleton* if it contains only one vertex. A singleton is called *naked* if it has no edges; otherwise it is called *self-looped*. Note that for a singleton we have $\mu(G,1) = 1$ if it is self-looped and $\mu(G,1) = 0$ if it is naked.

Two cycles in a graph G are *independent* if their vertex sets are disjoint. Note that the size of any vertex feedback set of G is bounded from below by the largest number of pairwise independent cycles that can be found in G. A cycle Γ in G is called *simple* if it visits every vertex in $V(G)$ at most once. Clearly, a set F is a vertex feedback set of G if and only if it intersects with every simple cycle in G.

A graph G is connected if for every two vertices there is a connecting path in G. Every graph G can be decomposed uniquely into isolated connected components $G_1, G_2, \ldots, G_k$. Similarly, every vertex feedback set F of G can be partitioned into vertex feedback sets $F_1, F_2, \ldots, F_k$ such that F_i is a vertex feedback set of G_i. Hence, $\mu(G, 1) = \sum_{i=1}^{k} \mu(G_i, 1)$.

For a graph G we define the *reduction* G' of G by the following procedure.

Algorithm Reduction (*Input:* graph G; *Output:* reduction G' of G);

$H \leftarrow G$;
While H contains an endpoint v do
  delete v and its incident edge from H;
While H contains a linkpoint v without a self-loop do begin:
  connect the two neighbors of v by a new edge;
  remove v from the graph with its two incident edges;
end;
$G' \leftarrow H$.

Let $H_1, H_2, \ldots, H_{t-1}, H_t = G'$ be the values of H while starting each reduction iteration of the second loop in REDUCTION. Also, let v_i be the linkpoint that was removed from H_i to obtain H_{i+1}. Suppose F is a vertex feedback set of H_{i+1} for some i, $1 \leq i < t$ and let Γ be a cycle in H_i that passes through v_i. A reduction of Γ obtained by replacing the linkpoint v_i on Γ by an edge connecting the neighbors of v_i yields a cycle $\hat{\Gamma}$ in H_{i+1}. The vertex set of $\hat{\Gamma}$ intersects the set F. Hence, F is also a vertex feedback set of H_i. On the other hand, every vertex feedback set F of H_i can be made a vertex feedback set of H_{i+1} by replacing v_i in F with one of its neighbors in H_i. Therefore, we have the following.

Lemma 1 *Let G be a graph and let G' be a reduction of G. Then,*

(a) every vertex feedback set of G' is also a vertex feedback set of G;

(b) $\mu(G', 1) = \mu(G, 1)$.

The next two properties of reduction graphs are also easily verified.

Lemma 2 *Let G be a graph with no endpoints and let G' be a reduction of G. Then, every branchpoint in G is also a branchpoint in G' and $\Delta_{G'} = \Delta_G$.*

Lemma 3 *Let G be a connected graph and let G' be a reduction of G. Then G' is either a connected rich graph or a singleton, and G' is a naked singleton if and only if G is a tree.*

Note that the reduction of a graph G is unique up to isomorphism. The complexity of computing the reduction of G is at most linear in $|E(G)|$.

2.2 Logarithmic unweighted performance ratio

The basis of the first approximation algorithm is the following lemma due to Erdős and Pósa [EP62].

Lemma 4 ([EP62], Lemma 3) *The shortest cycle in any rich graph G is of length $\leq 2\log_2|V(G)|$.*

Proof. Let t be the smallest integer such that $3 \cdot 2^{t-1} > |V(G)|$. Apply a breadth-first-search (BFS) on G of depth t starting at some vertex v. We now claim that the search hits some vertex twice and so there exists a cycle of length $\leq 2t$ in G. Indeed, if it were not so, then the induced BFS tree would contain at least $3 \cdot 2^{t-1}$ distinct vertices of G, which is a contradiction. □

Lemma 4 suggests the following algorithm for finding a small vertex feedback set in a graph G.

Algorithm MiniCycle (*Input:* graph G; *Output:* vertex feedback set F of G);

$F \leftarrow \emptyset$; $H \leftarrow G$;
While H is not a forest do begin:
  Find the reduction H' of H;
  Find the shortest cycle Γ' in H';
  Remove $V(\Gamma')$ (with any incident edges) from H';
  $H \leftarrow H'$;
  $F \leftarrow V(\Gamma') \cup F$;
end.

Finding a shortest cycle can be done by applying BFS at each vertex until a cycle is found and then selecting the smallest. A more efficient approach for finding the shortest cycle is described in [IR78].

Theorem 5 *The unweighted performance ratio of algorithm* MiniCycle *is at most $2\log_2|V(G)|$.*

Proof. Let G be the input graph to MiniCycle and let Γ' be a cycle found at some iteration of MiniCycle, when applied to a reduction graph H' of H. If H' contains a self-looped singleton then $|V(\Gamma')| = 1$. Otherwise, by Lemma 4 we have $|V(\Gamma')| \leq 2\log_2|V(H')|$. Now, every vertex feedback set of G must contain at least one element of $V(\Gamma')$, and cycles Γ' selected in different iterations of MiniCycle are pairwise independent. Since the algorithm takes the whole set $V(\Gamma')$ in each iteration, it errs by a factor of no more than

$|V(\Gamma')| \le 2\log_2 |V(G)|$. $\square$

Lemma 4 was obtained by Erdős and Pósa while estimating the smallest number of edges in a graph which contains a given number of pairwise independent cycles. Later on, in [EP64], they provided bounds on the value of $\mu(G,1)$ in terms of the largest number of pairwise independent cycles in G. Tighter bounds on $\mu(G,1)$ were obtained by Simonovits [Si67] and Voss [Vo68]. An approximation algorithm which achieved a performance ratio of $\sqrt{\log n}$ was then given by Monien and Schulz [MS81]. In the following sections we use some results of [Si67] and [Vo68] in order to obtain better approximation algorithms for the UVFS Problem.

2.3 Unweighted performance ratio 5

Our next algorithm which attains a performance ratio ≤ 5 makes use of the following definitions. A *2-4-subgraph* of G is a subgraph H of G such that the degree in H of every vertex is 2, 3, or 4. A 2-4-subgraph exists in any graph which is not a forest. A *maximal 2-4-subgraph* of G is a 2-4-subgraph of G which is not a subgraph of any other 2-4-subgraph of G. A maximal 2-4-subgraph can be found easily by applying depth-first-search (DFS) on G.

Algorithm SubG-2-4 (*Input:* graph G; *Output:* vertex feedback set F of G);

if G **is a forest then**
  $F \leftarrow \emptyset$;
else begin:
  using DFS, find a maximal 2-4-subgraph H **of** G;
  find the reduction H' **of** H;
  $F \leftarrow V(H')$;
end.

The complexity of SubG-2-4 is dominated by the complexity of DFS, and hence it is linear in $|E(G)|$.

Proposition 6 *For every graph G, the set F computed by* SubG-2-4 *is a vertex feedback set of G.*

Proof. Assume to the contrary that $F = V(H')$ is not a vertex feedback set of G. Then, there exists a cycle Γ in G such that $V(\Gamma) \cap V(H') = \emptyset$. On the other hand, by Lemma 1(a), the set $V(H')$ is a vertex feedback set of H. Hence, the cycle Γ cannot be entirely contained in H. Now, by Lemma 2 the set $V(H')$ contains all branchpoints of H. Recalling that H

contains no endpoints, we conclude that the cycle Γ intersects with H only in linkpoints of the latter.

Without loss of generality we can assume that Γ is simple. Consider the graph H^* defined by $V(H^*) = V(H) \cup V(\Gamma)$ and $E(H^*) = E(H) \cup E(\Gamma)$. Since Γ visits each linkpoint of H at most once, the graph H^* is a 2-4-subgraph of G. And, since Γ is not entirely contained in H, the latter is a proper subgraph of H^*. However, this contradicts the maximality of H. Therefore, $F = V(H')$ has to be a vertex feedback set of G. $\square$

To prove our next theorem we use the following lemma due to Voss. The proof of this lemma is included here for future reference in this paper.

Lemma 7 ([Vo68], Lemma 4) *Let G be a rich graph. Then, for every vertex feedback set F of G,*

$$|V(G)| \ \le \ (\Delta(G) + 1)\,|F| \ - \ 2\,.$$

Proof. Suppose $F = V(G)$. In this case we have $|V(G)| \le 4|V(G)| - 2 \le (\Delta(G) + 1)|V(G)| - 2$ and, therefore, the lemma holds trivially. So we assume from now on that $|F| < |V(G)|$.

Let E_F denote the set of edges in $E(G)$ whose terminal vertices are all vertices in F. Define $X = V - F$ and let E_X denote the set of edges in $E(G)$ whose terminal vertices are all vertices in X. Also, let $E_{F,X}$ denote the set of those edges in G that connect vertices in F with vertices in X. Clearly, E_F, E_X, and $E_{F,X}$ form a partition on $E(G)$. Now, the graph obtained by deleting F from G is a nonempty forest on X and, therefore, $|E_X| \le |X| - 1$. However, each vertex in X is a branchpoint in G and, so,

$$3\,|X| \ \le \ \sum_{v \in X} \Delta_G(v) \ = \ |E_{F,X}| \ + \ 2\,|E_X| \ \le \ |E_{F,X}| \ + \ 2\,(|X| - 1)$$

i.e.,

$$|E_{F,X}| \ \ge \ |X| \ + \ 2 \ = \ |V(G)| \ - \ |F| \ + \ 2\,.$$

On the other hand,

$$\Delta(G)\,|F| \ \ge \ \sum_{v \in F} \Delta_G(v) \ = \ |E_{F,X}| \ + \ 2\,|E_F|\,.$$

Combining the last two inequalities we obtain

$$|V(G)| \ \le \ (\Delta(G) + 1)\,|F| \ - \ 2\,|E_F| \ - \ 2\,. \qquad \square$$

Theorem 8 *The unweighted performance ratio of* SUBG-2-4 *is at most* 5.

Proof. Let G, H, H', and F be as in SUBG-2-4 and let $H_1, H_2, \ldots, H_k$ be the connected components of H. Each connected component H_i' of H' is the reduction of a respective component H_i of H. By Lemma 3, each component H_i' is either a singleton or a rich graph. If H_i' is rich then, by Lemma 7,

$$\mu(H_i', 1)(\Delta_{H_i'} + 1) \;\geq\; |V(H_i')| \,.$$

Obviously, this last inequality holds also when H_i' is a singleton. Now, by Lemma 2 we have $\Delta_{H_i'} = \Delta_{H_i}$. Therefore,

$$\mu(H', 1) \;=\; \sum_{i=1}^{k} \mu(H_i', 1) \;\geq\; \sum_{i=1}^{k} \frac{|V(H_i')|}{\Delta_{H_i'} + 1} \;\geq\; \frac{|V(H')|}{5} \;=\; \frac{|F|}{5} \,. \tag{1}$$

On the other hand, by Lemma 1(b) we have

$$\mu(G) \;\geq\; \mu(H, 1) \;=\; \mu(H', 1) \,,$$

and, so, by (1) we obtain the inequality $|F|/\mu(G,1) \leq 5$. □

2.4 Unweighted performance ratio less than 4

Next we obtain an improvement on the unweighted performance ratio of SubG-2-4 by using a different subgraph H of G.

Let G be a graph. A *2-3-subgraph* of G is a subgraph H of G such that the degree in H of every vertex is 2 or 3. Similarly, a *maximal 2-3-subgraph* of G is a 2-3-subgraph of G which is not a subgraph of any other 2-3-subgraph of G.

A linkpoint v in a 2-3-subgraph H of G is called a *critical linkpoint* if there is a cycle Γ in G such that $V(\Gamma) \cap V(H) = \{v\}$. Such a cycle Γ in G is called a *witness cycle of* v. Note that we can assume a witness cycle to be simple and, so, verifying whether a linkpoint v in H is a critical linkpoint is easy: Remove the set of vertices $V(H) - \{v\}$ from G, with all incident edges, and apply BFS to check whether there is a cycle through v in the remaining graph.

Let Γ be a simple cycle which is an isolated connected component of a 2-3-subgraph H of G. In each such cycle, we set one linkpoint arbitrarily to be the *representing linkpoint* of Γ.

The following algorithm improves the unweighted performance ratio of SubG-2-4 by using a 2-3-subgraph H of G instead of a 2-4-subgraph of G.

Algorithm SubG-2-3(*Input:* graph G; *Output:* vertex feedback set F of G);

If G is a forest then
    $F \leftarrow \emptyset$;

else begin:

Using DFS, find a maximal 2-3-subgraph H of G;

Using BFS, find the set X of critical linkpoints in H;

Let Y be the set of branchpoints in H;

Find the set Z of representing linkpoints of those isolated cycles in H that do not contain any critical linkpoints;

$F \leftarrow X \cup Y \cup Z$;

end.

It is straightforwrd to verify that the complexity of SubG-2-3 is linear in $|E(G)|$. The analysis of SubG-2-3 is based on the following two lemmas that were used in the proof of Theorem 1 in [Si67].

Lemma 9 *Let H be a maximal 2-3-subgraph of G and let Γ be a simple cycle in G. Then, one of the following holds:*

(a) Γ is a witness cycle of some critical linkpoint of H, or —

(b) Γ passes through some branchpoint of H, or —

(c) Γ is an isolated connected component of H.

Proof. Let Γ be a cycle in G and assume to the contrary that neither of (a)–(c) holds. This implies in particular that Γ cannot be entirely contained in H. We distinguish between two cases:

Case 1: Γ does not intersect with H at all. In this case we could join Γ and H to obtain a 2-3-subgraph H^* of G that contains H as a proper subgraph. This however contradicts the maximality of H.

Case 2: Γ intersects with H only in linkpoints of the latter. First note that Γ must intersect with H in at least two distinct linkpoints of H, or else Γ would be a witness cycle of the intersecting (critical) linkpoint. Since Γ is not contained in H by assumption, we can find two linkpoints v_1 and v_2 in $V(\Gamma) \cap V(H)$ that are connected by a path P along Γ such that $V(P) \cap V(H) = \{v_1, v_2\}$ and P is not entirely contained in H. Joining P and H, we obtain a 2-3-subgraph of G that contains H as a proper subgraph, thus contradicting the maximality of H. $\square$

Lemma 10 *Let H be a maximal 2-3-subgraph of G and let Γ_1 and Γ_2 be witness cycles in G of two distinct critical linkpoints in H. Then Γ_1 and Γ_2 are independent cycles.*

Proof. Let v_1 and v_2 be the critical linkpoints associated with Γ_1 and Γ_2, respectively, and assume to the contrary that $V(\Gamma_1) \cap V(\Gamma_2)$ contains a vertex $u \in V(G)$. Then, there is a path P in G that runs along parts of the cycles Γ_1 and Γ_2, starting from v_1, passing through

u, and ending at v_2. Since Γ_1 and Γ_2 are witness cycles, we have $V(P) \cap V(H) = \{v_1, v_2\}$. And, since v_1 and v_2 are distinct critical linkpoints, the vertex u cannot possibly coincide with either of them. Therefore, the path P is not entirely contained in H. Joining P and H we obtain a 2-3-subgraph of G that contains H as a proper subgraph, thus reaching a contradiction. □

Proposition 11 *For every graph G, the set F computed by* SUBG-2-3 *is a vertex feedback set of G.*

Proof. Let Γ be a cycle in G. We follow the three cases of Lemma 9 to show that $V(\Gamma) \cap F \neq \emptyset$.

(a) Γ is a witness cycle of some critical linkpoint of H. By construction, all critical linkpoints of H are in F.

(b) Γ passes through some branchpoint of H. By construction, all branchpoints of H are in F.

(c) Γ is an isolated connected component of H. By construction, there always exists a vertex v of $V(\Gamma)$ which is contained in F: either v is a critical linkpoint or v is a representing linkpoint of Γ. □

Lemma 12 *Let F be the vertex feedback set computed by* SUBG-2-3 *for a graph G which is not a forest. Then,*

$$|F| \leq 4\,\mu(G,1) - 2 .$$

Proof. Let H, X, Y, and Z be as in SUBG-2-3. Suppose $\mu(G,1) = 1$. Then, all cycles in G pass through some vertex v in G and, so, no vertex other than v can be a critical linkpoint in H. Now, if v is a linkpoint in H, then H is a cycle. Otherwise, one can readily verify that H must contain exactly two branchpoints. In either case we have $|F| \leq 2$. We assume from now on that $\mu(G,1) \geq 2$.

For every $v_i \in X$, let Γ_i be some witness cycle of v_i in G. By Lemma 10, the cycles Γ_i are pairwise independent.

Let $\{\Gamma_j^*\}$ be the set of the $|Z|$ isolated cycles in H that do not contain any critical linkpoints of H. Clearly, these cycles are pairwise independent. Furthermore, neither of them intersects with any of the witness cycles Γ_i. It thus follows that every vertex feedback set of G must contain at least one vertex of each of the $|X| + |Z|$ independent cycles $\{\Gamma_i\} \cup \{\Gamma_j^*\}$. Therefore, $\mu(G,1) \geq |X| + |Z|$. On the other hand, we recall that $|F| = |X| + |Y| + |Z|$.

We distinguish between the following two cases.

Case 1: $|Y| \le 2|X|$. Here we have,

$$|F| \;=\; |X|+|Y|+|Z| \;\le\; 3|X|+|Z| \;\le\; 3\mu(G,1) \;\le\; 4\mu(G,1)-2\,.$$

Case 2: $|Y| > 2|X|$. Let H_1 be the subgraph of H obtained by removing all critical linkpoints and all isolated cycles of H. We further assume here that, with each deletion of a critical linkpoint from H, we also remove recursively all the resulting endpoints (clearly, each vertex is removed with its incident edges). Hence, the graph H_1 contains no endpoints. Now, a deletion of each linkpoint from H, along with any resulting endpoints, can decrease the number of branchpoints by 2 at most. Therefore, the number of branchpoints left in H_1 is at least $|Y|-2|X|>0$.

Let H_1' be a reduction of H_1 and let H_2 be obtained by removing all the singleton components from H_1'. By Lemma 2, the graph H_2 is a rich graph and contains at least $|Y|-2|X|$ branchpoints. Hence, by Lemma 7 and Lemma 1(b),

$$|Y|-2|X| \;\le\; 4\mu(H_2,1)-2 \;\le\; 4\mu(H_1',1)-2 \;=\; 4\mu(H_1,1)-2\,,$$

and, so,

$$\begin{aligned} |F| \;&=\; |X|+|Y|+|Z| \\ &\le\; 4|X|+4|Z|+|Y|-2|X| \\ &\le\; 4(|X|+|Z|+\mu(H_1,1))-2\,. \end{aligned} \tag{2}$$

Now, any cycle of G which is entirely contained in H_1 cannot possibly intersect with any of the cycles Γ_i and Γ_j^*. So,

$$|X|+|Z|+\mu(H_1,1) \;\le\; \mu(G,1)\,.$$

The claim now follows by plugging the last inequality into (2). □

Theorem 13 *The unweighted performance ratio of* SubG-2-3 *is at most* $4-(2/|V(G)|)$.

Proof. This follows immediately from Lemma 12. □

3 Weighted Vertex Feedback Set

In this section we consider the approximation of the WVFS Problem described in Section 1. Namely, given an undirected graph G and a weight function w on its vertices, find a vertex feedback set of (G,w) with minimum weight. As in the previous section, we assume that G may contain parallel edges and self loops.

A graph is called *branchy* if it has no endpoints and, in addition, its set of linkpoints induces an independent set, i.e., each linkpoint is connected only to branchpoints. For a weighted graph (G, w), we define the *reduction* (G', w') of (G, w) by the following procedure ReductionW that repeatedly replaces a chain of linkpoints by a single linkpoint with weight equal to the minimum weight of the vertices in the chain.

Algorithm ReductionW (*Input:* (G, w); *Output:* reduction (G', w') of (G, w));

$(G', w') \leftarrow (G, w)$;
While G' contains an endpoint v do
    Delete v and its incident edge from G';
While G' contains a linkpoint v adjacent to another linkpoint u do begin:
    Connect the two neighbors of v by a new edge;
    Set the new weight w' of u to be $\min(w'(u), w'(v))$;
    Remove v from the graph with its two incident edges;
end.

The following lemma can be easily verified.

Lemma 14 *Let (G, w) be a weighted graph and let (G', w') be a reduction of (G, w). Then, G' is a branchy graph and $\mu(G', w') = \mu(G, w)$.*

We note that the complexity of ReductionW is linear in $|E(G)|$.

We are now ready to present our algorithms for finding an approximation for a minimum-weight vertex feedback set of a given weighted graph. In Section 3.1 we give an algorithm that achieves a performance ratio of $4 \log |V(G)|$. In Section 3.2 we present an algorithm that achieves a performance ratio of $2\Delta^2(G)$.

3.1 The primal-dual algorithm

The algorithm presented in this section is a generalization of the one presented in Section 2.2. In each iteration of the algorithm, we first find a reduction of the graph and then find a cycle Γ with a smallest number of vertices in the reduction graph. The algorithm then sets δ to be the smallest among the weights of vertices in $V(\Gamma)$. This value of δ is subtracted, in turn, from the weight of each vertex in $V(\Gamma)$. Vertices whose weight becomes zero are added to the vertex feedback set and deleted from the graph. Each such iteration is repeated until the graph is exhausted.

Algorithm MiniWCycle (*Input:* (G, w); *Output:* vertex feedback set F of (G, w));

$F \leftarrow \emptyset$; $(H, w_H) \leftarrow (G, w)$;
While H is not a forest do begin:
    Using ReductionW, find the reduction $(H', w_{H'})$ of (H, w_H);
    Find a cycle Γ' in H' with the smallest number of vertices;
    Set $\delta \leftarrow \min_{v \in V(\Gamma')} w_{H'}(v)$;
    Set $w_{H'}(v) \leftarrow w_{H'}(v) - \delta$ **for every** $v \in V(\Gamma')$;
    Let $X = \{v \in V(\Gamma') \;:\; w_{H'}(v) = 0\}$;
    Remove X (with all incident edges) from H';
    $(H, w_H) \leftarrow (H', w_{H'})$;
    $F \leftarrow X \cup F$;
end.

It is not hard to see that MiniWCycle computes a vertex feedback set of G. We now analyze the algorithm. The analysis uses techniques similar to those used in [Ho82], [Ho83], and [KVY93]. We note that the theorem can also be proved using the Local Ratio Theorem of Bar-Yehuda and Even [BE85].

Theorem 15 *The performance ratio of algorithm* MiniWCycle *is at most* $4 \log_2 |V(G)|$.

Proof. Given a vertex feedback set F of (G, w), let $\mathbf{x} = [x_v]_{v \in V(G)}$ be the indicator vector of F, namely, $x_v = 1$ if $v \in F$ and $x_v = 0$ otherwise. We denote by $\mathcal{C}$ the set of cycles in G. The problem of finding a minimum-weight vertex feedback set of (G, w) can be formulated in terms of $\mathbf{x}$ by an integer programming problem as follows:

$$\begin{array}{c} \text{minimize} \quad \sum_{v \in V(G)} w(v) \cdot x_v \\ \text{ranging over all nonnegative integer vectors } \mathbf{x} = [x_v]_{v \in V(G)} \text{ such that} \\ \displaystyle\sum_{v \in V(\Gamma)} x_v \;\geq\; 1 \quad \text{for every} \quad \Gamma \in \mathcal{C} \;. \end{array} \tag{3}$$

Let $\mathcal{C}_v$ denote the set of cycles passing through vertex v in G and consider the following integer programming *packing* problem:

$$\begin{array}{c} \text{maximize} \quad \sum_{\Gamma \in \mathcal{C}} y_\Gamma \\ \text{ranging over all nonnegative integer vectors } \mathbf{y} = [y_\Gamma]_{\Gamma \in \mathcal{C}} \text{ such that} \\ \displaystyle\sum_{\Gamma \in \mathcal{C}_v} y_\Gamma \;\leq\; w(v) \quad \text{for every} \quad v \in V \;. \end{array} \tag{4}$$

Clearly, the linear relaxation of (4) is the dual of the linear relaxation of (3), with y_Γ, $\Gamma \in \mathcal{C}$, being the dual variables.

Let $(H', w_{H'})$ be a reduction graph computed at some iteration of algorithm MiniWCycle. Then, for each cycle $\Gamma' \in H'$, we associate a unique cycle $\Gamma \in G$ as follows: If all vertices in $V(\Gamma')$ belong to G, then $\Gamma = \Gamma'$. Otherwise, we "unfold" the reduction steps in backward order, i.e., from the current iteration back to the first iteration in ReductionW: In each such step we add to Γ' chains of linkpoints (connecting vertices in Γ') that were deleted by algorithm ReductionW. When this process finishes, the cycle Γ' of H' transforms into a cycle Γ of G.

We now show that MiniWCycle can be interpreted as a primal–dual algorithm. We first show that it computes a dual feasible solution for (4) with a certain maximality property. The initial dual feasible solution is the one in which all the dual variables y_Γ are zero.

Let Γ'_i be a cycle chosen at iteration i of MiniWCycle and let Γ_i be the associated cycle in G. We may view the computation of iteration i of MiniWCycle as setting the value of the dual variable y_{Γ_i} to the weight δ of a lightest vertex in $V(\Gamma'_i)$. The updated weight $w_{H'}(v)$ of every $v \in V(\Gamma'_i)$ is precisely the slack of the dual constraint

$$\sum_{\Gamma \in \mathcal{C}_v} y_\Gamma \;\le\; w(v) \tag{5}$$

that corresponds to v.

It is clear that by the choice of δ, the values of the dual variables y_Γ at the end of iteration i of MiniWCycle satisfy the dual constraints (5) corresponding to vertices $v \in V(\Gamma'_i)$. It thus follows that the dual constraints hold for all vertices $v \in V(H')$ at iteration i.

Let v be a vertex that was removed from H to obtain H' in iteration i of MiniWCycle. It remains to show that the dual constraint (5) corresponding to such a vertex holds in each iteration j of the algorithm for every $j \geq i$.

We show this by backward induction on j. By the previous discussion it follows that the constraints corresponding to vertices that exist in the last iteration all hold. Suppose now that the dual constraints corresponding to vertices in $V(H')$ in iteration j are not violated. We show that the dual constraints corresponding to vertices in $V(H) - V(H')$ in that iteration are also not violated. Let c be a chain of linkpoints in H in iteration j. Algorithm ReductionW deletes all vertices in c except for a representative v which has the minimum weight in c. We now observe that the set of cycles that pass through a vertex of c is the same for all vertices in c. This implies that if the dual constraint corresponding to v is not violated, then the dual constraints corresponding to any vertex in c is also not violated.

The algorithm essentially constructs a primal solution $\mathbf{x}$ from the dual solution $\mathbf{y}$: It selects into the vertex feedback set all vertices for which: (i) the corresponding dual constraints are tight; and (ii) in the iteration the constraint first became tight, the corresponding vertex belonged to the graph. As stated earlier, this construction yields a feasible solution.

Let $\mathbf{x}^* = [x_v^*]_{v\in V(G)}$ and $\mathbf{y}^* = [y_\Gamma^*]_{\Gamma\in\mathcal{C}}$ denote the optimal primal and dual fractional solutions, respectively. It follows from the duality Theorem that

$$\sum_{v\in V(G)} w(v)\cdot x_v \;\geq\; \sum_{v\in V(G)} w(v)\cdot x_v^* \;=\; \sum_{\Gamma\in\mathcal{C}} y_\Gamma^* \;\geq\; \sum_{\Gamma\in\mathcal{C}} y_\Gamma \,. \tag{6}$$

Hence, to prove the theorem, it suffices to bound the ratio between the LHS and the RHS of (6). First note that $y_\Gamma \neq 0$ only for cycles Γ in G that are associated with cycles Γ' that were chosen at some iteration of MINIWCYCLE. By the above construction of $\mathbf{x}$, it is clear that the dual variable y_Γ of each such cycle Γ contributes its value to at most $V(\Gamma')$ vertices. Hence,

$$\sum_{v\in V(G)} w_v\cdot x_v \;\leq\; \sum_{v\in V(G)}\sum_{\Gamma\in\mathcal{C}_v} y_\Gamma \;\leq\; \sum_{\Gamma\in\mathcal{C}} y_\Gamma\cdot|V(\Gamma')|\,.$$

Now, in each iteration, the graph H' is a branchy graph. Therefore, by arguments similar to those appearing in the proof of Lemma 4, we have $|V(\Gamma')| \leq 4\log_2 n$. Hence the theorem is proved. □

Proposition 16 *For planar graphs, the weighted performance ratio of* MINIWCYCLE *is at most 10.*

Proof. We first notice that algorithm REDUCTIONW preserves planarity, and therefore at each iteration of algorithm MINIWCYCLE we remain with a planar graph.

We claim that every rich planar graph G must contain a face of length at most 5. Assume the contrary. By summing up the length of all the faces, we get that $2|E| \geq 6|Z|$, where Z denotes the set of faces of G. By Euler's formula,

$$|E| - |V| + 2 = |Z|$$

Hence, $2|E| \leq 3|V| - 6$. However, since the degree of each vertex is at least 3, we get that $2|E| \geq 3|V|$ which is a contradiction.

Furthermore, this implies that a branchy planar graph must contain a cycle of length at most 10. □

3.2 Low-degree graphs

The algorithm presented in this section is based on the following generalization of Lemma 7 to branchy graphs.

Lemma 17 *Let G be a branchy graph. Then, for every vertex feedback set F of G,*

$$|V(G)| \;\leq\; 2\Delta^2(G)\cdot|F|$$

Proof. Let F be a vertex feedback set of G. Without loss of generality we can assume that F contains only branchpoints. Let G' be the reduction of G. Clearly, F is also a vertex feedback set of G'. Hence, By Lemma 7 we get that,

$$|V(G')| \le (\Delta(G')+1) \cdot |F| - 2 .$$

Notice that $\Delta(G') \le \Delta(G)$. Since G is a branchy graph,

$$|V(G)| \le \frac{(\Delta(G)+2) \cdot |V(G')|}{2}$$

Hence,

$$|V(G)| \le \frac{(\Delta(G)+2) \cdot (\Delta(G)+1) \cdot |F|}{2} - 2 \le 2\Delta^2(G) \cdot |F|$$

□

We now present a weighted greedy algorithm for finding a feedback set in a graph G.

Algorithm WGreedy (*Input:* (G,w); *Output:* vertex feedback set F of (G,w));

$F \leftarrow \emptyset$; $i \leftarrow 1$; $(H, w_H) \leftarrow (G, w)$;
while H **is not a forest do begin:**
  using ReductionW, find the reduction $(H'_i, w_{H'_i})$ **of** (H, w_H);
  $\alpha_i \leftarrow \min_{v \in V(H'_i)} w_{H'_i}(v)$;
  $U_i \leftarrow \{u \in V(H'_i) \mid w_{H'_i}(u) = \alpha_i\}$;
  $F \leftarrow F \cup U_i$;
  remove U_i **from** H'_i **with its incident edges;**
  $(H, w_H) \leftarrow (H'_i, w_{H'_i})$;
  $i \leftarrow i+1$;
end.

For a subset $S \subseteq V$, let $w(S)$ denote the sum of weights of the vertices in S. We now prove the following theorem.

Theorem 18 *Let G be a branchy graph. Denote by F the vertex feedback set computed by algorithm* WGreedy, *and by F^* a minimum-weight vertex feedback set in G. Then, $w(F) \le 2\Delta^2(G) \cdot w(F^*)$.*

Proof. Assume that the number of iterations the *while* loop is executed in algorithm WGreedy is p. We define the following weight functions $w_1, \ldots, w_p$ on $V(G)$. The weight function w_i is defined, for $1 \le i \le p$, as follows:

$$\text{For all } v \in V(G): \quad w_i(v) = \begin{cases} \alpha_i - \alpha_{i-1} & \text{if } v \in V(H'_i) \\ 0 & \text{otherwise} \end{cases} ,$$

where $\alpha_0 = 0$.

For a subset S, let $w_i(S)$ denote the sum of weights of the vertices in S, where the weight function is w_i. Clearly,

$$w(F) = \sum_{i=1}^{p} w(U_i) = \sum_{i=1}^{p} w_i(F)$$

Suppose that at one of the reduction steps of algorithm WGreedy, a chain c of equal weight linkpoints was reduced to a single vertex, say, v. Suppose further that v was added to F. If F^* also contains a vertex from the chain c, then without loss of generality, we can assume that this vertex is also v.

Let $u \in F^*$. Obviously, $u \in H'_1$. We claim that if $u \notin F$, then $u \in H'_i$ for all $i = 1, 2, \ldots, p$. Assume this is not the case. Then, with respect to the order in which vertices entered F in algorithm WGreedy, let u be the first vertex such that $u \in F$, $u \notin F^*$, and u was removed from the graph in a reduction step. This means that u was at the time of its removal a linkpoint that had an adjacent linkpoint u' with smaller weight. But then, by exchanging u for u' in F^*, we obtain a vertex feedback set which has smaller weight, contradicting the optimality of F^*. Hence, for a vertex $u \in F^*$, $w(u) \geq \sum_{i=1}^{p} w_i(u)$. Therefore,

$$w(F^*) \geq \sum_{i=1}^{p} w_i(F^*)$$

Notice that in the graph H'_i, the weight function w_i assigns the same weight to all vertices. Hence, by Lemma 17, we have that $w_i(F) \leq 2\Delta^2(H'_i) \cdot w_i(F^*)$ for all $i = 1, 2, \ldots, p$. Since $\Delta(H'_i) \leq \Delta(G)$ for all i, the theorem follows. □

It follows from Lemma 14 that the performance ratio of algorithm WGreedy for (G, w) is at most $2\Delta^2(G)$ for *any* graph G.

4 Graphs with Blackout Vertices

We now consider a generalization of the unweighted vertex feedback set problem where we mark each vertex of a graph as either an *allowed vertex* or a *blackout vertex*. In such graphs, vertex feedback sets cannot contain any blackout vertices. The motivation for dealing with this modified problem is clarified in the next section where we use the algorithms developed herein to reduce the computational complexity of Bayesian inference. Note that a vertex feedback set can be found in a graph G with blackout vertices if and only if every cycle in G contains at least one allowed vertex. A graph G with the latter property will be called a *valid graph*. Every subgraph of a valid graph is valid.

The vertex feedback set problem for graphs with blackout vertices is in effect a special case of the weighted vertex feedback set problem. Indeed, given a valid graph G, we assign weight $|V(G)|$ to each blackout vertex and unit weight to all other vertices. It is clear that, with this choice of weights, there is no point in choosing a blackout vertex to a vertex feedback set. Furthermore, setting a large enough weight (say, $4|V(G)|\log_2|V(G)|$) to the blackout vertices in G, we can apply MiniWCycle to find a vertex feedback set of $(G,1)$ and the upper bound on the performance ratio stated in Theorem 15 will still hold. We now show that this bound can be improved, and that the same bounds obtained for the unweighted case can be achieved here as well.

We denote the set of allowed vertices in G by $A(G)$ and the set of blackout vertices by $B(G)$. Let $\Delta_a(G)$ denote the maximum degree of an allowed vertex in G. We first generalize Lemma 7.

Lemma 19 *Let G be a valid rich graph. Then, for every vertex feedback set F of G,*

$$|V(G)| \leq (\Delta_a(G)+1)\,|F| - 2 .$$

Proof. Replace each occurrence of $\Delta(G)$ in the proof of Lemma 7 by $\Delta_a(G)$. □

We next modify several of the definitions of the previous sections. Let G be a valid graph. A *2-3-subgraph* of G is a subgraph H of G such that the degree in H of every vertex in $A(G)$ is either 2 or 3. The degree of a vertex belonging to $B(G)$ in H is not restricted. Similarly, a *maximal 2-3-subgraph* of G is a 2-3-subgraph which is not a subgraph of any other 2-3-subgraph of G.

A linkpoint v in a 2-3-subgraph H is called a *critical linkpoint* if v is an allowed vertex, and there is a cycle Γ in G such that $V(\Gamma)\cap V(H)\subseteq\{v\}\cup B(G)$. We refer to such a cycle Γ in G as a *witness cycle of v*.

A cycle in a valid graph G is *branchpoint-free* if it does not pass through any allowed branchpoints; that is, a branchpoint-free cycle passes only through linkpoints and blackout vertices of G. A reduction graph of a valid graph G is not necessarily valid, since the reduction process may generate a cycle consisting of blackout vertices only. However, if we assume G to have no endpoints and no branchpoint-free cycles, then the following can be easily verified.

Lemma 20 *Let G be a valid graph without any branchpoint-free cycles and with no endpoints. Then, the reduction G' of G is valid and $\mu(G',1)=\mu(G,1)$.*

The following algorithm achieves an unweighted performance ratio of less than 4.

Algorithm ResSubG-2-3 (*Input:* valid graph G; *Output:* vertex feedback set F of G);

if G **is a forest then**
  $F \leftarrow \emptyset$;
else begin:
  Using DFS, find a maximal 2-3-subgraph H **of** G;
  Using BFS, find the set X **of critical linkpoints in** H;
  Find a set W **that covers all branchpoint-free cycles of** H **which**
    are not covered by X;
  Set Y **to be the set of branchpoints in** H;
  $F \leftarrow X \cup Y \cup W$;
end.

We now elaborate on how the set W is computed. Let H_b be the subgraph of H induced by linkpoints and blackout vertices. For every isolated cycle in H_b, we arbitrarily choose an allowed linkpoint from that cycle to W. Next, we replace each maximal (with respect to containment) chain of allowed linkpoints in H_b by an edge, resulting in a graph H_b^*. We assign a unit cost to all edges corresponding to a chain of linkpoints, and a zero cost to all other edges, and compute a minimum-cost spanning forest T of H_b^*. We now add to W one linkpoint from each chain of allowed linkpoints in H_b that corresponds to an edge in $H_b^* - T$.

The analysis of ResSubG-2-3 is based on the following lemmas.

Lemma 21 *Let H be a maximal 2-3-subgraph of a valid graph G and let Γ be a simple cycle in G. Then, one of the following holds:*

(a) Γ is a witness cycle of some critical linkpoint of H, or —

(b) Γ passes through some allowed branchpoint of H, or —

(c) Γ is a cycle in H that consists only of blackout vertices and linkpoints.

Proof. The proof follows along the lines of that of Lemma 9: We assume that Γ is a cycle in G and that neither of (a)–(c) holds. We then distinguish between two cases: 1) Γ does not intersect with H at all, or it intersects with H only in blackout vertices; and 2) Γ intersects with H only in blackout vertices or linkpoints of H. The proof that both cases lead to a contradiction is the same as the one in Lemma 9. □

Lemma 22 *Let H be a maximal 2-3-subgraph of G and let Γ_1 and Γ_2 be witness cycles in G of two distinct critical linkpoints in H. Then $V(\Gamma_1) \cap V(\Gamma_2) \subseteq B(G)$.*

The proof of the lemma is similar to that of Lemma 10.

Proposition 23 *For every graph G, the set F computed by* ResSubG-2-3 *is a vertex feedback set of G.*

Proof. Let Γ be a cycle in G. We follow the three cases of Lemma 21 to show that $V(\Gamma) \cap F \neq \emptyset$. Cases (a) and (b) are proved as in Proposition 11. Hence, it remains to check case (c), namely, where Γ passes through either blackout vertices or linkpoints of H.

When $V(\Gamma)$ contains a critical linkpoint, then ResSubG-2-3 selects that linkpoint into the vertex feedback set F and when Γ is an isolated component in H, then ResSubG-2-3 selects an arbitrary allowed representative vertex into F. We now show that W covers all other cycles Γ that pass only through either blackout vertices or linkpoints in H.

Assume the contrary, and let Γ be such a cycle which is not covered by W. In the construction of H_b^*, each chain of allowed linkpoints in Γ is replaced by an edge. Since W does not cover Γ, all edges in H_b^* corresponding to chains of linkpoints in Γ were necessarily chosen to the minimum-cost spanning forest T. Since T is a spanning forest, Γ must contain at least one edge that connects two blackout vertices. However, in that case, the cost of T can easily be reduced, which contradicts the claim that T is a minimum-cost spanning forest. □

Lemma 24 *Let F be the vertex feedback set computed by* ResSubG-2-3 *for a valid graph G which is not a forest. Then,*

$$|F| \leq 4\,\mu(G,1) - 2\,.$$

Proof. Let H, X, Y and W be as in ResSubG-2-3. The case $\mu(G,1) = 1$ is identical to that in the proof of Lemma 12. Hence, we assume from now on that $\mu(G,1) \geq 2$.

Recall that by Lemma 22, witness cycles of critical linkpoints can share only blackout vertices. Therefore, the minimum number of vertices needed to cover such cycles is $|X|$.

Let $\{\Gamma_j^*\}$ be the set of branchpoint-free cycles in H that do not contain any critical linkpoints of H. These cycles may intersect with any witness cycle of a critical linkpoint only at blackout vertices. We now claim that any smallest set W' of vertices of $V(H)$ that intersects with the vertex set of each Γ_j^* must be of size $|W|$. To see this, note that W' contains allowed linkpoints only. If we remove from H_b^* all the edges that correspond to linkpoints belonging to W', then we clearly end up with a forest. By construction, the minimum number of edges (or allowed linkpoints), needed to be removed from H_b^* so as to make it into a forest, is $|W|$.

Now, such a set W' cannot possibly have any vertices in common with X. Therefore,

$$\mu(G,1) \geq |X| + |W'| = |X| + |W|\,.$$

On the other hand, we recall that $|F| = |X| + |Y| + |W|$.

We distinguish between the following two cases.

Case 1: $|Y| \le 2|X| + 2|W|$. Here we have,

$$|C| \;=\; |X| + |Y| + |W| \;\le\; 3|X| + 3|W| \;\le\; 3\mu(G,1) \;\le\; 4\mu(G,1) - 2\;.$$

Case 2: $|Y| > 2|X| + 2|W|$. Let F^* be a vertex feedback set of G of size $\mu(G,1)$ and let W' be a smallest subset of F^* that intersects with the vertex set of each Γ_j^*. Clearly, W' consists of allowed linkpoints of H, and, as we showed earlier in this proof, $|W'| = |W|$. Let H_1 be the subgraph of H obtained by removing all critical linkpoints of H and all linkpoints in W'. As we did in the proof of Lemma 12, with each deleted linkpoint, we also remove recursively all resulting endpoints from H while obtaining H_1. Thus, a deletion of a linkpoint from H can decrease the number of branchpoints by 2 at most. Hence, the number of branchpoints left in H_1 is at least $|Y| - 2|X| - 2|W| > 0$. Furthermore, the graph H_1 does not contain any branchpoint-free cycles or endpoints. So, by Lemma 20, the reduction H_1' of H_1 is a valid graph and $\mu(H_1',1) = \mu(H_1,1)$.

Let H_2 be a valid graph obtained by removing all singleton components from H_1'. By Lemma 2, the graph H_2 is a rich graph with at least $|Y| - 2|X| - 2|W|$ branchpoints. Hence, by Lemma 19,

$$|Y| - 2|X| - 2|W| \;\le\; 4\mu(H_2,1) - 2 \;\le\; 4\mu(H_1',1) - 2 \;=\; 4\mu(H_1,1) - 2\;.$$

Therefore,

$$\begin{aligned} |F| &\;=\; |X| + |Y| + |W| \\ &\;\le\; 4|X| + 4|W| + |Y| - 2|X| - 2|W| \\ &\;\le\; 4(|X| + |W| + \mu(H_1,1)) - 2\;. \end{aligned} \tag{7}$$

Recalling the definition of W' as a subset of a smallest vertex feedback set F^* of G, and noting that F^* must intersect with the vertex set of any cycle in H_1, we have

$$|X| + |W| + \mu(H_1,1) \;=\; |X| + |W'| + \mu(H_1,1) \;\le\; |F^*| \;=\; \mu(G,1)\;.$$

The claim now follows by plugging the last inequality into (7). □

Theorem 25 *The unweighted performance ratio of* ResSubG-2-3 *is at most* $4 - (2/|V(G)|)$.

Proof. This follows immediately from Lemma 24. □

5 The Loop Cutset Problem and its Application

In this section we consider a variant of the WVFS Problem for directed graphs. The underlying graph of a directed graph D is the undirected graph formed by ignoring the directions of the edges in D. 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 Γ if the two edges adjacent to v in Γ 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 that loop Γ is called an *allowed vertex with respect to* Γ. 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. Our problem is to find a minimum-weight loop cutset of a given directed graph D and a weight function w. We denote by $\mu(D, w)$ the sum of weights of the vertices in such a loop cutset. Greedy approaches to the loop cutset problem have been suggested by [SC90] and [St90]. Both methods can be shown to have a performance ratio as bad as $\Omega(n/4)$ in certain planar graphs [St90]. An application of approximation algorithms to the loop cutset problem in the area of Bayesian inference is described later in this section.

The approach we take is to reduce the weighted loop cutset problem to the weighted vertex feedback set problem solved in the previous section. Given a weighted directed graph (D, w), we define the *splitting* weighted undirected graph (D_s, w_s) as follows. Split each vertex v in D into two vertices v_{in} and v_{out} in D_s such that all incoming edges to v become undirected incident edges with v_{in}, and all outgoing edges from v become undirected incident edges with v_{out}. In addition, we connect v_{in} and v_{out} by an undirected edge. Set $w_s(v_{\text{in}}) = \infty$ and $w_s(v_{\text{out}}) = w(v)$. For a set of vertices X in D_s, we define $\psi(X)$ as the set obtained by replacing each vertex v_{in} or v_{out} in X by the respective vertex v in D from which these vertices originated.

Our algorithm can now be easily stated.

Algorithm LoopCutset (*Input:* (D, w); *Output:* loop cutset F of (D, w));

Construct (D_s, w_s);
Apply MiniWCycle on (D_s, w_s) to obtain a vertex feedback set X;
$F \leftarrow \psi(X)$.

Note that each loop in D is associated with a unique cycle in D_s, and vice-versa, in a straightforward manner. Let $I(\Gamma)$ denote the loop image of a cycle Γ in D_s, and $I^{-1}(K)$ denote the cycle image of a loop K in D. It is clear that the mapping I is $1-1$ and onto.

The next lemma shows that algorithm LoopCutset outputs a loop cutset of (D, w).

Lemma 26 *Let (D, w) be a directed weighted graph and (D_s, w_s) be its splitting graph. Then: (i) If F is a vertex feedback set of (D_s, w_s) having finite weight, then $\psi(F)$ is a loop cutset of (D, w), and $w_s(F) = w(\psi(F))$. (ii) If U is loop cutset of D, then the set U_s obtained from U by replacing each vertex $v \in U$ by vertex $v_{out} \in D_s$ is a vertex feedback set of D_s, and $w(U) = w_s(U_s)$.*

Proof. We prove (i). The proof of (ii) is similar. Let Γ be a loop in D. To prove the lemma we show that an allowed vertex with respect to Γ belongs to $\psi(F)$. Let $I^{-1}(\Gamma)$ be the unique cycle image of Γ in D_s. Since F is a cycle cover of D_s having finite weight, there must be a vertex $v_{\text{out}} \in F$ in $I^{-1}(\Gamma)$. Now, it is clear that vertex $v \in \Gamma$ from which v_{out} originated is an allowed vertex with respect to Γ as needed. To complete the proof, by the finiteness of $w_s(F)$, we must have $w_s(F) = w(\psi(F))$, since $w_s(v_{\text{out}}) = w(v)$ for each vertex in F. $\square$

It follows from Lemma 26 that $\mu(D, w) = \mu(D_s, w_s)$. In addition, due to Theorem 15 applied to the graph D_s, and since the number of vertices in D_s is twice the number of vertices in D, we get the following bound on the performance ratio of algorithm LoopCutset.

Theorem 27 *The performance ratio of* LoopCutset *is at most* $4\log_2(2|V(D)|)$.

For planar graphs we have:

Theorem 28 *The performance ratio of* LoopCutset *is at most 10 for planar graphs.*

Proof. Since the splitting graph of a planar graph is planar we have,

$$w(\psi(F)) = w_s(F) \leq 10\,\mu(D_s, w_s)$$

where the equality is due to Lemma 26 and the inequality is due to Lemma 16. Since $\mu(D_s, w_s) = \mu(D, w)$, the claim is proved. $\square$

We now show that in the unweighted loop cutset problem, we can achieve a performance ratio better than 4. In this case, for each vertex $v \in D$, the weight of $v_{\text{in}} \in D_s$ is one unit, and the weight of $v_{\text{out}} \in D_s$ is ∞. This is exactly the case considered in the previous section, since vertices with infinite weight in D_s can be treated as blackout vertices. We can therefore apply ResSubG-2-3 in the LoopCutset algorithm instead of applying MiniWCycle and obtain the following improved performance ratio.

Theorem 29 *When using* ResSubG-2-3*, the unweighted performance ratio of* LoopCutset *is at most* $4 - (2/|V(D)|)$.

Proof. We have,

$$w(\psi(F)) = w_s(F) \leq 4\mu(D_s, w_s) - 2$$

where the equality is due to Lemma 26, and the inequality is due to Lemma 24. Since $\mu(D_s, w_s) = \mu(D, w) \le n$, the claim is proved. $\square$

5.1 An application

We conclude this section with an application of approximation algorithms for the loop cutset problem.

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:

$$P(u_1, \ldots, u_n) = \prod_{i=1}^{n} P(u_i \mid u_{i_1}, \ldots, u_{i_{j(i)}}) \tag{8}$$

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

It is worth noting that Bayesian networks are useful knowledge representation schemes for many artificial intelligence tasks. Bayesian networks allow a wide spectrum of independence assumptions to be considered by a model builder so that a practical balance can be established between computational needs and adequacy of conclusions. For a complete exploration of this subject see [Pe88].

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 $i = 1, \ldots, n$. In principle, such computations are straightforward because each Bayesian network defines the joint probability distribution $P(u_1, \ldots, u_n)$ from which all conditional probabilities can be computed by dividing the appropriate sums. However, such computations are inefficient both in time and space unless they use conditional independence assumptions defined by Eq. (8). We shall see next how our approximation algorithms for the loop cutset problem reduce the computations needed for solving 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* by Z.

Verma and Pearl [VP88] have 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 $\{t_1, \ldots, t_m\}$, 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 [GP90] proved a converse to this theorem. Both results are presented and extended in [GVP90].

Using the close relationship between blocked trails and conditional independence, Kim and Pearl [KP83] 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. UPDATE-TREE views each vertex as a processor that repeatedly sends messages to each of its neighboring vertices. When equilibrium is reached, each vertex i contains the conditional probability distribution $P(u_i \mid v_1 = \mathbf{v}_1, \ldots, v_l = \mathbf{v}_l)$. The computations reach equilibrium regardless of the order of execution in time proportional to the length of the longest trail in the network.

Pearl [Pe86] solved the updating problem on any Bayesian network as follows. First, a set of vertices S is selected, such that any two vertices in the network are connected by at most one *active* trail in $S \cup Z$, 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. Note that according to the definition of active trails, the set S in Pearl's algorithm is a loop cutset of the Bayesian network. In this paper we have developed approximation algorithms for finding S.

When the domain size of the variables varies, then UPDATE-TREE is called a number of times which is bounded from above by 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 as the weight of a vertex, then solving the weighted loop cutset problem with these weights optimizes Pearl's updating algorithm in the case where the domain sizes are allowed to vary.

6 Discussion

It is useful to relate the vertex feedback set problem with the vertex cover problem in order to establish lower bounds on the performance ratios attainable for the vertex feedback set problem. A vertex cover of an undirected graph is a subset of the vertex set that intersects with each edge in the graph. The vertex cover problem is to find a minimum weight vertex cover of a given graph. There is a simple polynomial reduction from the vertex cover problem to the vertex feedback set problem: Given a graph G, we extend G to a graph H by adding a vertex v_e for each edge $e \in E(G)$, and connecting v_e with the vertices in G with which e is incident in G. It is easy to verify that there always exists a minimum vertex feedback set in H whose vertices are all in $V(G)$ and this vertex feedback set is also a minimum vertex cover of G. In essence, this reduction replaces each edge in G with a cycle in H, thus transforming any vertex cover of G to a vertex feedback set of H.

Due to this reduction, it follows that the performance ratio obtainable for the vertex feedback set problem cannot be better than the one obtainable for the vertex cover problem. The latter problem has attracted a lot of attention over the years but has so far resisted any approximation algorithm that achieves in general graphs a constant performance ratio less than 2. We note that the above reduction retains planarity. However, for planar graphs, Baker [Ba83] provided a Polynomial Approximation Scheme (PAS) for the vertex cover problem. For the UVFS problem, there are examples showing that 4 is the tightest constant performance ratio of algorithm SubG-2-3. It is an open question whether there exists an algorithm for the cycle cover problem that achieves precisely the performance ratio obtainable for the vertex cover problem.

Another consequence of the above reduction is a lower bound on the unweighted performance ratio of the following greedy algorithm, GreedyCyc, for the vertex feedback set problem. In each iteration, GreedyCyc removes a vertex of maximal degree from the graph, adds it to the vertex feedback set, and removes all endpoints in the graph. A similar greedy algorithm for the vertex cover problem is presented in [Jo74] and in [Lo75]. The latter algorithm was shown to have an unweighted performance ratio no better than $\Omega(\log |V(G)|)$ [Jo74]. Due to the reduction to the cycle cover problem, the same lower bound holds also for GreedyCyc, as demonstrated by the graphs of [Jo74]. A tight upper bound on the worst-case performance ratio of GreedyCyc is unknown.

Finally, one should notice that the following heuristics may improve the performance ratios of our algorithms. For example, in each iteration MiniWCycle chooses to place in the cover all zero-weight vertices found on the smallest cycle. This choice might be rather poor especially if many weights are equal. It may be useful in this case to perturb the weights of the vertices before running the algorithm. Similarly, in algorithm SubG-2-3, there is no point in taking blindly all branchpoints of H. An appropriate heuristic here may be to pick the branchpoints one by one in decreasing order of residual degrees. Furthermore, the subgraph H itself should be constructed such that it contains as many high degree vertices as possible.

Acknowledgement

We would like to thank David Johnson for bringing [EP62] to our attention, and Samir Khuller for helpful discussions.

References

[BE85] Bar-Yehuda R. and Even S., *A local-ratio theorem for approximating the weighted vertex cover problem, Annals of Discrete Mathematics,* 25 (1985), 27–46.

[Ba83] Baker B. S., *Approximation algorithms for NP-complete problems on planar graphs, Proceedings 24th IEEE Symposium on Foundations of Computer Science*, 1983, 265-273.

[DP87] Dechter R. and Pearl J., *The cycle cutset method for improving search performance in AI, Proceedings 3rd IEEE on AI Applications,* Orlando, 1987.

[DP88] Dechter R. and Pearl J., *Network-based heuristics for constraint satisfaction problems, Artificial Intelligence,* 34 (1988), 1–38.

[DP90] Dechter R. and Pearl J., *Enhancement schemes for constraint processing: backjumping, learning, and cutset decomposition, Artificial Intelligence,* 41 (1990), 273–312.

[EP62] Erdős P. and Pósa L., *On the maximal number of disjoint circuits of a graph, Publ. Math Debrecen*, 9 (1962), 3–12.

[EP64] Erdős P. and Pósa L., *On the independent circuits contained in a graph, Canad. J. Math,* 17 (1964), 347–352.

[GJ79] Garey M.R. and Johnson D.S., *Computers and Intractability: A Guide to the Theory of NP-completeness,* W. H. Freeman, San Francisco, California, 1979.

[GP90] Geiger, D. and Pearl, J., *On the logic of causal models,* In *Uncertainty in Artificial Intelligence 4,* Eds. Shachter R.D., Levitt T.S., Kanal L.N., and Lemmer J.F., North-Holland, New York, 1990, 3–14.

[GVP90] Geiger, D., Verma, T.S., and Pearl, J., *Identifying independence in Bayesian networks, Networks,* 20 (1990), 507–534.

[Ho82] Hochbaum D.S., *Approximation algorithms for set covering and vertex covering problems, SIAM J. Computing,* 11 (1982), 555–556.

[Ho83] Hochbaum D.S., *Efficient bounds for the stable set, vertex cover, and set packing problems, Discrete Applied Math,* 6 (1983), 243–254.

[IR78] Itai A. and Rodeh M., *Finding a minimum circuit in a graph, SIAM J. Computing,* 7 (1978), 413–423.

[Jo74] Johnson D.S., *Approximation algorithms for combinatorical problems, J. Comput. Sys. Sciences,* 9 (1974), 256–278.

[KVY93] Khuller S., Vishkin U., and Young, N., *A primal-dual parallel approximation technique applied to weighted set and vertex cover,* In *Proceedings of Integer Programming and Combinatorial Optimization,* 1993, 333–341.

[KP83] Kim H. and Pearl J., *A computational model for combined causal and diagnostic reasoning in inference systems,* In *Proceedings of the Eighth IJCAI*, Morgan-Kaufmann, San Mateo, California, 1983, 190–193.

[MS81] Monien B.and Schulz R., *Four approximation algorithms for the feedback vertex set problem,* In *Proceedings of the 7th Conference on Graph Theoretic Concepts of Computer Science*, 1981, 315-326.

[Lo75] Lovász L., *On the ratio of optimal integral and fractional covers, Discrete Math.,* 13 (1975), 383–390.

[Pe86] Pearl, J., *Fusion, propagation and structuring in belief networks, Artificial Intelligence,* 29:3 (1986), 241–288.

[Pe88] Pearl, J., *Probabilistic reasoning in intelligent systems: Networks of plausible inference.* Morgan Kaufmann, San Mateo, California, 1988.

[Si67] Simonovits M., *A new proof and generalizations of a theorem by Erdős and Pósa on graphs without $k+1$ independent circuits, Acta Mathematica Academiae Hungaricae Tomus,* 18 (1967), 191–206.

[St90] Stillman, J., *On heuristics for finding loop cutsets in multiply connected belief networks,* In *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*, Cambridge, Massachusetts, 1990, 265–272.

[SC90] Suermondt H.J. and Cooper G.F., *Probabilistic inference in multiply connected belief networks using loop cutsets, Int. J. Approx. Reasoning*, 4 (1990), 283–306.

[VP88] Verma, T. and Pearl, J., *Causal networks: Semantics and expressiveness,* In *Proceedings of Fourth Workshop on Uncertainty in Artificial Intelligence,* Minneapolis, Minnesota (published by the Association for Uncertainty in Artificial Intelligence, Mountain View, California), 1988, 352–359.

[Vo68] Voss H.J., *Some properties of graphs containing k independent circuits, Proc. Colloq. Tihany*, Academic Press, New York, 1968, 321–334.