Journal of Heuristics, 7: 235–250 (2001)

A Fast and Effective Algorithm for the Feedback Arc Set Problem

YOUSSEF SAAB
Computer Engineering and Computer Science Department, 321 Engineering Building West, University of Missouri-Columbia, Columbia, MO 65211, USA
email: saab@cecs.missouri.edu

Abstract

A divide-and-conquer approach for the feedback arc set is presented. The divide step is performed by solving a minimum bisection problem. Two strategies are used to solve minimum bisection problem: A heuristic based on the stochastic evolution methodology, and a heuristic based on dynamic clustering. Empirical results are presented to compare our method with other approaches. An algorithm to construct test cases for the feedback arc set problem with known optimal number of feedback arcs, is also presented.

Key Words: feedback set, acyclic subgraph, consistent inequalities

1. Introduction

A feedback set in a directed graph is a set of arcs that includes at least one arc of every directed cycle. The feedback arc set problem (FAS) seeks a feedback set of minimum size. The removal of all arcs in a feedback set makes the resulting graph acyclic. The interest in FAS dates back to as early as 1957 (Unger, 1957) in a study of asynchronous logical feedback networks. Younger (1963) and Lempel and Cederbaum (1966), mention that FAS is included among a list of research problems in Seshu and Reed [1961, p. 299], and attribute the original formulation of FAS to Runyon. The early interset in FAS seems to be due to its importance in simplifying the analysis of feedback systems. A frequently used approach in the analysis of large-scale systems with feedback is to remove a small set of arcs to make the system feedback-free. Well-established methods are then used to study this feedback free system. The perturbation caused by the removed arcs are then reintroduced and iteratively analyzed. The speed of convergence of the iterative analysis process has been empirically observed to be proportional to the number of removed arcs (Ramachandran, 1988). Another application of FAS is the determination of a large set of consistent inequalities (Younger, 1963). Given a collection of inequalities among several variables, certain inequalities may be inconsistent with others if these inequalities must be transitive. For example, $d < a$ is inconsistent with $a < b$, $b < c$, and $c < d$. Removing a minimum number of inconsistent inequalities, results in a large collection of consistent inequalities. By viewing the variable as vertices of a graph and inequalities as arcs between vertices, a minimum feedback set of the resulting graph, represents a minimum set of inequalities that are inconsistent with others. Slater (1971) studied FAS in the context of tournaments. A tournament is a special

graph in which an arc $u \rightarrow v$ represents a win for u over v in a round-robin tournament. Slater considered the reversal of a minimum set of arcs that would make the tournament consistent. Recently, FAS found applications in information visualization and specifically in drawing directed graphs, where it is desired that most arcs are monotonic in a certain direction (Eades and Lin, to appear).

2. Preliminaries

A directed graph $G(V, E)$ consists of a set of vertices V and a set of arcs E. Each arc is an ordered pair of vertices. For simplicity, in this paper the term graph means "directed graph". An arc $u \rightarrow v$ is incident with both u and v, it is outgoing from u, it is incoming to v, and it is directed from u to v· vertices u and v are respectively the origin and terminus of $u \rightarrow v$. The indegree (outdegree) of a vertex v is the number of arcs incoming to (outgoing from) v and is denoted by $d^-(v)(d^+(v))$. The degree of a vertex v is $d(v) = d^-(v) + d^+(v)$. A source is a vertex with no incoming arcs and a sink is a vertex with no outgoing arcs. Thus a vertex v is a source if $d^-(v) = 0$ and is a sink if $d^+(v) = 0$. There can be more than one arc between a pair of vertices and in both directions. A path from u to v is a sequence of arcs $\langle e_1, \ldots, e_k \rangle$ such that: u is the origin of e_1, v is the terminus of e_k, and the terminus of e_i is the origin of e_{i+1} arc for $1 \leq i < k$. A cycle is a path $\langle e_1, \ldots, e_k \rangle$ such that the terminus of e_k is the origin of e_1. The length of a cycle equals the number of its edges. For a subset X of vertices of a graph $G(V, E)$, $G[X]$ is the subgraph of G induced by X. The set of vertices of $G[X]$ is X, and the set of arcs of $G[X]$ includes all arcs in G that are between vertices in X. Relation R on vertices of G given by: $u \; R \; v$ if there exist a path from u to v and a path from v to u, is an equivalence relation that partitions the vertex set into equivalence classes. The subgraphs induced by the equivalence classes of R are the strongly connected components of G. Graph G is strongly connected if all its vertices are in one equivalence class under R.

3. Previous approaches

FAS is NP-Hard for general graphs (Garey and Johnson, 1979), but it is solvable in polynomial time for planar graphs (Lucchesi, 1976; Lucchesi and Younger, 1978) and for reducible flow graphs (Ramachandran, 1988). A feedback arc set of size no more than $\frac{1}{2}|E|$ can be obtained using the following heuristic (Berger and Shor, 1990):

```
F := ∅;
while G ≠ ∅ do
        select a vertex v in G;
        if d⁻(v) < d⁺(v) then
                add all arcs incoming to v to F;
        else
                add all arcs outgoing from v to F;
        remove v and all arcs incident to it from G
return F.
```

The bound $\frac{1}{2}|E|$ is the best possible for general graphs. For example, if $G(V, E)$ is a graph such that, for any pair of vertices u and v in V, both $u \rightarrow v$ and $v \rightarrow u$ are in E, then any minimum feedback arc set of G contains exactly $\frac{1}{2}|E|$. For a graph $G(V, E)$ with no cycles of length 2, Berger and Shor (1990), by using a specific order for selecting the next vertex to be processed by the above heuristic, obtained an algorithm that computes a feedback arc set of cardinality at most $(\frac{1}{2} - \frac{c}{\Delta^2})|E|$ where Δ is the maximum degree of G and c is a constant. Another approach for computing a feedback arc set of size no more than $\frac{1}{2}|E|$ exploits a connection between vertex ordering and feedback arc sets. A vertex ordering is the arrangement of vertices on a line from left to right. Under such ordering each arc is directed either from left to right or from right to left. Each of the set of all leftward arcs and the set of rightward arcs is a feedback arc set. The smaller of these two sets is a feedback arc set of size at most $\frac{1}{2}|E|$.

The connection between vertex ordering and feedback arc sets was first used by Younger (1963) who established the following:

1. There exists a vertex ordering such that the set of all leftward arcs is a minimum feedback arc set. Such an ordering is called an optimum ordering.
2. For two subgraphs G_1 and G_2 of a graph $G(V, E)$, let $N(G_1, G_2)$ be the number of arcs each of which is outgoing from a vertex in G_1 and incoming to a vertex in G_2. A consecutive subgraph of a graph $G(V, E)$ with respect to a vertex ordering is a subgraph induced by a subset of consecutive vertices in this ordering. If G_1 and G_2 are two consecutive subgraphs of G with respect to an optimum ordering of G such that the last vertex in G_1 precedes the first vertex in G_2, then $N(G_1, G_2) \geq N(G_2, G_1)$.
3. A feedback arc set F of a graph $G(V, E)$ is minimal if it contains no proper subset that is also a feedback arc set. So, for any arc $e \in F$, the subgraph $H(V, (E - F) \cup \{e\})$ of G has at least one cycle. The set of all leftward arcs with respect to an optimum ordering of a graph $G(V, E)$ must be minimal.

Properties (2) and (3) are necessary for a vertex ordering to be optimum. A vertex ordering that satisfies: (a) $N(G_1, G_2) \geq N(G_2, G_1)$ for all appropriate consecutive subgraphs G_1 and G_2, and (b) the set of all leftward arcs is a minimal feedback arc set, is called an admissible ordering. Any ordering can be transformed into an admissible ordering using the following iterative improvement method:

let $v_1, \ldots, v_n$ *be an initial ordering;*
improve := *true;*
while *improve* = *true* do
    improve := *false;*
    for $1 \leq i \leq j < k \leq n$ do
        $G_1 := G[v_i, \ldots, v_j]$;
        $G_2 := G[v_{j+1}, \ldots, v_k]$;
        if $N(G_1, G_2) < N(G_2, G_1)$ then
            change order to $v_1, \ldots, v_{i-1}, v_{j+1}, \ldots, v_k, v_i, \ldots, v_j, v_{k+1}, \ldots, v_n$;
            renumber vertices according to the new order;
            improve := *true;*

```
        F := set of leftward arcs with respect to current ordering;
        for each e ∈ L do
            if H(V, (E − F) ∪ {e}) is acyclic then
                use topological sorting [Cormen (1990), p. 485] to reorder H;
                renumber vertices according to new order;
                remove from F all arcs that become rightward;
                improve := true
return v_1, ..., v_n.
```

The connection between vertex orderings and feedback arc sets is also exploited by other researchers. Eades, Lin and Smyth (1993) developed a linear time algorithm (ELS) that first finds an ordering and then puts all leftward arcs in the feedback arc set. Their heuristic maintains two lists $S1$ and $S2$ which are initially empty. Each vertex is processed once and is either placed at the end of $S1$ or at the beginning of $S2$. After processing all vertices, $S1$ and $S2$ are concatenated. A more precise description of this method is in the following pseudo-code:

```
S1 := ∅;
S2 := ∅;
while G ≠ ∅ do
        if there is a sink u in G then
                put u at the beginning of S2 and set v := u
        else if there is a source u in G then
                put u at the end of S1 and set v := u
        else
                choose a vertex u for which d+(u) − d−(u) is maximum;
                put u at the end of S1 and set v := u;
        remove v and all arcs incident to it from G;
concatenate S1 and S2 into a list S: place S2 at the end of S1;
return set of all leftward arcs in the ordering S.
```

For a tournament graph $G(V, E)$, the above heuristic produces a feedback arc set of size at most $\frac{1}{2}|E| - \frac{1}{6}|V|$. The above heuristic is also improved in Eades and Lin (to appear) by increasing the greediness of the choice of the next vertex to be processed when there are no sources or sinks in the graph. The improved heuristic produces a feedback arc set of size at most $\frac{1}{4}|E|$ for a cubic graph $G(V, E)$.

Another heuristic by Eades, Smyth and Lin (ESL) (1989) finds a feedback arc set of all leftward arcs in a vertex ordering obtained using the following divide-and-conquer procedure:

```
order(G)
        if G has no arcs then
                S := any vertex sequence
        else if G has an odd number of vertices then
```

```
        let v be a vertex of minimal indegree in G;
        remove v and all arcs incident to it from G;
        S1:= order(G);
        prepend v to S1 to form S
else
        sort vertices of G into non-decreasing indegree order v1, ..., vn;
        G1 := subgraph of G induced by v1, ..., vn/2;
        G2 := subgraph of G induced by vn/2+1, ..., vn;
        S1 := order(G1);
        S2 := order(G2);
        concatenate S1 with S2 to form S
return S.
```

The last approach we mention in this section is due to Lempel and Cederbaum (1996). Their algorithm finds all the minimum feedback arc sets simultaneously. First a boolean expression in product-of-sums form is derived from the permanent of the arc adjacency matrix of the graph. The boolean expression is then expanded into sum-of-product form. Finally the minimum feedback arc sets are those corresponding to products with the least number of literals.

4. The new approach

The method of Lempel and Cederbaum (1996), although it finds all minimum feedback arc sets, requires an exponential time as one may expect since FAS is NP-Hard (Garey and Johnson, 1979). The method that finds an admissible vertex ordering as suggested by Younger (1963) can be quite effective in finding near-optimal solutions but requires $\Omega(n^4)$ computation time. Other approaches that are not computationally expensive, may fail to find near-optimal solutions because they process vertices in some order based on local selection criteria, and after a vertex is selected, either all incoming arcs or all outgoing arcs of that vertex are added to the feedback arc sets. The divide-and-conquer strategy used in the heuristic of Eades, Smyth, and Lin (1989), is efficient since the problem is quickly reduced to small subproblems that can be easily handled. However, the effectiveness of this approach is dependent on how the divide step is performed. Eades, Smyth, and Lin choose to divide the graph based on the indegrees of the vertices. Although it is plausible that vertices of small indegrees be placed to the left of other vertices, The divide step suggested by Eades, Smyth, and Lin does not take into account the final outcome of the division, which is revealed by asking the question: Which arcs are forced into the feedback arc set by the divide step? Recall that in the divide-and-conquer heuristic of Eades, Smyth, and Lin, graph G is divided into two subgraphs G_1 and G_2 and the vertices of G_1 precede those in G_2 in the final ordering. Since the feedback arc set delivered consists of all leftward arcs with respect to this ordering, the arcs that are forced into the feedback arc set, are those that originate in G_2 and terminate in G_1. Therefore for delivering small feedback arc sets, the aim of the divide step should be the minimization of the number of arcs that become leftward after each divide step. This is exactly what our method does.

Our divide-and-conquer method were mainly motivated by a result of Younger (1963). We became aware of the divide-and-conquer heuristic of Eades, Smyth and Lin (1989) after the development of our method. Younger proved that if H is a consecutive subgraph with respect to an optimum ordering of a graph G, then the restriction of the optimum ordering to the vertices of H, is an optimum ordering for H. Suppose now that we have a method that partitions the vertex set V of G into two subsets V_1 and V_2 such that there is an optimum ordering of G with all vertices of V_1 to the left of those in V_2. We can then obtain a minimum feedback arc set of G by taking the union of a minimum feedback arc set in $G_1 = G[V_1]$ and a minimum feedback arc set in $G_2 = G[V_2]$ and $\{i \rightarrow j : i \in V_2$ and $j \in V_1\}$. Therefore, the existence of a polynomial-time algorithm that partitions the vertex set as prescribed is unlikely given that FAS is NP-Hard (Garey and Johnson, 1979). As an approximation, we may settle for a less ambitious task, which involves finding a partition (V_1, V_2) such that the cardinality of the set $\{i \rightarrow j : i \in V_2$ and $j \in V_1\}$ is minimized. This entails finding a minimum cut, which can be found using network flow theory (Ford and Fulkerson, 1962). However, in order to find a minimum cut, we have to solve a maximum flow problem between every pair of vertices. For a graph with n vertices and m arcs, if we use the preflow-push algorithm of Goldberg and Tarjan (1986) which runs in $\mathcal{O}(nm \log(n^2/m))$, to solve each flow problem, we end up with a minimum cut algorithm that runs in $\mathcal{O}(n^3 m \log(n^2/m))$. Such an algorithm is computationally expensive and contradicts our motivation for using a divide-and-conquer strategy to achieve an efficient algorithm. A minimum cut algorithm may yield an unbalanced partition and may have to be invoked $\mathcal{O}(n)$ time. To achieve efficiency, we must insist on finding a balanced partition.

Define a partition of a graph $G(V, E)$ as an ordered pair (V_1, V_2) such that $V_1 \cup V_2 = V$ and $V_1 \cap V_2 = \emptyset$. Define the cost of partition (V_1, V_2) as $cost(V_1, V_2) = |\{i \rightarrow j : i \in V_2$ and $j \in V_1\}|$. Define a bisection of a graph $G(V, E)$ as a partition (V_1, V_2) of V such that the cardinalities of the two subsets V_1 and V_2 are as close as possible. In practice, this can be enforced by requiring that $|V_1| \leq \alpha|V|$ and $|V_2| \leq \alpha|V|$, where $1/2 \leq \alpha < 1$ is a constant. In our experiments to be presented later, we used $\alpha = 0.6$. Now consider the following problem:

The graph bisection problem (*GB*): Given a directed graph $G(V, E)$, find a bisection (V_1, V_2) of G of minimum cost.

GB is a generalization of the undirected graph bisection problem (UGB) which is NP-Hard (Garey and Johnson, 1979). By a simple transformation, UGB can be reduced to GB. Given an undirected input graph $G(V, E)$, construct a directed graph $G'(V, E')$, where $E' = \{i \rightarrow j, j \rightarrow i : i - j \in E\}$. A bisection (V_1, V_2) of G' is optimal for GB if and only if it is optimal in G for UGB. Therefore, this simple transformation shows that GB is also NP-Hard, so it is unlikely that an efficient polynomial-time algorithm for GB exists. However, in the past few years we have developed fast an effective heuristics for UGB, which, with minor efforts, can be modified to work for GB. We will discuss such methods later in this paper. For now, let us assume that we have at our disposal a function *bisect*(G) that takes a directed graph $G(V, E)$ as input and returns a bisection (V_1, V_2) of G of small cost. Such a function is used in a divide-and-conquer strategy aiming at reducing the solution of a large graph to the solution of smaller graphs. By noting that an arc can be part of a cycle only if it is origin and terminus are in the same strongly connected component

of the input graph, an algorithm that partitions the vertex set into several subsets each of which induces one strongly connected component is also used [Cormen (1990), p. 488] to decompose the graph into smaller and more manageable pieces. Let $scc(G)$ be a function that takes a directed graph $G(V, E)$ as input and returns a partition $P = \{S : S \subseteq V$ and S induces a strongly connected component of $G\}$. We are now ready to present our approach for FAS:

```
fas(G)
    P:= scc(G);
    if P has only one element then {G is strongly connected}
        (V1, V2) := bisect(G);
        F := fas(G[V1]) ∪ fas(G[V2]) ∪ {i → j : i ∈ V2 and j ∈ V1}
    else
        F := ∅;
        for each S ∈ P do
            F := F ∪ fas(G[S])
    return F.
```

Function *fas* returns a feedback arc set F of its input graph $G(V, E)$. If G is not strongly connected, then F is computed as the union of sets each of which is a feedback arc set in one strongly connected component of G. If G is strongly connected, then function *bisect* is used to decompose the vertex set of G into two subsets V_1 and V_2 of about equal size. The set F is then computed as the union of a feedback set of F_1 of $G[V_1]$, a feedback set F_2 of $(G[V_2]$ and the set $L = \{i \rightarrow j : i \in V_2$ and $j \in V_1\}$. Function fas is efficient because the input graph is quickly decomposed into smaller subgraphs either by function *bisect* or by function *scc*.

5. Solution of GB by stochastic evolution

Stochastic evolution (SE) is a general methodology for combinatorial optimization. In Saab (1990), SE was applied to Network Bisection, Vertex Cover, Set Partition, Hamilton Circuit, Traveling Salesman, Linear Ordering, Standard Cell Placement, and Multi-way Circuit Partitioning problems. After the publication of Saab and Rao (1991), SE was used by other researchers on different kinds of problems such as estimating the complexity of synthesized design designs from finite-state-machine specifications (Mitra, Panda and Pal, 1993), ordering of binary decision diagrams (Ney et al. 1992), automated system partitioning for synthesis of multi-chip modules (Raghava and Bayoumi, 1994), and for the minimization of the frontwidth in finite element calculations (Souza et al., 1994). This section presents a specific adaptation of SE for solving GB. Readers are referred to Saab (1990) and Saab and Rao (1991) for a general description of SE.

SE is an process that iteratively improves an initial solution through a sequence of local perturbations called moves. Let (V_1, V_2) be a partition of the vertex set of a directed graph $G(V, E)$. This partition can be changed by moving one vertex from V_1 to V_2 or from V_2 to V_1. for each vertex $i \in V$, let *move*(i) be a function that transfers vertex i from its current

subset to the complementary subset in the partition (V_1, V_2), and let $gain(i)$ be the reduction in $cost(V_1, V_2)$ after $move(i)$ is executed, so

$$gain(i) = cost(V_1, V_2) - cost(V_1 - \{i\}, V_2 \cup \{i\}) \text{ if } i \in V_1,$$

and

$$gain(i) = cost(V_1, V_2) - cost(V_1 \cup \{i\}, V_2 - \{i\}) \text{ if } i \in V_2.$$

The cost of (V_1, V_2) is decreased by $gain(i)$ after execution of $move(i)$. Therefore, If $gain(i) > 0$ ($gain(i) < 0$), then $cost(V_1, V_2)$ will be reduced (increased) after executing $move(i)$. A partition (V_1, V_2) is a bisection, if $|V_1| \leq \alpha|V|$ and $|V_2| \leq \alpha|V|$; where $1/2 \leq \alpha < 1$ is a constant. Let $randint(l, h)$ be a function that returns a random number in the interval $[l, h]$. An initial bisection (V_1, V_2) is changed using the following function *perturb* which uses a parameter $p \leq 0$ and two stacks S_1 and S_2 to store, in last-first order, vertices moved from V_2 to V_1 and from V_1 to V_2, respectively:

```
perturb(V, V1, V2, p)
        S1 := ∅;
        S2 := ∅;
        for each i ∈ V do
                if gain(i) > randint(p,0) then
                        move(i);
                        if i ∈ V1 then
                                push i onto stack S1
                        else
                                push i onto stack S2;
        if |V1| > |V2| then j = 1 else j = 2;
        while |Vj| > α|V| do
                        pop a vertex i from Sj;
                        move(i); {reverses previous move(i)}
```

Note that since $p \leq 0$, $move(i)$ is always performed when $gain(i) > 0$. However, when $gain(i) \leq 0$, $move(i)$ may or may not be performed depending on the random number returned by *randint*. The last while loop in *perturb* corrects any size imbalance that may have resulted from moving vertices between V_1 and V_2. This is achieved by returning the last vertices moved into the larger of V_1 and V_2 to the other subset. The overall SE algorithm for GB can now be described as follows:

```
(V1, V2) := a random initial bisection of G(V, E);
(B1, B2) := (V1, V2); {save best bisection}
p := p0; {initial value for parameter p}
set value for iteration control parameter R;
counter := 0;
```

```
repeat
        C_pre := cost(V1, V2);
        perturb(V, V1, V2, p);
        C_post := cost(V1, V2);
        if C_post < C_pre then
                (B1, B2):=(V1, V2); {save best bisection}
                counter := counter - R {allow for more iterations}
        else
                counter:=counter + 1;
        if C_post = C_pre then
                p := p − δ {decrease p to allow for more movements of vertices in
                                perturb}
        else
                p := p0 {restore original value of p}
until counter > R
return (B1, B2).
```

The SE algorithm requires three input parameters R, p_0 and δ. Parameter R controls the number of iteration of SE. In each iteration of SE, if the current bisection is improved, then iteration counter is decremented by R, and so SE rewards itself by allowing more iterations. Otherwise, the iteration counter is incremented by 1. The parameter p_0 initializes the variable p which controls the rate of vertex movements in *perturb*. The smaller is the value of p, the larger is the rate of vertex movements in *perturb*. When p is close to 0, few vertices are moved in *perturb* and SE behaves like a descent method, since the gain of most moved vertices is greater than 0 at the time they are moved in *perturb*. The parameter δ is used to decrease the value of p, when little or no change to the current bisection has occurred in the last call to *perturb*, as signified by the fact that the cost of the bisection did not change. Whenever *perturb* changes the cost of the current bisection, p is reset to its initial value p_0. In our implementation of SE, we used $R = 10$, $p_0 = -1$, and $\delta = 2$. The best bisection found in all iterations is returned.

6. Solution of GB by dynamic clustering

Clustering, contraction, or compaction describe an operation in which a subset X of vertices of a graph $G(V, E)$, is coalesced to form a single new vertex x. Arcs entering (leaving) a vertex originally in X, enter (leave) the new vertex x in the modified graph. More precisely, the vertex set of G becomes $(V - X) \cup \{x\}$, and each arc $u \to v$ in E is replaced as follows: (1) If both u and v are in X then arc $u \to v$ is discarded, (2) If both u and v are in $V - X$ then arc $u \to v$ is left unchanged, (3) If u is in X and v is in $V - X$ then arc $u \to v$ is replaced by arc $x \to v$, and (4) If u is in $V - X$ and v is in X then arc $u \to v$ is replaced by arc $u \to x$.

For the undirected graph bisection problem and for network bisections, Clustering has previously been used by several researchers to enhance the performance of iterative methods. In De Souza et al. (1994) and Saab (1995), we proposed an algorithm for network bisection

that combines clustering with iterative improvement, and so we call such a strategy dynamic clustering. Since bisectioning of directed graphs is quite similar to bisectioning undirected graphs, we adapt the method used in De Souza et al. (1994) and Saab (1995) for bisecting networks to solve GB. A current bisection is modified by moving sequences of vertices from one subset of the bisection to the other. In each sequence, a subset of vertices is identified for clustering. All chosen subsets, which are also disjoint, are clustered simultaneously and the iterative improvement process is repeated on the clustered graph. When no improvement or clustering can be achieved, the iterative process repeats again for the original unclustered graph until no further improvement can be made. The best bisection found is returned. Let *improve_and_cluster*$(G, V_1, V_2, G', V_1', V_2')$ be a function that takes as input a graph $G(V, E)$ along with an initial bisection (V_1, V_2) of G, and outputs a compacted graph $G'(V', E')$ along with a bisection (V_1', V_2') of G'. A pseudo-code for the dynamic clustering algorithm for GB can be described as follows:

Generate a random initial bisection (V_1, V_2) *of* G*;*
save (V_1, V_2) *as the best current bisection;*
while *(improvements are made)* do
    let H *be a copy of graph* G*;*
    let (P_1, P_2) *be the best current bisection;*
    while *(improvement or compactions are made)* do
        improve_and_cluster$(H, P_1, P_2, H', P_1', P_2')$*;*
        $H := H'$*;*
        $(P_1, P_2) := (P_1', P_2')$*;*
    compute the new best bisection of G *from* (P_1, P_2)*;*

As in the SE algorithm, for a given partition (V_1, V_2) of a directed graph $G(V, E)$, let *gain*(i) be the reduction in *cost*(V_1, V_2) when vertex i is moved from its current subset to the other subset of (V_1, V_2). Function *improve_and_cluster* performs the following steps:

(1) Free all vertices and set $c = 0$.
(2) *Forward move*: Rank subsets V_1 and V_2 according to: (a) number of elements, (b) in case of tie, according to the largest gain of a vertex in the subset, and (c) in case of tie, randomly by tossing a balanced coin. Call the subset of larger rank F, and call the other subset T. Move a sequence of vertices $f_1, \ldots, f_k$ from F to T using a highest-gain-first scheme until either F is out of free vertices or a vertex with strictly positive gain is moved. Lock $f_1, \ldots, f_k$ in T, set $c = c + 1$, and let $L_c = \{f_1, \ldots, f_k\}$.
(3) *Restore balance*: If $|V_1| \leq \alpha|V|$ and $|V_2| \leq \alpha|V|$ then do nothing. Otherwise, call F the larger of V_1 and V_2, and call the other subset T. Move a sequence of vertices $r_1, \ldots, r_j$ from F to T using highest-gain-first scheme until either F is out of free vertices or the current size of F is no more than $\alpha|V|$. Lock $r_1, \ldots, r_j$ in T, set $c = c + 1$, and let $L_c = \{r_1, \ldots, r_j\}$.
(4) *Save*: If the current partition is an improved bisection then save it.
(5) *Repeat*: If there are still free vertices then go to Step 2.
(6) *Clustering*: Cluster G by coalescing together some subsets of $G[L_1], G[L_2], \ldots, G[L_c]$.

Note that if (P_1, P_2) is the best bisection found by *improve_and_cluster*, then subsets $L_1, L_2, \ldots,$ and L_c are mutually disjoint, and for each $1 \leq j \leq c$, either $L_j \subseteq P_1$ or $L_j \subseteq P_2$. Therefore vertices that are coalesced together in Step 6 belong to the same subset in the best bisection (P_1, P_2). Therefore, (P_1, P_2) clusters to a bisection (V_1', V_2') of the same cost in the clustered graph.

Given a partition (V_1, V_2) of a graph $G(V, E)$, let $u_1, \ldots, u_k$ be a sequence of vertices moved, either from V_1 to V_2 or from V_2 to V_1, using a highest-gain-first scheme. Call the last vertex u_k the tip of the sequence, and call the sequence a cone if:

(1) $u_1, \ldots, u_{k-1}$ had non-positive gain at the time of their transfer.
(2) $g(u_k) > 0$ when u_k is moved.

It can be shown that if $u_1, \ldots, u_k$ is a cone moved from V_1 to V_2 using a highest-gain-first scheme and if $k > 1$, then there exist $1 \leq i < k$ such that $u_i \rightarrow u_k$ is an arc in the current graph processed by *improve_and_cluster*. Similarly, if $u_1, \ldots, u_k$ is a cone moved from V_2 to V_1 using a highest-gain-first scheme and if $k > 1$, then there exist $1 \leq i < k$ such that $u_k \rightarrow u_i$ is an arc in the current graph processed by *improve_and_cluster*. The proof of this is similar to a proof of a similar result in Saab (1995) and will not be repeated here.

The forward move step of *improve_and_cluster* terminates either because F is out of free vertices or when the last vertex moved has a strictly positive gain. Therefore, all the L_i's computed in this step, with the possible exception of the last one, are cones. Each of the L_i's generated in the restore balance step of *improve_and_cluster* may be subdivided into several cones in a straightforward way. For example, suppose $L_i = \{u_1, \ldots, u_{20}\}$ and that u_2, u_7, and u_9 are the only vertices with strictly positive gains at the time of their transfer, then L_i has in it three cones $\{u_1, u_2\}$, $\{u_3, \ldots, u_7\}$, and $\{u_8, u_9\}$. Let $K_1, \ldots, K_d$ be all the cones with more than one vertex extracted from $L_1, \ldots, L_c$ in the above manner. Step 6 of *improve_and_cluster* clusters a subset of vertices in each of the cones $K_1, \ldots,$ and K_d as follows. For $1 \leq i \leq d$, let t_i be the tip of K_i. If vertices of K_i were moved from V_1 to V_2, then cluster the subset $\{t_i\} \cup \{u : u \in K_i$ and $u \rightarrow t_i$ is an arc$\}$. If vertices of K_i were moved from V_2 to V_1, then cluster the subset $\{t_i\} \cup \{u : u \in K_i$ and $t_i \rightarrow u$ is an arc$\}$.

Other clustering approaches are possible, However in Saab (1995), it was concluded that clustering is most effective when the size of each of the clustered subset is small to avoid entrapment in local minima, and when the number of clustered subset is large enough for efficient execution time. The above clustering approach achieves both goals. See Saab (1995) for more details about our method of combining clustering and iterative improvement.

7. Generation of test cases

In empirical testing of heuristic methods for FAS, it is desirable to have input graphs for which the optimal number of feedback arcs is known. We describe here one method to generate such graphs. Since the maximum number of arc disjoint cycles is a lower bound on the size of a minimum feedback arc set, our method constructs a graph in which there exists a feedback arc set with size equal to a number of randomly planted arc disjoint cycles. Therefore, the number of planted cycles is the size of a minimum feedback set. The cycles

will be planted by exploiting the relationship between vertex ordering and feedback arc sets. Essentially, each of the planted cycles has exactly one leftward arc with respect to a randomly chosen vertex ordering, and is randomly planted. The number of leftward arcs with respect to this chosen vertex ordering is therefore equal to the number of the planted arc disjoint cycles. Since the set of leftward arcs with respect to a vertex ordering is a feedback arc set, the number of planted cycles is also the size of a minimum feedback arc set. The resulting graph may contain multiple arcs between the same pair of vertices in the two possible directions.

Our algorithm takes as input the number of vertices n, the size of a minimum feedback arc set f, a lower bound on the total number of arcs m, and a seed s for the random number generator. A vertex ordering is specified by a permutation $\pi : N \to N$, where $N = \{1, \ldots, n\}$. Thus $\pi(i)$ is the vertex in position i. Let *randint*(l, h) be a function that returns a random integer in the interval $[l, h]$. A pseudo-code of the algorithm for the generation of test cases for FAS, is as follows:

```
read n, f, m, and s;
seed random number generator with s;
generate a random permutation π of n elements;
num_arc := 0;
repeat f times
        i := randint(2, n);
        j := randint(1,i − 1);
        add arc π(i) → π(j) to graph;   {leftward arc}
        num_arc:= num_arc + 1;
        while j ≠ i do {complete a cycle with rightward arcs}
                k := randint(j + 1,i);
                add arc π(j) → π(k) to graph;
                num_arc := num_arc + 1;
                j := k;
if num_arc < m then    {add additional rightward arcs}
        repeat m-num_arc times
                i := randint(1,n − 1);
                j := randint(i + 1,n);
                add arc π(i) → π(j) to graph;
```

8. Experimental results

Eighty random graphs on 100, 500, 1000, and 2000 vertices were generated such that the average vertex outdegree is 2, 3, 4, 8, and 16. Forty of these graphs were generated as follows: for each pair of vertices $i \neq j$, add arc $i \to j$ to graph with probability p. In a random graph on n vertices generated in this manner, the expected vertex outdegree is $(n-1)p$. So, for each n, p was chosen to yield a graph with the desired value of average outdegree. The other forty random graphs were generated using the method of the previous section.

Table 1. Results on random graphs.

						Min		Mean		Max	
Deg	Node	Least	ELS	ESL	MESL	DC	SE	DC	SE	DC	SE
2											
	100	14	0.0	14.3	21.4	14.3	28.6	18.6	28.6	21.4	28.6
	500	37	16.2	35.1	37.8	0.0	5.4	10.0	12.2	18.9	24.3
	1000	80	21.2	26.2	33.8	0.0	5.0	7.9	13.4	16.2	18.8
	2000	140	28.6	33.6	30.7	0.0	7.1	8.0	12.8	14.3	21.4
3											
	100	39	7.7	15.4	25.6	0.0	0.0	5.9	6.4	7.7	15.4
	500	150	22.0	35.3	18.7	0.0	3.3	3.4	9.7	8.0	14.0
	1000	278	24.1	25.2	22.3	0.0	6.1	4.5	8.5	7.9	15.1
	2000	531	23.7	22.2	19.0	0.0	2.3	1.5	5.1	3.2	8.3
4											
	100	69	23.2	31.9	23.2	1.4	0.0	8.3	6.2	13.0	11.6
	500	282	19.9	25.5	21.3	1.1	0.0	3.7	5.3	10.6	8.2
	1000	567	17.8	19.0	18.0	0.0	3.4	2.8	6.4	5.8	11.5
	2000	1059	17.3	18.7	15.7	0.0	2.5	1.1	3.6	2.7	5.1
8											
	100	212	11.8	9.4	9.9	0.0	0.5	3.5	4.2	7.1	10.8
	500	981	12.0	8.7	9.3	0.0	1.4	1.2	2.9	2.4	4.2
	1000	1949	9.8	8.5	8.3	0.0	0.5	0.4	2.1	1.3	5.4
	2000	3719	13.6	12.6	12.5	0.0	2.1	0.7	4.9	1.3	7.1
16											
	100	555	3.6	6.7	6.7	0.0	4.1	3.9	5.9	7.6	8.5
	500	2621	5.3	4.2	4.1	0.0	0.6	1.2	3.9	6.2	6.4
	1000	5079	5.9	7.3	6.4	0.0	2.0	0.5	3.9	1.0	5.4
	2000	10145	6.4	7.5	7.2	0.0	1.9	0.2	3.6	0.6	6.4

Therefore, for each of these graphs, the optimal number of feedback arcs is known. For each value of the number of vertices n, we selected the lower bound on the number of edges to be δn, where δ is the desired average outdegree. We also selected the size of the optimal feedback arc set small enough so that the resulting random graph would have average degree equal to δ.

We ran our divide-and-conquer algorithm using both stochastic evolution (SE) and dynamic clustering (DC) to solve GB. Since both stochastic evolution and dynamic clustering start with a random initial bisection, we ran our algorithm 10 times, and computed, in each case, the minimum, maximum, and mean of the size of the returned feedback arc set. We also implemented algorithm ELS (1993), ESL (1989), and a modified version of ESL (MESL). ESL is, like our method, a divide-and-conquer approach for FAS. However, unlike our method, it does not use graph decomposition into strongly connected components.

Table 2. Results on random graphs with known optimal number of feedback arcs.

						Min		Mean		Max	
Deg	Node	Opt	ELS	ESL	MESL	DC	SE	DC	SE	DC	SE
2											
	100	10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
	500	30	3.3	10.0	13.3	0.0	0.0	3.3	2.0	10.0	6.7
	1000	80	20.0	13.8	6.2	2.5	3.8	6.4	7.1	10.0	11.2
	2000	120	15.0	16.7	10.0	0.8	0.8	2.7	3.6	4.2	5.8
3											
	100	40	32.5	20.0	12.5	7.5	2.5	10.5	10.2	17.5	22.5
	500	150	51.3	46.0	58.7	7.3	8.7	11.9	13.6	21.3	20.0
	1000	280	57.1	60.4	63.6	7.5	8.9	12.2	13.5	20.4	19.6
	2000	500	88.6	62.2	51.6	7.8	11.0	9.3	13.7	11.4	17.2
4											
	100	70	35.7	31.4	30.0	1.4	2.9	6.0	7.7	10.0	10.0
	500	270	69.6	64.4	69.3	4.4	6.3	8.3	13.2	12.6	24.1
	1000	570	75.6	77.7	82.3	4.7	7.2	8.8	10.8	20.9	18.1
	2000	1000	97.5	100.2	102.1	4.7	4.8	6.8	13.9	8.3	30.5
8											
	100	150	48.0	78.7	74.0	0.0	0.0	1.8	4.8	4.7	21.3
	500	650	88.0	97.4	95.7	0.9	1.4	1.9	18.2	3.7	43.1
	1000	1250	94.3	101.4	106.2	1.7	1.8	2.6	14.4	3.4	39.2
	2000	2000	144.0	149.1	154.1	1.2	1.8	2.3	9.6	4.9	23.2
16											
	100	350	45.7	63.4	69.1	1.1	0.6	4.7	5.4	8.9	8.6
	500	1200	123.8	134.9	133.6	0.3	0.4	0.9	1.2	1.2	3.3
	1000	2400	113.8	137.2	139.7	0.0	0.2	0.7	12.3	1.5	60.0
	2000	4000	172.8	191.2	189.4	0.1	0.1	0.5	6.9	1.7	27.4

MESL is a modification of ESL to include such decomposition. Let *ESL_bisect*(G) be a function that takes a directed graph $G(V, E)$ as input and returns a partition (V_1, V_2) of G according to the divide step of ESL. By replacing function *bisect* in our method, by function *ESL_bisect*, we get an implementation of MESL. All the implemented algorithms do not guarantee a minimal feedback arc set. However, we use, for all the algorithms, a post-processing procedure to transform the feedback set found into a minimal one. This is achieved by checking, for each arc in the feedback set, if it is in a cycle with arcs not in the feedback set, and if not so, it is removed from the feedback set.

All algorithms were implemented in the C programming language and were ran on a Sun Sparc 1+ workstation. All of them were reasonably fast, the largest CPU time per run was at most 7 minutes. In terms of speed, the linear time algorithm ELS is the clear winner followed by ESL and MESL running at about the same speed, followed by SE and DC

with execution time dependent on the particular instance. For some cases SE was a little faster than DC, and for other case SE was a little slower than DC. Performance of these 5 algorithms is summarized in Tables 1 and 2. Table 1 shows the least size of all feedback arc sets found by any of the 5 algorithms. Results of the algorithms are then expressed as percentages over the least size. Table 2 shows results on graphs generated using the method of the previous section. It shows the optimal number of feedback arc sets, and results of the algorithms are then expressed as percentages over this optimal number. Results in Tables 1 and 2 shows DC and SE as clear winners in terms of performance with DC yielding the best performance. The size of the smallest feedback arc set found in 10 runs of DC and SE was never more than 8% and 11% above the optimal size for graphs listed in Table 2, and was never more than 1.4% and 12.5% above the least known size for graphs listed in Table 1. ELS, ESL, and MESL mostly do not perform as well as DC or SE. ELS, ESL, and MESL performed the best on graphs with average degree 16 in Table 1.

9. Conclusion

We presented a fast and effective divide-and-conquer approach for the feedback arc set problem. The divide step is performed by solving a minimum bisection problem using either stochastic evolution (SE) or dynamic clustering (DC). Empirical results showed good performance using both SE and DC with DC being the more effective of the two approaches. However, SE has an advantage over DC since it is a simpler approach to implement. Both SE and DC are efficient and can be used on large graphs.

Acknowledgment

I would like to thank Dr. Peter Eades and Dr. Xuemin Lin for providing copies of Eades and Lin (to appear) and Eades, Smith and Lin (1989).

References

Berger, B. and P.W. Shor. (1990). "Approximation Algorithms for the Maximum Acyclic Subgraph Problem." In *Proc. First ACM-SIAM Symp. Discrete Algorithms*. pp. 236–243.

Calazans, Ney, et al. (1992). "Advanced Ordering and Manipulation Techniques for Binary Decision Diagrams." In *European Design Automation Conference*, Brussels, Belgium. pp. 452–457.

Cormen, T., C. Leiserson, and R. Rivest. (1990). *Introduction to Algorithms*. Cambridge, Mass: The MIT Press.

De Souza, C., R. Keunings, L. Wolsey, and O. Zone. (1994). "A New Approach to Minimizing the Frontwidth in Finite Element Calculations." *Computer Methods in Applied Mechanics and Engineering* 111(3/4), 323–334.

Eades, P. and X. Lin. (To appear). "A Heuristic for the Feedback Arc Set Problem." *Australasian J. of Combinatorics*.

Eades, P., X. Lin, and W.F. Smyth. (1993). "A Fast and Effective Heuristic for the Feedback Arc Set Problem." *Inf. Proc. Lett.* 47, 319–323.

Eades, P., W.F. Smyth, and X. Lin. (1989). "Heuristics for the Feedback Arc Set Problem." Tech. Rept. 1, School of Computing Science, Curtin University of Technology, Perth, Western Australia.

Ford, L. and D. Fulkerson. (1962). *Flows in Networks*. Princeton, NJ: Princeton University Press.

Garey, M.R. and D.S. Johnson. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness.* New York, NY: W. H. Freeman and Company.

Goldberg, A. and R. Tarjan. (1986). "A New Approach to the Maximum Flow Problem." In *Proc. Eight ACM Symp. on Theory of Computing*. pp. 136–146.

Lempel, A. and I. Cederbaum. (1966). "Minimum Feedback Arc and Vertex Sets of a Directed Graph." *IEEET Trans. Circuit Theory* CT-13(4), 399–403.

Lucchesi, C.L. (1976). "A Minimax Equality For Directed Graphs." Doctoral Thesis, University of Waterloo, Ontario, Canada.

Lucchesi, C.L. and D.H. Younger. (1978). "A Minimax Theorem For Directed Graphs." *J. London Math. Soc.* 2(17), 369–374.

Mitra, B., P. Ranjan Panda, and P. Pal Chaudhuri. (1993). "Estimating the Complexity of Synthesized Designs from FSM Specifications." *IEEE Design & Test of Computers* 10(1), 30–35.

Cherabuddi, Raghava V. and Magdy A. Bayoumi. (1994). "Automated System Partitioning for Synthesis of Multi-Chip Modules." In *Fourth Great Lakes Symposium on VLSI, Design Automation of High Performance VLSI Systems*. pp. 21–25.

Ramachandran, V. (1988). "Finding Minimum Feedback Arc Set in Reducible Flow graphs." *J. Algorithms* 9, 299–313.

Saab, Y. (1990). "Combinatorial Optimization by Stochastic Evolution with Applications to the Physical Design of VLSI Circuits". Doctoral Thesis, University of Illinois at Urbana-Champaign.

Saab, Y. (1993). "Post-Analysis-Based Clustering Dramatically Improves the Fiduccia-Mattheyses Algorithm." In *European Design Automation Conference*, Hamburg, Germany. pp. 22–27.

Saab, Y. (1995). "A Fast and Robust Network Bisection Algorithm." *IEEE Trans. Computers* 44 (7), 903–913.

Saab, Y. and V. Rao. (1991). "Combinatorial Optimization by Stochastic Evolution." *IEEE Trans. Computer-Aided Design* 10(4), 525–535.

Seshu, S. and M.B. Reed. (1961). *Linear Graphs and Electrical Networks*. Reading, Mass: Addison Wesley.

Slater, P. (1971). "Optimal Ranking of Tournaments." *Networks* 1, 135–138.

Unger, S.H. (1957). "A Study of Asynchronous Logical Feedback Networks." Tech. Rept. 320, Research Lab. of Electronics, MIT, Cambridge, Mass.

Younger, D.H. (1963). "Minimum Feedback Arc Sets for a Directed Graph." *IEEE Trans. Circuit Theory* CT-10, 238–245.