Markov-Chain-Based Heuristics for the Minimum Feedback Vertex Set Problem

Mile Lemaić and Ewald Speckenmeyer
{lemaic,esp}@informatik.uni-koeln.de

Universität zu Köln
Institut für Informatik
Pohligstr. 1
D-50969 Köln, Germany

Abstract

Let $G = (V, A)$ be a directed graph. A vertex set $F \subseteq V$ is called feedback vertex set (FVS) if its removal from G results in an acyclic graph. Because determining a minimum cardinality FVS is known to be NP hard, [15], one is interested in designing fast approximation algorithms determining near-optimum FVSs.

The paper presents deterministic and randomised heuristics based on Markov chains. In this regard, an earlier approximation algorithm developed in [31] is revisited and refined. Experimental results demonstrate the overall performance superiority of our algorithms compared to other algorithms known from literature with respect to both criteria, the sizes of solutions determined, as well as the consumed runtimes.

Keywords: feedback vertex set, digraph, Markov chain

1 Introduction

Let $G = (V, A)$ be a (di)graph with vertex set V and arc set A. A subset $F \subset V$ is called a feedback vertex set (FVS) of G if $G - F$ is acyclic. The **minimum FVS problem** consists of determining a FVS of minimum cardinality. The feedback arc set (FAS) problem is defined similarly: determine a minimum cardinality set $F \subset A$ such that $G - F$ is acyclic. Each of these two problems can be polynomially reduced to the other, [14], so in this paper we will concentrate on the FVS problem. Feedback problems occur as the core of many real world problems from various areas, such as deadlock breaking, [25], program verification, [30], Bayesian inference, [33], et al.

Even if determining a minimum cardinality FVS is known to be NP-hard for both undirected and directed graphs, [15], it is more intricate dealing with directed than with undirected graphs. While for general undirected graphs there exist approximation algorithms with performance ratio of 2, [1, 2], the best known approximation algorithm for directed graphs achieves a performance ratio of $O(\log n \log\log n)$, [7], based on a result from [29].

Besides this challenging problem of designing efficient algorithms that produce FVSs guaranteeing good worst case ratios, one is often more interested in obtaining efficiently FVSs that typically come close to the optimum ones, while sometimes they may be arbitrarily bad. These algorithms rely on exploiting heuristics in order to estimate the suitability of the vertices of the digraph to be included into the FVS, and the vertex with the highest score will be included into the solution each time. Heuristic algorithms of this type can be found in [11, 19, 23]. These algorithms typically produce solutions that are better than those computed by algorithms guaranteeing certain worst case ratio solutions.

The heuristic primarily aims at detecting vertices covering as many cycles as possible, and the highest ranked vertex is considered to be a good choice to be included into the FVS. Counting the number of cycles in digraphs can easily be seen to be #P complete. Therefore, simple heuristics are used, like assigning to each vertex v the product of the number of incoming arcs into v and the number of outgoing arcs from v, see e.g. [11].

Heuristics of this type exploit local information of the digraph only, while cycles however have a global structure. Unsurprisingly, the degree-product heuristic may fail arbitrarily. As an example consider the digraph G from Figure 1. Obviously, $\{v\}$ is an optimum FVS of G. The degree-product heuristic however iteratively selects the vertices $y_1, \ldots, y_r$ with degree-product m^2 if e.g. $r = \lfloor \frac{m^2}{3} \rfloor$. Note that for growing values of m the FVS thus determined consists of the huge majority of all vertices of the digraph.

An approach developed in [31], exploiting information on random walks in G, provided by the theory of Markov chains, has some potential to deliver more reliable information on the suitability of a vertex v to be included into a FVS of G by incorporating also global information into the evaluation.

For the digraph G in Figure 1 the mean return time in random walks in G is smallest for vertex v, indicating v to be the best choice for the FVS, see [31]. Mean return times of random walks are the reciprocal values of the

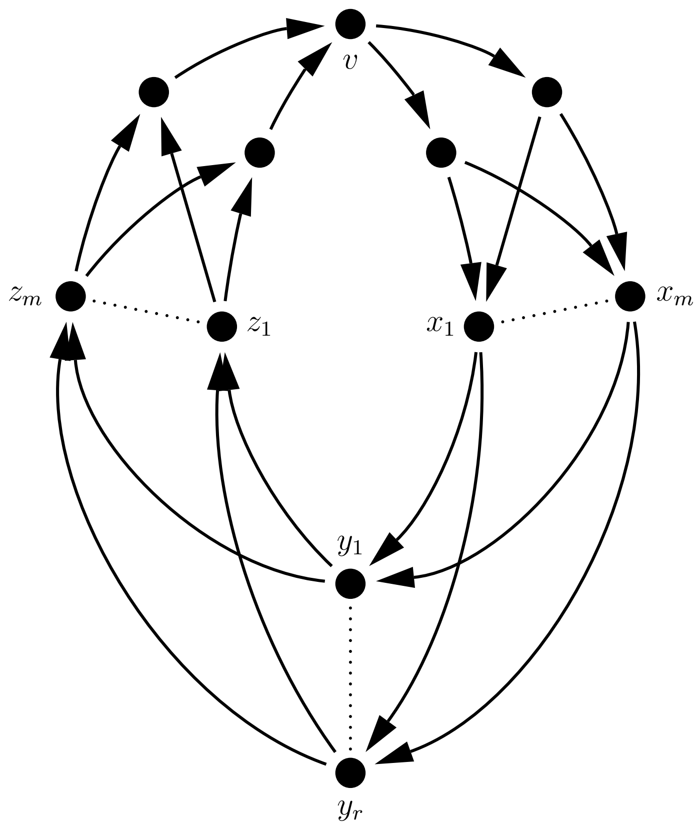

Figure 1: A digraph for which the degree-product heuristic fails arbitrarily to yield the optimum FVS $\{v\}$.

stationary distribution vector of a Markov chain corresponnding to the digraph, while the stationary distribution vector is obtained as the solution of a system of linear equations. Experimental results with an implementation of this Markov-based heuristic, as reported in [31], were very promising. On the other hand, the heuristic can be fooled as was shown by Heusch, [27]. Figure 2 displays an example digraph.

This Markov-based FVS heuristic will be improved in this paper in three ways. The first improvement is very simple but useful. Reversing the directions of all arcs of a digraph G leads to a new digraph G^{-1} with the property that all FVSs of G are FVSs of G^{-1} and vice versa. The mean return times of the vertices in G and G^{-1} however are not the same in general. A mix of both mean return times will be used as heuristic measure for classifying good candidates to be included in a FVS. The above mentioned example digraph in Figure 2 fooling the Markov-based algorithm will be solved correctly with this extension.

The second improvement puts some noise on the canonical transition matrix corresponding to the digraph G. The transition probability from a vertex u to a vertex v in a random walk will no longer be the inverse of the outdegree of u. The best solution from several independent runs for a digraph will be output.

The third improvement performs a limited search for good solutions in the space of minimal FVSs of G in the following way: Suppose F is the current minimal FVS of G. Then, a convenient fraction $F \setminus \widetilde{F}$ of the FVS F is reintegrated into the digraph $G - F$ yielding the digraph $G - \widetilde{F} =: G'$ which is no longer acyclic. After that, a FVS $\widetilde{\widetilde{F}}$ of G' is determined by the mixed version of the Markov-based FVS algorithm. $\widetilde{F} \cup \widetilde{\widetilde{F}}$ not necessarily is a minimal FVS of G from which a minimal FVS F' of G is extracted. This process is iterated a fixed number of times and the best solution is output. Experimental runs with this algorithm always determined solutions for the benchmark digraphs from [11] better than those presented there within a small fraction of runtime consumed by the algorithm applied there. Solving all 40 benchmark digraphs from [11] the approximation algorithm presented there determined FVSs of total size 6941 within 41626 seconds, while the sizes of the FVSs determined by the best approximation algorithm developed here sum up to 6749 consuming only 733 seconds.

Besides these new FVS algorithms we describe and implement fast methods for solving sparse systems of linear equations, as they always occur when dealing with sparse digraphs.

The paper is organised as follows: The rest of this section presents notations while section 2 introduces the basic approximation framework. Section 3 describes the original Markovian heuristic from [31] and also introduces the above mentioned extension involving the reversion of arcs. These heuristics are applied to construct a Greedy Randomised Adaptive Search Procedure (GRASP), [10], as well as the above mentioned local search procedure. These two procedures are described in sections 4 and 5, respectively. Finally, section 6 presents the experimental results while section 7 contains concluding remarks.

1.1 Notation

Throughout this paper, $G = (V, A)$ denotes a directed graph (digraph) with vertex set V and arc set A. Given an arc $(u, v) \in A$, we will say that u is a **predecessor** of v, while v is a **successor** of u. An arc (v, v) is called **loop**. The **indegree** and **outdegree** of a vertex $v \in V$ is denoted by $\mathrm{d}_G^-(v)$ and $\mathrm{d}_G^+(v)$, respectively.

For a vertex $v \in V$ of G, $G - v$ denotes the digraph resulting from G by removing v along with all its incident arcs. This notation is also used for sets F of vertices. Thus, $F \subset V$ is a **feedback vertex set (FVS)** of G if $G - F$ is acyclic. It is said to be **minimal** if there is no proper subset $F' \subsetneqq F$ that is a FVS of G, whereas it is said to be **optimum** if F is of minimum cardinality.

Finally, the **reversed digraph** G^{-1} of G is the digraph

$$G^{-1} = (V, A^{-1}), \quad \text{where} \quad A^{-1} = \{(v, u) : (u, v) \in A\}.$$

2 The basic approximation framework

Recall that cycles occur only within strongly connected components (SCCs) of a digraph. Therefore, in order to break all cycles, the strongly connected components have to be considered. Decomposing a digraph into its SCCs can be done in linear time, [32]. As a consequence, our greedy algorithm follows a general framework, shown in Algorithm 1, that is common in approximation algorithms for determining FVSs.

```
function MinFVS(digraph $G = (V, A)$) do

    vertex set $F$ ← $\emptyset$;
    vertex $v$;

    DoReductions($G$, $F$);

    for all SCCs $G_i$ of $G$ do
        $v$ ← SelectVertex($G_i$);
        $F$ ← $F$ ∪ $\{v\}$ ∪ MinFVS($G_i - v$);
    od;

    return $F$;
od;
```

Algorithm 1: The basic approximation procedure.

The functions `DoReductions(`G`, `F`)` and `SelectVertex(`G`)` have to be specified in order to describe an algorithm based on this framework. `DoReductions(`G`, `F`)` performs five reductions on G and adjusts F accordingly. The reductions, as described in [21], are based on the following reasoning: If G contains a loop (v, v), then v belongs to any FVS of G and has to be removed from G and appended to F. If v has in- or outdegree 0, it obviously cannot be part of any cycle in G, so v is removed. Finally, suppose v has indegree 1 and let u be its unique predecessor. Then, every cycle broken by v is also broken by u. Thus, v can be excluded from further consideration. This is technically done by removing v from G and adding an arc (u, w) for every (former) successor w of v. The case of $\mathrm{d}_G^+(v) = 1$ is handled similarly.

Omitted in Algorithm 1 is the minimisation of the final FVS F of G by removing **redundant** vertices. A vertex $v \in F$ is said to be redundant if $F \setminus \{v\}$ is still a FVS of G. Thus, in order to obtain a minimal FVS, we check the vertices of F for redundancy in reversed order of insertion and remove them in case of being redundant. Intuitively, this seems plausible since the earlier selected vertices are expected to belong more likely to an optimum FVS.

The choice of the function `SelectVertex(`G`)` primarily influences the FVS determined by the approximation algorithms. In [11], the sum, product and maximum of in- and outdegree are chosen as selection criterion, whereas in [23] an approach based on essential cycles was used. Our approach relies on Markov chains and is described next.

3 A Markov chain heuristic

3.1 The Markovian FVS algorithm

The main idea of all our Marekov chain based algorithms is to model a random walk in the digraph $G = (V, A)$ for which a FVS is to be determined. For this purpose we associate with each pair $(v_i, v_j) \in V \times V$ of vertices the probability p_{ij} of moving from vertex v_i to vertex v_j. If there is no arc from v_i to v_j, this transition probability is 0. Otherwise, we set $p_{ij} = \frac{1}{\mathrm{d}_G^+(v_i)}$. Thus, the transition probabilities p_{ij} induce the $|V| \times |V|$ **transition matrix** P_G of G. Obviously, P_G is non-negative and the row elements of each row sum up to 1.

As P_G can be seen as a finite-state Markov chain, the theory of Markov

```
function SelectVertex(digraph $G = (V, A)$) do

    matrix $P$ $\leftarrow$ TransitionMatrix($G$);
    vector $\pi$ $\leftarrow$ StationaryDistribution($P$);

    vertex $v$ $\leftarrow$ vertex corresp. to maximum entry of $\pi$;

    return $v$;
od;
```

Algorithm 2: The vertex selection heuristic of MFVS.

chains yields some interesting properties of P_G, see e.g. [26]. Important for our approach is the result that there is a unique positive vector $\pi = (\pi_i)$ satisfying

$$^t\pi P_G = {}^t\pi \quad \text{and} \quad \sum_{i=1}^{n} \pi_i = 1 \tag{1}$$

if and only if G is strongly connected. The vector π — called the **stationarty distribution vector** of G — encodes information about the cycle structure of G. That is, π_i^{-1} is the mean return time to vertex v_i in random walks in G. The smaller π_i^{-1} is, the more (short) cycles are expected to contain v_i. Because a FVS breaks all cycles, a vertex v_i with the least mean return time π_i^{-1} is considered to be a good candidate for the inclusion into a FVS.

So, the vertex selection heuristic first determines the stationary distribution vector $\pi = (\pi_i)_{v_i \in V}$ of the transition matrix P_G by solving the system (1) of linear equations using standard techniques (see section 3.3). Then, we choose the vertex v_i with the least mean return time π_i^{-1} to be included into the FVS. The selection process is summarised in Algorithm 2. Together with the framework from Algorithm 1 it yields the Markovian FVS algorithm (MFVS) developed in [31].

3.2 The mean approach

The idea behind the mean approach is simple: Any polynomial time greedy heuristic for the selection of vertices for an optimum FVS will fail eventually (unless $P = NP$). Although it cannot be expected to detect *all* failures, we would like to detect (and correct) at least *some* of them. In order to do so, another 'independent' heuristic of the same kind is required. With such an alternative heuristic, both of these heuristics can be exploited in a

```
function SelectVertexMean(digraph $G=(V,A)$) do

     matrix $P$ ← TransitionMatrix($G$);
     vector $\pi$ ← StationaryDistribution($P$);
     $P$ ← TransitionMatrix($G^{-1}$);
     $\pi$ ← $\pi$ + StationaryDistribution($P$);

     vertex $v$ ← vertex corresp. to maximum entry of $\pi$;

     return $v$;
od;
```

Algorithm 3: The vertex selection heuristic of $\mathrm{MFVS}_{\mathrm{Mean}}$.

way yet to be specified in order to select a vertex to be included into the FVS.

For heuristics based on Markov chains it is quite simple to construct such an alternative heuristic. For, note that a digraph G and its reversed digraph G^{-1} have the same FVSs. Hence, determine the stationary distribution — once for M_G and once for $M_{G^{-1}}$ — and take the componentwise mean of these two vectors. It remains to decide which mean to be taken — the arithmetic, the geometric or another mean. Experiments indicate that the arithmetic mean is best. Thus, the complete procedure for selecting a vertex is displayed in Algorithm 3.

Figure 2 shows an example digrasph due to Heusch, [27], for which $\mathrm{MFVS}_{\mathrm{Mean}}$ outperforms MFVS. While $\mathrm{MFVS}_{\mathrm{Mean}}$ determines the optimum solution $\{A,\, B\}$, MFVS successively selects the vertices $M_1, \ldots, M_r$. This example also shows that MFVS can behave arbitrarily bad. Nevertheless, such examples may exist for $\mathrm{MFVS}_{\mathrm{Mean}}$, too.

3.3 The runtime of MFVS and $\mathrm{MFVS}_{\mathrm{Mean}}$

Basically, the runtime of $\mathrm{MFVS}_{\mathrm{Mean}}$ is twice the runtime of MFVS. So, we concentrate on the runtime of MFVS.

Let $G = (V, A)$ be a digraph and let F be the FVS of G which is determined by MFVS. Because each vertex $v \in F$ is selected by means of solving a $|V| \times |V|$ system of linear equations which takes time $O(|V|^{2.376})$, [5], one might argue that MFVS runs in time $O(|V|^{2.376}|F|)$.

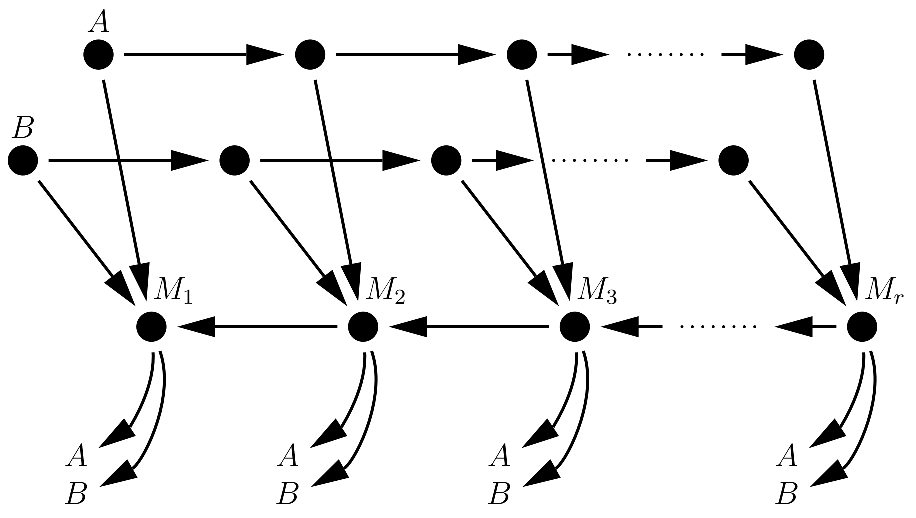

Figure 2: A pathological graph for MFVS where $\text{MFVS}_{\text{Mean}}$ determines the minimum cardinality FVS $\{A, B\}$.

However, we must keep in mind that the digraphs arising in practical applications are typically sparse. In addition, the stationary distribution vector π does not necessarily need to be evaluated to full precision in order to identify the largest entry of π. These two facts suggest applying an iterative method to solve the $|V| \times |V|$ system of equations. By doing so, we obtain a better runtime of MFVS.

A basic iterative method for solving a system of linear equations is the power method, [17, 34], which is used in our implementation. Applied to the problem of determining a vector $\pi = (\pi_i)$ satisfying ${}^t\pi P = {}^t\pi$ and $\sum \pi_i = 1$ for a stochastic matrix P (associated with M_G), the power method consists of computing ${}^t x P^k$ for sufficiently large values of k, where $x = (x_i)$ is an arbitrary vector with $\sum x_i = 1$. For if G is strongly connected and M_G is aperiodic, then

$$\lim_{k\to\infty} {}^t x P^k = {}^t\pi$$

holds. The periodic case is easily transformed into an equivalent aperiodic one.

Now, suppose we want to evaluate π to an accuracy of 10^{-d}. Let λ be the subdominant eigenvalue of P having index one. Then, as shown in [17], roughly

$$\frac{d}{-\log_{10}|\lambda|}$$

iterations are needed. On the other hand, P has exactly $|A|$ nonzero entries. Hence, the vector matrix product of any vector y and P can be implemented to run in time $O(|A|)$ using a sparse matrix package. This yields a runtime of $O(|A|\frac{d}{-\ln|\lambda|})$ to determine π to the desired accuracy. Because d usually is bounded, e.g. by the machine precision, it can be regarded as constant. So, the total runtime of MFVS reduces to

$$O(|A||F|\frac{1}{-\ln|\lambda|}).$$

Thus, the runtime of MFVS is mainly governed by $\frac{1}{-\ln|\lambda|}$. If $|\lambda|$ is well-separated from 1, i.e. there is some $\delta > 0$ with $|\lambda| \leq 1-\delta$ for each input instance $G=(V,A)$, MFVS runs in time $O(|A||F|)$, being considerably faster than the approach involving direct matrix inversion with the time bound of $O(|V|^{2.376}|F|)$. Unfortunately, this is not always the case. However, while the closeness of $|\lambda|$ to 1 might theoretically cause runtime problems, for most practical applications this is not an issue and the power method is the algorithm of choice. Although we have implemented it in the simplest form, we have not encountered any runtime problems. Besides, there are approaches to accelerate the power method. A survey can be found in [18], where the focus is on the PageRank computation, [3]. A more general approach to speedup the computation of the stationary distribution vector is based on the state compression technique described in [34].

4 Randomised algorithms

The GRASP (Greedy Randomised Adaptive Search Procedure) concept is often used to solve problems from areas such as propositional satisfiability, [24], crossing minimisation, [16], Steiner tree, [22], and set cover problems, [9], to mention just a few. The feedback vertex set problem is a particular set cover problem and Resende et al, [11], have adapted GRASP to solve this problem on digraphs.

Their algorithm using default settings (denoted by $\text{GRASP}_{\text{Resende}}$) works as follows: A FVS is constructed employing a greedy heuristic, as depicted in Algorithm 1. To select a vertex, first, a restricted candidate list (RCL) is constructed. It consists of all vertices v for which the degree product $\mathrm{d}^-_G(v)\cdot\mathrm{d}^+_G(v)$ is maximal. Then, a vertex of that list is chosen randomly to belong to the FVS. Once having determined a FVS for the digraph, it is transformed into a minimal one by testing vertices for redundancy in reversed order of insertion. In the default settings this process is repeated 2048 times and the smallest FVS generated is output.

4.1 Randomising $\mathrm{MFVS}_{\mathrm{Mean}}$

Inspired by the algorithm $\mathrm{GRASP}_{\mathrm{Resende}}$, we have been looking for a way to randomise $\mathrm{MFVS}_{\mathrm{Mean}}$ – the best deterministic Markov chain algorithm so far. For this purpose randomly perturb the entries of the transition matrix P, thus obtaining a matrix Q. Of course, Q is no longer stochastic. So, assuming that the entries of Q are still positive, the rows of Q have to be scaled appropriately to obtain a perturbed stochastic matrix $\widetilde{P}$. Obviously, the transition digraphs of the initial and the perturbed Markov chain are the same. Based on Markov chains perturbed in that way, the vertices are selected as described above. This process is iterated and the smallest FVS obtained is output.

How to perturb P? After several experiments, allowing a perturbation of up to 10% proved to be reasonable. The resulting algorithm is denoted by $\mathrm{GRASP}_{\mathrm{MFVS}_{\mathrm{Mean}}}$. This simple setting of the perturbation parameter seems unsatisfactory, though. Particularly, perturbing P depending on the matrix itself is desireable and remains to be investigated.

5 Searching locally with Markov chains

Local search procedures are frequently applied in many fields of combinatorial optmisation. In the case of directed FVS problems we are only aware of the uninformed search procedure described in [28]. Inspired by that procedure, a new informed search procedure MarkovSearch is devised which uses Markov chains.

Let $F \subset V$ be an initial minimal FVS of a digraph $G = (V, A)$, e.g. determined by $\mathrm{MFVS}_{\mathrm{Mean}}$. Then, MarkovSearch expands a fixed number of k minimal successor FVSs of F and holds them in a priority queue Q. After that, following the principle of best-first search, MarkovSearch extracts from Q an unexpanded FVS of minimum cardinality for expansion. A parameter t determines the number of such expansions. Finally, the best FVS is chosen as an approximation of the optimum one.

The expansion itself is governed by another parameter called **step width**. Suppose F and F' are two minimal FVSs of G and define the step width from F to F' by

$$\delta(F, F') := |F \setminus F'|.$$

If MarkovSearch generates F' as a successor FVS of F, we will say that MarkovSearch moves from F to F' in one step of width $\delta(F, F')$. On this basis, the expansion of successor FVSs is constrained in such a way that only a fixed step width is allowed – say a step width of at most d. Then, the generation of such successors is performed as follows.

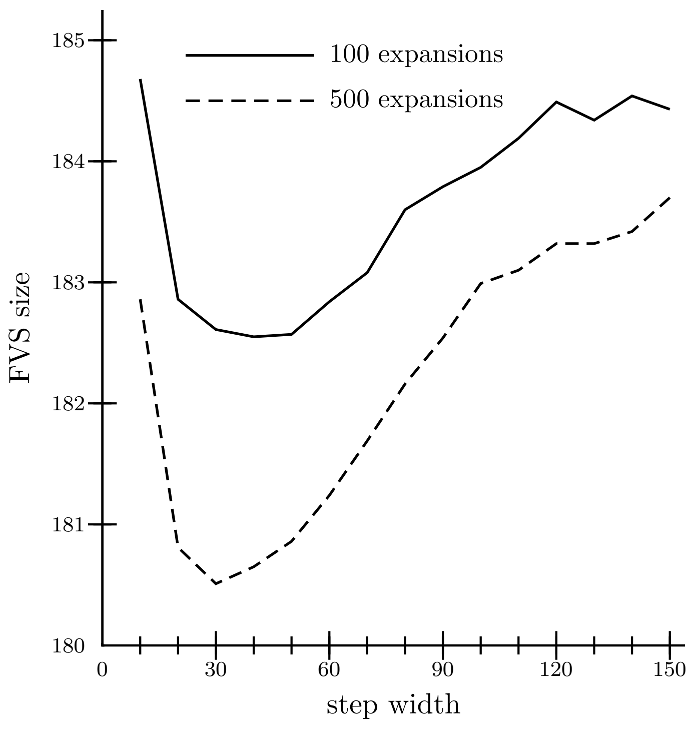

Figure 3: Test results for MarkovSearch on 100 random digraphs (with arc density $p = 0.05$) consisting of 300 vertices.

Select a subset $X \subset F$ of cardinality d and set $\widetilde{F} := F \setminus X$. Because of the minimality of F, $\widetilde{F}$ is not a FVS of G, i.e. $\widetilde{G} := G - \widetilde{F}$ is cyclic. Let $\widetilde{\widetilde{F}} := \mathrm{MFVS}_{\mathrm{Mean}}(\widetilde{G})$ be the FVS of $\widetilde{G}$ determined by $\mathrm{MFVS}_{\mathrm{Mean}}$. Then, $\widetilde{F} \cup \widetilde{\widetilde{F}}$ is a FVS of G with $\delta(F, \widetilde{F} \cup \widetilde{\widetilde{F}}) \leq d$ which is not necessarily minimal and has to be transformed into a minimal one by removing redundant vertices.

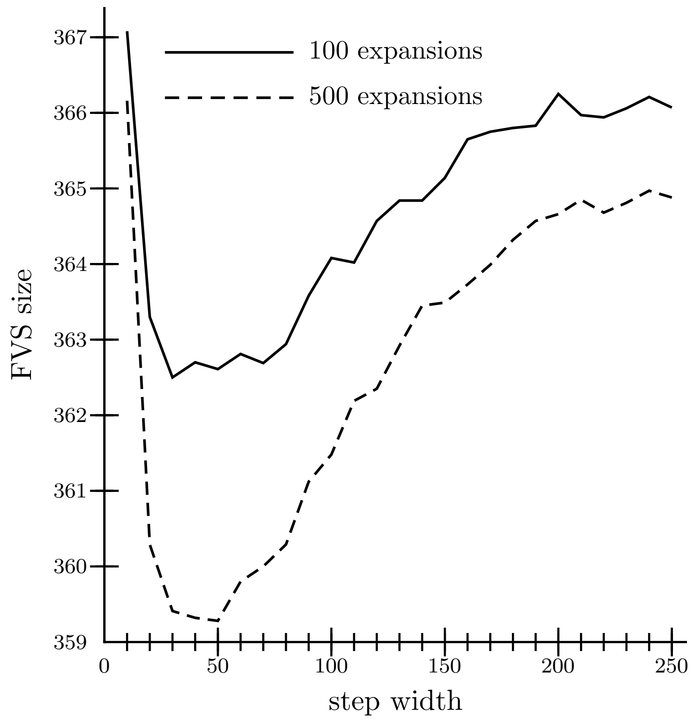

Figure 4: Test results for MarkovSearch on 100 random digraphs (with arc density $p = 0.05$) consisting of 500 vertices.

The FVS obtained in that way is denoted by F'. Note that $\delta(F, F') \leq d$ does not hold necessarily because some vertices of $\widetilde{F}$ might be redundant. Thus, d represents only a reference value for the step width. The choice of the subset X remains open. Experiments with several heuristics provided the best results for randomly chosen $X \subset F$. The complete search procedure is shown in Algorithm 4.

Determining a convenient step width d turned out to be difficult. As the optimum step width strongly depends on the given digraph, we have no rule determining d. Yet, Figures 3 and 4 give strong evidence that an optimum step width indeed exists depending on the parameters of the class of digraphs under consideration. Furthermore, the optimum step width does not seem to depend on the number of expansions. For further hints on the choice of the step width d we refer to [20].

```
function MarkovSearch(digraph $G=(V,A)$, integer $k$, $d$, $t$) do

    dictionary $D$
    priority queue $Q$;
    vertex set $F_{\mathtt{min}}$;
    vertex set $F$;
    vertex set $F'$;
    integer i;

    $D$ ← $\emptyset$;
    $Q$ ← $\emptyset$;

    $F$ ← $\mathtt{MFVS}_{\mathtt{Mean}}(G)$;
    $D$.insert($F$);
    $Q$.insert($F$);

    $F_{\mathtt{min}}$ ← $F$;

    while ($Q \neq \emptyset$ and $t > 0$) do
        $t$ ← $t-1$;

        $F$ ← $Q$.extract_min();

        for $i$ = 1 to $k$ do
            $F'$ ← GenSucc($G$, $F$, $d$);

            if ($D$.search($F'$) $\neq$ nil) do
                $D$.insert($F'$);
                $Q$.insert($F'$);

                if ($|F'| < |F_{\mathtt{min}}|$) do
                    $F_{\mathtt{min}}$ ← $F'$;
                od;
            od;
        od;
    od;

    return $F_{\mathtt{min}}$;
od;
```

Algorithm 4: The local search procedure MarkovSearch.

6 Experimental results

We want to demonstrate the effectiveness of the described algorithms. Since the test instances are often too big to allow for an optimum solution, we are not able to make a statement on the quality of the approximative solutions determined by the implementations of the FVS algorithms considered here. Therefore, we compare the solutions of these algorithms in order to evaluate their relative performances.

We first compare the algorithms MFVS and $\mathrm{MFVS}_{\mathrm{Mean}}$. To get a reference, we include test results of the algorithm Simple which selects the feedback vertices v according to the heuristic

$$\sum_{(u,v)\in A} \mathrm{d}_G^-(u) \cdot \sum_{(v,u)\in A} \mathrm{d}_G^+(u),$$

i.e. the product of the number of length-2 paths ending in v and starting from v, respectively. The tests are run with samples of 100 random digraphs from $G_p(n)$, where $G_p(n)$ is the class of digraphs G with n vertices and each arc is chosen independently with probability p. The results are shown in Table 1 and 2. As can be seen, $\mathrm{MFVS}_{\mathrm{Mean}}$ outperforms the other two. Furthermore, note that while MFVS is weaker than Simple, $\mathrm{MFVS}_{\mathrm{Mean}}$ performs considerably better than Simple. This demonstrates that the mean concept results in a major improvement over the original MFVS algorithm. Moreover, the time consumption of all three algorithms is almost the same – at least for sparse digraphs. Furthermore, as expected, the performance differences of the algorithms diminish as the density of the digraphs increases.

Because $\mathrm{MFVS}_{\mathrm{Mean}}$ turned out to be the best among the deterministic algorithms, we proceed comparing it with $\mathrm{GRASP}_{\mathrm{Resende}}$ which is considered to be the best approximation algorithm for the FVS problem in digraphs, [8]. The test set consists of 40 benchmark digraphs which are part of the $\mathrm{GRASP}_{\mathrm{Resende}}$ implementation, [11]. As reported in [13], that implementation had a bug which led $\mathrm{GRASP}_{\mathrm{Resende}}$ to produce too big FVSs. We have corrected the bug and Table 3 presents the results divergating from [11]. The table indicates that the deterministic single-iteration algorithm $\mathrm{MFVS}_{\mathrm{Mean}}$ can compete with the 2048 iterations of $\mathrm{GRASP}_{\mathrm{Resende}}$, although in the end $\mathrm{GRASP}_{\mathrm{Resende}}$ is still slightly better.

Next, we compare the randomised algorithms $\mathrm{GRASP}_{\mathrm{MFVS}_{\mathrm{Mean}}}$ and MarkovSearch to $\mathrm{GRASP}_{\mathrm{Resende}}$. $\mathrm{GRASP}_{\mathrm{MFVS}_{\mathrm{Mean}}}$ has been run performing 10 iterations with the first one being unperturbed. With respect to

vertices	Simple	MFVS	$\text{MFVS}_{\text{Mean}}$
50	7.53 (σ=1.77)	7.53 (σ=1.72)	7.48 (σ=1.70)
100	32.67 (σ=2.42)	33.38 (σ=2.31)	32.36 (σ=2.27)
150	67.27 (σ=2.88)	68.30 (σ=2.86)	66.33 (σ=2.50)
200	106.60 (σ=2.40)	108.39 (σ=2.73)	105.44 (σ=2.34)
250	148.53 (σ=2.62)	150.33 (σ=2.73)	146.81 (σ=2.66)
300	191.47 (σ=3.08)	194.32 (σ=3.14)	189.65 (σ=2.80)
350	236.39 (σ=2.98)	239.36 (σ=3.16)	234.04 (σ=2.51)
400	281.29 (σ=3.00)	285.54 (σ=2.89)	279.16 (σ=2.74)
450	327.58 (σ=2.94)	332.14 (σ=3.30)	325.04 (σ=2.64)
500	374.43 (σ=2.59)	379.16 (σ=3.19)	371.74 (σ=2.56)

Table 1: Test results for random digraphs with arc density $p = 0.05$. The average is taken over 100 instances. The standard deviation is denoted by σ.

vertices	Simple	MFVS	$\text{MFVS}_{\text{Mean}}$
50	17.73 (σ=1.68)	18.05 (σ=1.72)	17.49 (σ=1.62)
100	55.41 (σ=1.95)	56.03 (σ=1.97)	54.86 (σ=1.92)
150	98.44 (σ=2.16)	99.58 (σ=2.30)	97.26 (σ=1.96)
200	143.16 (σ=2.31)	145.09 (σ=2.34)	142.05 (σ=2.27)
250	189.52 (σ=2.23)	191.51 (σ=2.17)	188.26 (σ=2.09)
300	236.71 (σ=1.95)	239.31 (σ=2.30)	235.34 (σ=1.83)
350	284.80 (σ=1.94)	286.81 (σ=2.26)	283.03 (σ=2.08)
400	332.73 (σ=2.18)	335.77 (σ=2.10)	331.00 (σ=2.09)
450	380.77 (σ=2.17)	384.31 (σ=2.15)	378.96 (σ=1.98)
500	429.69 (σ=1.96)	432.65 (σ=1.98)	427.46 (σ=1.66)

Table 2: Test results for random digraphs with arc density $p = 0.1$. The average is taken over 100 instances. The standard deviation is denoted by σ.

MarkovSearch, in each of the 100 expansions two successor FVSs ($k = 2$) have been generated. For this special test suite the step width $d = \min\{40, \frac{|F|}{2}\}$ was chosen. Table 3 shows MarkovSearch to be superior. But even $\text{GRASP}_{\text{MFVS}_{\text{Mean}}}$ with its 10 iterations clearly outperforms $\text{GRASP}_{\text{Resende}}$. In this regard, it is remarkable that both algorithms produce much better results using only a fraction of iterations of $\text{GRASP}_{\text{Resende}}$ and consuming only a fraction of time despite the fact that our Markovian algorithms have a higher worst case runtime than the corresponding $\text{GRASP}_{\text{Resende}}$ version.

7 Conclusion

Our experiments have demonstrated that the new Markovian approximation algorithms for the minimum FVS problem outperform the existing ones with respect to both, quality of the delivered solutions and their runtimes. This holds particularly for MarkovSearch, so MarkovSearch can be considered as the algorithm of choice for the approximation of the minimum FVS problem.

References

[1] V. Bafna, P. Berman, T. Fujito *Constant ratio approximations of the weighted feedback vertex set problem for undirected graphs*, ISAAC95, Algorithms and Computation, J. Staples, P. Eades, N. Katoh and A. Moffat, editors, Lecture Notes in Computer Science Vol. 1004, Springer-Verlag, pages 142-151, 1995

[2] A. Becker, D. Geiger. *Approximation algorithms for the loop cutset problem.* In Proc. of the 10th conference on Uncertainty in Artificial Intelligence, pages 60-68, 1994

[3] S. Brin, L. Page, R. Motwami, T. Winograd. *The PageRank citation ranking: bringing order to the Web*, Technical Report 1999-0120, Computer Science Department, Stanford University, 1999

[4] M. Cai, X. Deng, W. Zang. *An Approximation Algorithm for Feedback Vertex Sets in Tournaments.* SIAM J. Comput. 30, 6, pages 1993-2007, 2001

[5] D. Coppersmith, S. Winograd. *Matrix multiplication via arithmetic progressions*, Journal of Symbolic Computation, Vol. 9, No. 3, pages 251–280, 1990

[6] C. Demetrescu, I. Finocchi. *Combinatorial Algorithms for Feedback Problems in Directed Graphs* Information Processing Letters 86 (IPL), pages 129-136, 2003

[7] G. Even, J. Naor, B. Schieber, M. Sudan. *Approximating minimum feedback sets and multicuts in directed graphs.* Algorithmica, 20(2), pages 151-174, 1998

[8] F. Fages, A. Lal. *A Global Constraint for Cutset Problems*, Proceedings CPAIOR, 2003

[9] T. A. Feo, M. G. C. Resende. *A probabilistic heuristic for a computationally difficult set covering problem*, Operations Research Letters, 8:67-71, 1989

[10] T. A. Feo, M. G. C. Resende. *Greedy randomized adaptive search procedures*, Journal of Global Optimization, 6, 1995

[11] Paola Festa, Panos M. Pardalos, Mauricio G. C. Resende. *Fortran Subroutines For Computing Approximate Solutions Of Feedback Set Problems Using GRASP*

[12] M. X. Goemans, D. P. Williamson. *Primal-dual approximation algorithms for feedback problems in planar graphs.* In W. H. Cunningham, S. T. McCormick, and M. Queyranne, editors, Integer Programming and Combinatorial Optimization, number 1084 in Lecture Notes in Computer Science, pages 147-161. Springer-Verlag, 1996

[13] B. Hasselman. *An efficient method for detecting redundant feedback vertices*, CPB Netherlands Bureau for Economic Policy Analysis, Discussion Paper 29, 2004

[14] D. S. Hochbaum, editor. *Approximation Algorithms for NP-Hard Problems*, Thomson Publishing Company, 1997

[15] R. Karp. *Reducibility among combinatorial problems*, In R. Miller and J. Thatcher, editors, *Complexity of Computer Communications*, pages 85–103. Plenum Press, 1972

[16] M. Laguna, R. Martí. *GRASP and path relinking for 2-layer straight line crossing minimization*, INFOMRS Journal on Computing, 1998

[17] A. N. Langville, C. D. Meyer. *Updating Markov chains with an eye on Google's PageRank*, SIAM J. Matrix Analysis and Applications, Vol. 27, No. 4, pages 968–987, 2006

[18] A. N. Langville, C. D. Meyer. *A Survey of Eigenvector Methods for Web Information Retrieval*, SIAM Review, Vol. 47, No. 1, pages 135–161, 2005

[19] D. Lee, M. Reddy. *On determining scan flip-flops in partial-scan designs.* In Proceedings of the International Conference on Computer-Aided Design. pages 322–325, 1990

[20] M. Lemaić. *Markov-Chain-Based Heuristics for the Feedback Vertex Set Problem for Digraphs*, Ph.D. thesis, Institut für Informatik, Universität zu Köln, Germany, http://kups.ub.uni-koeln.de/volltexte/2008/2547/, 2008

[21] H. Levy, D. W. Low. *A contraction algorithm for finding small cycle cutsets.* Journal Of Algorithms Vol. 9. pages 470–493, 1988

[22] S. L. Martins, C. C. Ribeiro. *A parallel GRASP for the Steiner problem in graphs*, In Proceedings of the Irregular 98, 1998

[23] S. Park, S. Akers. *A graph theoretic approach to partial scan design by k-cycle elimination.* In The Proceedings of the International Test Conference. pages 303–311, 1992

[24] M. G. C. Resende, T. A. Feo. *A GRASP for satisfiability*, In D. S. Johnson, M. A. Trick, editors, *Cliques, Coloring and Satisfiability: The Second DIMACS Implementation Challenge*, volume 26 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 499-520, American Mathematical Society, 1996

[25] B. K. Rosen. *Robust linear algorithms for cutsets*, Journal of Algorithms, Vol. 3, pages 205– 212, 1982

[26] S. M. Ross. *Introduction to probability models*, Academic Press, 1993

[27] B. Schwikowski. *Empirische Analyse von Approximationsalgorithmen für das Feedback-Vertex-Set-Problem in Digraphen*, Diploma thesis, Heinrich-Heine-Universität Düsseldorf, 1994

[28] B. Schwikowski, E. Speckenmeyer. *On computing all minimal solutions for feedback problems*, Discrete Applied Mathematics 117, 253–265, 2002

[29] P. D. Seymour, *Packing directed circuits fractionally.* Combinatorica Vol. 15, pages 281–288, 1995

[30] A. Shamir. *A linear time algorithm for finding minimum cutsets in reducible graphs*, SIAM Journal on Computing, Vol. 8 No. 4, pages 645–655, 1979

[31] E. Speckenmeyer. *On Feedback Problems in Digraphs.* In Graph-Theoretic Concepts in Computer Science, pages 218-231. Springer-Verlag, Berlin, Heidelberg, 1989

[32] R. E. Tarjan. *Depth-first search and linear graph algorithms*, SIAM Journal on Computing, Vol. 1, No. 2, pages 146–160, 1972

[33] B. Yehuda, J. Geiger, J. Naor, R. M. Roth. *Approximation algorithms for the vertex feedback set problem with application in constraint satisfaction and bayesian inference*, In Proc; 5th Annual ACM-SIAM Symposium on Discrete Algorithms, pages 344–354, 1994

[34] A. Xie, P. A. Beerel. *Accelerating Markovian analysis of asynchronous systems using string-based state compression*, In Proc. International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC), pages 247–260, IEEE Computer Society Press, 1998

vert.	arcs	Opt.	$\text{MFVS}_{\text{Mean}}$	$\text{GRASP}_{\text{Resende}}$	$\text{GRASP}_{\text{MFVS}_{\text{Mean}}}$	M-Search
50	100	3	3	3	3	3
50	150	9	9	9	9	9
50	200	13	13	13	13	13
50	250	17	17	17	17	17
50	300	19	20	19	20	19
50	500	28	29	28	29	28
50	600	31	32	31	32	32
50	700	33	34	33	33	33
50	800	34	37	34	34	35
50	900	36	38	36	36	36
100	200	9	9	9	9	9
100	300	17	19	17	17	19
100	400	23	25	23	25	23
100	500	32	33	32	33	32
100	600	36	39	38	38	37
100	1000	53	55	54	55	53
100	1100	54	56	55	56	55
100	1200	57	58	58	58	57
100	1300	60	63	61	62	60
100	1400	61	66	62	64	62
500	1000	31	34	32	33	31
500	1500	≤ 63	70	68	69	65
500	2000	≤ 101	108	107	108	105
500	2500	≤ 132	143	146	143	138
500	3000	≤ 162	178	175	173	167
500	5000	≤ 238	254	252	250	244
500	5500	≤ 252	270	270	266	261
500	6000	≤ 266	282	284	279	273
500	6500	≤ 277	296	294	292	288
500	7000	≤ 289	305	304	305	298
1000	3000	≤ 128	143	139	137	132
1000	3500	≤ 162	175	175	172	168
1000	4000	≤ 193	210	206	205	201
1000	4500	≤ 229	248	249	248	237
1000	5000	≤ 260	282	283	275	267
1000	10000	≤ 473	505	508	497	490
1000	15000	≤ 588	620	619	612	600
1000	20000	≤ 655	682	692	681	674
1000	25000	≤ 704	736	733	730	718
1000	30000	≤ 748	768	773	766	762
sum		≤ 6576	6964	6941	6883	6749
time			193 sec.	41626 sec.	800 sec.	733 sec.

Table 3: Test results for the test suite supplied with $\text{GRASP}_{\text{Resende}}$.