FORTRAN SUBROUTINES FOR COMPUTING APPROXIMATE SOLUTIONS OF FEEDBACK SET PROBLEMS USING GRASP

PAOLA FESTA, PANOS M. PARDALOS, AND MAURICIO G.C. RESENDE

ABSTRACT. We describe FORTRAN subroutines for approximately solving the feedback vertex and arc set problems on directed graphs using a Greedy Randomized Adaptive Search Procedure (GRASP). The algorithms are described in detail. Implementation and usage of the package is outlined and computational experiments are reported illustrating solution quality as a function of running time. The source code can be downloaded from the URL `http://www.research.att.com/~mgcr/src/gfsp.tar.gz`.

1. INTRODUCTION

Let $G=(V,E)$ be a graph with vertex set V and arc set E. A *path* P in G connecting vertex u to vertex v is a sequence of arcs $e_1,\cdots,e_r$ in E, such that $e_i=(v_i,v_{i+1})$, $i=1,\cdots,r$ with $v_1=u$ and $v_{r+1}=v$. A *cycle* C in G is a path $C=(v_1,\cdots,v_r)$, with $v_1=v_r$. A *feedback vertex (arc) set* of G is a subset of vertices (arcs) $S\subseteq V$ ($S\subseteq E$) such that each cycle in G contains at least one vertex (arc) in S. Let w be a function that assigns a nonnegative weight to each vertex (arc) of G. Then the weight of a feedback vertex (arc) set is the sum of the weights of its vertices (arcs), and a *minimum feedback vertex (arc) set* of a *weighted graph (G,w)* is a feedback vertex (arc) set of G of minimum weight.

This kind of NP-hard problem is also known as the *hitting cycle problem*, since one must hit every cycle in C. In addition to the *minimum feedback vertex (arc) set problem*, it also generalizes a number of problems, the *subset minimum feedback vertex (arc) set problem* and the *graph bipartization problem*, in which one must remove a minimum-weight set of vertices so that the remaining graph is bipartite.

A general NP-hardness proof for all *feedback set problems* restricted to planar graphs has been given in [12]. These results apply to the planar bipartization problem, the planar (directed, undirected, or subset) feedback vertex set problems, already proved to be NP-hard [7, 6]. Furthermore, it is NP-complete for planar graphs with no in-degree or out-degree exceeding three [7], general graphs with no in-degree or out-degree exceeding two [7], and arc-directed graphs [7].

The feedback vertex (arc) set problem has found applications in many fields, including deadlock prevention [11], program verification [10], and Bayesian inference [1]. Therefore, it is natural that in the past few years there have been intensive efforts on approximation algorithms for these kinds of problems. A recent survey of feedback set problems can be found in Festa, Pardalos, and Resende [5].

Although the approximation algorithms guarantee a solution of a certain quality, for many practical real world cases, heuristic methods can lead to better solutions in a reasonable amount of CPU time. One such heuristic is the greedy randomized adaptive search

Date: May 1999.

Key words and phrases. Combinatorial optimization, feedback set problems, graph bipartization, local search, GRASP, FORTRAN subroutines.

AT&T Labs Research Technical Report: 99.6.1.

```
procedure gfvs(V,E,maxitr,S*)
1   S* = ∅; V^0 = V; E^0 = E;
2   do k = 1,··· ,maxitr →
3       V = V^0; E = E^0;
4       α = UNIF[0,1];
5       ConstructGreedyRandomizedSolution(V,E,α,S);
6       LocalSearch(V,E,S);
7       if (|S| < |S*|) →
8           S* = S;
9       fi;
10  od;
end GRASP;
```

FIGURE 1. A GRASP algorithm for feedback vertex set

procedure (GRASP) introduced by Feo and Resende [4, 3] and used by Pardalos, Qian, and Resende [9] to find approximate solutions of large instances of the feedback vertex set problem.

GRASP [3] is an iterative sampling method for finding approximate solutions to combinatorial optimization problems. GRASP iterations are repeated, each iteration finding an approximate solution to the problem. The best solution found, over all GRASP iterations, is returned by the method as the GRASP solution. Each GRASP iteration is made up of two phases: a construction phase and a local search phase, also known as a local improvement phase. During the construction phase a feasible solution is iteratively constructed. One element at a time is randomly chosen from a *Restricted Candidate List* (RCL), whose elements are sorted according to some greedy criterion, and is added to the solution begin built. The randomization used in the algorithm makes it unlikely that the greedy choice is always selected during construction. Therefore, the construction phase solution is rarely the greedy solution. In the second phase, the neighborhood of the constructed solution is searched for an improved solution.

In this paper, we describe `gfvs` and `gfas`, two sets of FORTRAN subroutines that apply GRASP to find approximate solutions of the feedback vertex set and the feedback arc set problem, respectively. The paper is organized as follows. The algorithms implemented in `gfvs` and `gfas` are described in Section 2.1 and in Section 2.2, respectively. In Section 3, we describe the design and implementation of `gfvs` and `gfas`, two sets of FORTRAN subroutines distributed with the packages. Computational testing is presented in Section 5.

2. THE ALGORITHMS

In this section, we describe the greedy randomized adaptive search procedures for feedback vertex set and feedback arc set problems.

2.1. A GRASP procedure for the feedback vertex set problem.

The GRASP algorithm for the feedback vertex set problem implemented in `gfvs` is the procedure proposed by Pardalos, Qian, and Resende [9]. The pseudo-code for `gfvs` is shown in Figure 1.

As sketched in Section 1, GRASP is a multi-start method characterized by two phases: a construction phase and a local search phase. The main loop of the GRASP consists of lines 2–10. A solution is generated in line 5 with a local search taking place in line 6, while the best solution found is updated in lines 7–9. During both the construction and

```
procedure in0out0(V,E,S)
  for v ∈ V →
    if (|in(v)| = 0 or |out(v)| = 0) →
      S = S ∪ {v};
      V = V \ {v};
      E = E \ {(x,y) ∈ E | x = v  or  y = v};
    fi;
  rof;
end in0out0;
```

FIGURE 2. Solution preserving reduction `in0out0`

```
procedure in1(V,E,out)
  for v ∈ V →
    if (|in(v)| = 1) →
      for (u,v) ∈ E →
        V = V \ {v};
        E = E ∪ {(u,w) ∈ E | w ∈ out(v)} \ {(v,w) ∈ E | w ∈ out(v)};
        out(u) = out(u) ∪ out(v);
      rof;
    fi;
  rof;
end in1;
```

FIGURE 3. Solution preserving reduction `in1`

```
procedure out1(V,E,in)
  for v ∈ V →
    if (|out(v)| = 1) →
      for (v,u) ∈ E →
        V = V \ {v};
        E = E ∪ {(w,v) ∈ E | w ∈ in(v)} \ {(w,v) ∈ E | w ∈ in(v)};
        in(v) = in(v) ∪ in(u);
      rof;
    fi;
  rof;
end out1;
```

FIGURE 4. Solution preserving reduction `out1`

the local search phase the GRASP uses solution preserving reduction techniques. In these procedures, vertices and arcs are removed from G in such a way that the reduced graph and original graph have the same feedback vertex set. Four solution preserving reductions are used. Let S be a feedback vertex (arc) set of G and for each vertex $i \in V$ let define $in(i) = \{j \mid (j,i) \in E\}$ and $out(i) = \{j \mid (i,j) \in E\}$, then the solution preserving reductions are shown in Figures 2– 5.

```
procedure loop(V,E,S)
   for (v,v) ∈ E →
      S = S ∪ {v};
      V = V \ {v};
      E = E \ {(v,u) ∈ E  or  (u,v) ∈ E};
   rof;
end loop;
```

FIGURE 5. Solution preserving reduction `loop`

```
procedure ReduceInstanceSize(V,E,S)
   in0out0(V,E,S);
   in1(V,E);
   out1(V,E);
   loop(V,E,S);
   for (at least one reduction succeeds) →
      in0out0(V,E,S);
      in1(V,E);
      out1(V,E);
      loop(V,E,S);
   rof;
end ReduceInstanceSize;
```

FIGURE 6. Solution preserving reductions

```
procedure ConstructGreedyRandomizedSolution(V,E,α,S)
   S = ∅;
   ReduceInstanceSize(V,E,S);
   do k = 1,··· ,n →
      MakeRCL(α);
      s = SelectIndex();
      S = S ∪ {s};
      UpdateGraph(V,E);
      ReduceInstanceSize(V,E,S);
   od;
end ConstructGreedyRandomizedSolution;
```

FIGURE 7. GRASP construction phase pseudo-code

The above reductions are implemented in the procedure `ReduceInstanceSize`, outlined in Figure 6. For detailed analysis of these reduction procedures, see also Levy and Lowe [8].

To describe the construction phase, one needs to provide an adaptive greedy function, a construction mechanism for the RCL, and a probabilistic selection procedure. These three components form an iterative procedure that constructs a solution, one vertex at a time, biased by the adaptive greedy function. During the construction phase a feasible solution is iteratively constructed. One element at a time is randomly chosen from a *Restricted*

```
procedure LocalSearch(V, E, S)
1   flag = 1;
2   while (flag) →
3       flag = 0;
4       for i = 1, ..., |S|  →
5           V' = V ∪ {S \ {s_i}};
6           E' = E ∪ {(v, w) ∈ E | v or w ∈ {S \ {s_i}};
7           ReduceInstanceSize(V', E', S);
8           if (V' = ∅) →
9               S = S \ {s_i};
10              flag = 1;
11              break;
12          fi;
13      rof;
14  elihw;
end LocalSearch;
```

FIGURE 8. GRASP local search phase pseudo-code

Candidate List (RCL), whose elements are sorted according to some greedy criterion, and is added to the feedback vertex set being built and removed from the graph with all its incident arcs. Since the computed solution, in general, may not be locally optimal with respect to the adopted neighborhood definition, the local search phase tries to improve it. These two phases are iterated and the best solution found is kept as an approximation of the optimal solution. Figure 7 shows the pseudo-code for the construction phase of GRASP. The main loop in lines 3–9 is repeated at most n times, as at most $n = |V|$ vertices can be selected to be inserted in the cutset S. A restricted candidate list (RCL) is computed in line 4 through the procedure `MakeRCL`. Following Pardalos, Qian, and Resende [9], the greedy function used in the construction procedure is

$$G(v) = |in(v)| \cdot |out(v)|,$$

that favors the balance between the in- and out-degrees of node v. Let

$$\underline{G} = \min_{v \in V} G(v) \quad \text{and} \quad \overline{G} = \max_{v \in V} G(v),$$

and let α ($0 \leq \alpha \leq 1$) be a real number chosen at random (in `gfvs`) using the uniform distribution. Then a restricted candidate list for this problem is the set of vertices

$$\text{RCL} = \{v \in V \mid G(v) \geq \underline{G} + \alpha \cdot (\overline{G} - \underline{G})\}.$$

The vertex selection in the line 5 of the GRASP construction phase is random, restricted to vertices belonging to the RCL. In line 6 the feedback vertex set is updated to include the selected vertex s, which is removed from the graph together with all its incident arcs in line 7. The solution preserving reductions are applied on the graph in line 8.

Once a cutset S is generated by the construction procedure, local search eliminates its redundant elements, resulting in a minimal cutset. The pseudo-code of the local search heuristic is presented in Figure 8, where s_i is the i-th element of S.

For any given cutset, the local search heuristic checks whether each vertex of the cutset is redundant. This is done, in lines 2–10, by excluding each vertex s_i from S and applying

```
procedure fas2fvs(G = (V,E), G' = (V',E'))
V' = E;
for each arc e_i = (v_i, v_j) ∈ E →
    if (there exists e_j = (v_j, v_k) ∈ E) →
        E' = E' ∪ {(e_i, e_j)};
    fi;
rof;
end fas2fvs;
```

FIGURE 9. Procedure to reduce a feedback arc set problem into a feedback vertex set problem

TABLE 1. The distribution

makefile	`Makefile`	
drivers	`driver-gfvs.f`	`driver-gfas.f`
subroutines	`gfvs.f`	`gfas.f`
sample input	`sample.dat`	
sample output	`sample-gfvs.out`	`sample-gfas.out`
instructions	`READ.ME`	

the reduction procedures to the resulting graph. If in a given iteration, no reduction heuristic can be successfully applied, then the reduced graph is cyclic and s_i is not redundant. Otherwise, the heuristic will return an empty reduced graph, indicating that vertex s_i is redundant and can be dropped from the current cutset.

2.2. A GRASP procedure for solving the feedback arc set problem.

Feedback vertex and feedback arc set problems are reducible to each other. In all reductions, there is a one-to-one correspondence between feasible solutions and their corresponding costs. Therefore, an approximate solution to one problem can be translated into an approximate solution of the corresponding translated problem. To solve feedback arc set problems, an $O(|E|)$ time procedure, due to Even, Naor, Schieber, and Sudan [2], is applied to translate the instance of the feedback arc set problem into an equivalent feedback vertex set problem, which can be solved by `gfvs`.

Given a graph $G = (V,E)$ on which a feedback arc set problem is defined, `gfvs` is applied to the graph $G' = (V',E')$ defined by the procedure `fas2fvs` described in Figure 9. For each arc in G there is a corresponding vertex in G'. For each pair of arcs in G for which the head of the first arc is the tail of the second, there is an arc in G' whose tail is the vertex corresponding to the first arc in G and whose head is the vertex corresponding to the second arc.

3. DESIGN AND IMPLEMENTATION OF THE SUBROUTINES

We followed several design guidelines in the implementation of `gfvs` and `gfas`. The codes are written in ANSI standard FORTRAN 77 and are intended to run without modification on UNIX platforms (it should run on other environments without modification). There are no `common` blocks in the codes and all arrays and variables are passed as parameters.

The distribution consists of nine files which are listed in Table 1.

```
10 22

1 10    1  9    7  1    1  5    1  2    2  8    7  2    2  4
2  3    3 10    3  6    3  5    8  4    4  5    5  6    6  8
6  7    7  9    8  7    10 8    9  8    9 10
```

FIGURE 10. Input file of feedback set instance.

```
GRASP for Feedback Set Problem----------------

    itr =       1    cutset size =      2 ***

    Execution terminated with no error.

GRASP solution--------------------------------

 size of feedback vertex cutset:            2

    iteration best cutset found:            1

seed at start of best iteration:   2065020212

smallest vertex cutset:

vertex:       6
vertex:       8

----------------------------------------------

Stop - Program terminated.
```

```
GRASP for Feedback Set Problem----------------

    itr =       1    cutset size =      4 ***
    itr =       2    cutset size =      3 ***

    Execution terminated with no error.

GRASP solution--------------------------------

    size of feedback arc cutset:            3

    iteration best cutset found:            2

seed at start of best iteration:   1347579962

smallest arc cutset:

arc:    5   6
arc:    8   7
arc:    2   3

----------------------------------------------

Stop - Program terminated.
```

FIGURE 11. Sample output of `driver_gfvs.f` and `driver_gfas.f` for sample instance

4. USAGE OF THE SUBROUTINES

The subroutines in files `gfvs.f` and `gfas.f` compute an approximate solution of the feedback vertex and arc set problems, respectively. The user interface with them is subroutine `gmpsg`, which must be called from a driver program. In addition to a number of auxiliary arrays, the driver passes the following representation of the input graph:

`n`: Number of nodes in graph,
`m`: Number of arcs in graph,
`vtx1`: Integer array where `vtx1(i)` is tail of arc `i`,

vtx2: Integer array where vtx2(i) is head of arc i,

the following array dimensions:

maxv: Dimension of number of nodes declared in the drivers,
maxe: Dimension of number of arcs declared in the drivers.

The sample driver programs for gfvs and gfas included in the distribution (driver-gfvs.f and driver-gfas.f, respectively) are set for problems of dimension $|V| \leq 200$ nodes and $|E| \leq 2000$ arcs. The input/output parameters that define Fortran input/output devices are set to the standard values $\texttt{in} = 5$ and $\texttt{iout} = 6$. These parameters can be set by the user for problems of different dimension or if an alternate input or output device is required.

All variables and arrays needed by subroutines gfvs and gfas are defined in the main programs in files driver-gfvs.f and driver-gfas.f, respectively. Subroutines readp and outsol, also provided in the drivers are examples of code that can be used for input and output, respectively.

Five parameters that control the execution of the algorithm need to be set before the optimization module is called: alpha, the restricted candidate list parameter, whose value is either between 0 and 1, inclusively, or can be set to a negative value to indicate that each GRASP iteration uses a different randomly generated alpha value; look4, a stopping parameter that forces GRASP to stop if a cutset of size at least look4 is found, is an integer that must satisfy $0 \leq \texttt{look4} \leq |V|$ for the feedback vertex set problem or $0 \leq \texttt{look4} \leq |E|$ for the feedback arc set problem; maxitr, the maximum number of GRASP iterations, is an integer such that $\texttt{maxitr} > 0$; prttyp, the output option parameter, is an integer that is set to 0 (silent run, gfvs and gfas do not write anything), 1 (gfvs and gfas print out solution improvements), or 2 (gfvs and gfas print out the solution found in each GRASP iteration); and seed, the pseudo random number generator seed, an integer such that $1 \leq \texttt{seed} \leq 2^{31} - 1$. The default settings for alpha, look4, maxitr, prttyp, and seed are, respectively, -1, 0, 1024, 1, and 270001.

The driver programs call readp, which reads the input data and returns an error condition errcnd. If $\texttt{errcnd} = 0$ is returned by readp, then the drivers call the optimization subroutines which attempt to find a small cutset of the input graph. Subroutines gfvs and gfas return error condition errcnd indicating the status of the optimization. Error condition errcnd can have the following values:

$\texttt{errcnd} = 1$ if a $|V| > \texttt{maxv}$;
$\texttt{errcnd} = 2$ if a $|E| > \texttt{maxe}$;
$\texttt{errcnd} = 3$ if an input node is less than 1 or greater than $|V|$;
$\texttt{errcnd} = 4$ if $\texttt{look4} < 0$ or $\texttt{look4} > |V|$ for the feedback vertex set problem;
$\texttt{errcnd} = 4$ if $\texttt{look4} < 0$ or $\texttt{look4} > |E|$ for the feedback arc set problem;
$\texttt{errcnd} = 5$ if $\texttt{maxitr} < 1$;
$\texttt{errcnd} = 6$ if $\texttt{prttyp} \neq 0, 1, 2$;
$\texttt{errcnd} = 7$ if $\texttt{seed} < 1$ or $\texttt{seed} > 2147483647$.

As an example, consider an instance with 10 vertices and 22 arcs whose input file is shown in Figure 10. Running the driver programs driver_gfvs.f and driver_gfas.f using the default settings this instance produces the outputs shown in The outputs list each iteration for which an improving solution was found and describe the best solutions found by listing the cutsets.

5. Computational Results

In this section, we illustrate the effectiveness of the subroutines by running the feedback set procedures on a subset of the test problems used in [9]. The experiment was limited

TABLE 2. Cutset elements in incumbent solution as a function of GRASP iteration: FVS problem

		GRASP iteration					
nodes	arcs	2	8	32	128	512	2048
50	100	3	3	3	3	3	3
50	150	10	9	9	9	9	9
50	200	19	17	15	14	14	13
50	250	21	21	19	19	17	17
50	300	28	24	23	21	20	20
50	500	30	29	29	28	28	28
50	600	37	36	36	36	33	33
50	700	39	38	38	37	33	33
50	800	40	40	38	38	38	37
50	900	42	41	40	39	38	38
100	200	10	10	9	9	9	9
100	300	24	23	21	20	19	18
100	400	36	29	29	28	26	26
100	500	43	41	41	41	40	38
100	600	53	49	48	47	45	45
100	1000	57	57	56	56	55	55
100	1100	72	71	71	66	66	66
100	1200	75	73	72	71	68	68
100	1300	76	75	73	72	72	71
100	1400	79	78	75	74	73	71
500	1000	45	43	42	40	39	37
500	1500	124	119	114	107	106	105
500	2000	180	180	180	174	166	166
500	2500	234	229	229	221	221	220
500	3000	276	273	266	259	258	257
500	5000	370	356	354	350	341	341
500	5500	378	374	360	360	360	357
500	6000	383	383	377	376	372	371
500	6500	391	388	381	381	379	369
500	7000	408	394	388	383	383	383
1000	3000	244	237	233	233	230	224
1000	3500	319	307	301	296	290	290
1000	4000	357	354	354	348	340	340
1000	4500	428	407	407	407	405	400
1000	5000	469	469	456	456	456	450
1000	10000	733	726	714	714	712	712
1000	15000	817	804	804	803	803	798
1000	20000	860	853	849	847	841	841
1000	25000	889	881	881	877	872	868
1000	30000	912	897	897	892	892	892

TABLE 3. Cutset elements in incumbent solution as a function of GRASP iteration: FAS problem

feedback arc set		feedback vertex set		GRASP iteration					
nodes	arcs	nodes	arcs	2	8	32	128	512	2048
50	100	100	202	6	6	6	6	6	6
50	150	150	462	36	25	25	23	21	21
50	200	200	836	69	68	68	62	61	57
50	250	250	1224	105	100	90	90	90	82
50	300	300	1729	141	138	129	128	128	121
50	500	500	5024	366	355	344	343	332	321
50	600	600	7206	459	445	440	434	430	426
50	700	700	9782	562	540	536	531	517	517
50	800	800	12771	656	654	629	629	621	621
50	900	900	16109	751	739	735	734	726	714
100	200	200	397	16	16	15	14	14	14
100	300	300	908	73	63	63	56	54	49
100	300	400	1564	128	123	119	112	105	98
100	400	500	2419	209	208	204	199	194	184
100	500	600	3641	317	305	291	291	290	278
100	1000	1000	9997	710	710	702	702	696	678
100	1100	1100	12133	816	797	797	797	785	773
100	1200	1200	14482	923	901	899	892	879	878
100	1300	1300	16822	1017	994	992	992	972	972
100	1400	1400	19609	1127	1097	1093	1093	1073	1073
500	1000	1000	2034	90	87	79	79	75	73
500	1500	1500	4570	365	347	340	326	325	311
500	2000	2000	7999	725	704	704	690	690	684
500	2500	2500	12446	1180	1175	1133	1127	1114	1114
500	3000	3000	18034	1637	1637	1631	1595	1595	1590
1000	3000	3000	9233	686	686	686	686	677	677
1000	3500	3500	12254	1061	1047	1026	1021	1012	1007
1000	4000	4000	15997	1451	1399	1369	1369	1362	1362
1000	4500	4500	19945	1842	1784	1784	1779	1759	1759
1000	5000	5000	24739	2347	2273	2273	2238	2238	2238

to half of the test problems having at least 50 and at most 1000 vertices with at most 30000 arcs (a total of 40 instances). The GRASP procedures `gfvs` and `gfas` were applied to each input graph to find approximate solutions to the feedback vertex set (FVS) and feedback arc set (FAS) problems, respectively. The default parameter settings were used, as defined in the driver files of the distribution, i.e. `alpha` $= -1$, `look4` $= 0$, `maxitr` $= 2048$, `prttyp` $= 1$, and `seed` $= 270001$.

The experiments were done on a Silicon Graphics Challenge computer with 20 196 MHz MIPS R10000 processors and 6.1 Gb of main memory. The code was compiled on the SGI Fortran compiler `f77` using the flags `-O3 -r4 -64 -static`. Processes were limited to a single processor. CPU times in seconds were computed by calling the system routine `etime()`.

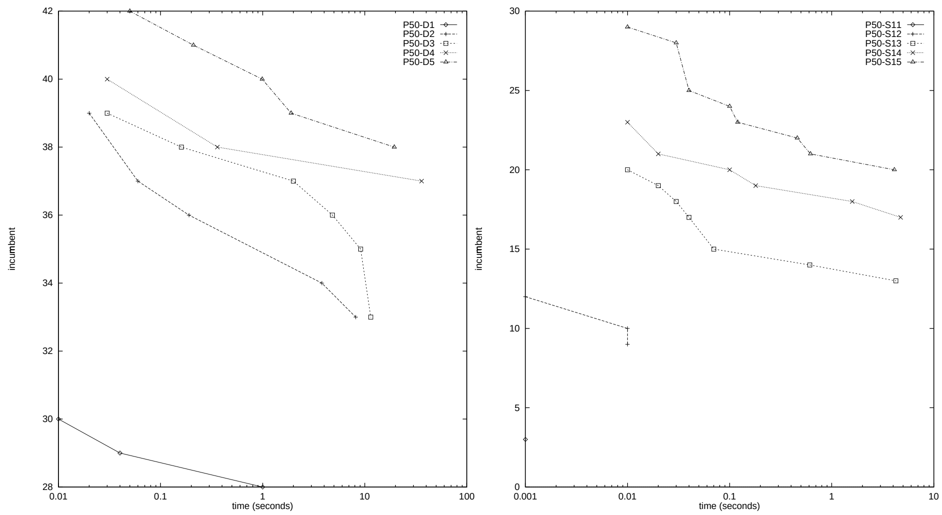

FIGURE 12. FVS: dense and sparse 50-node graphs

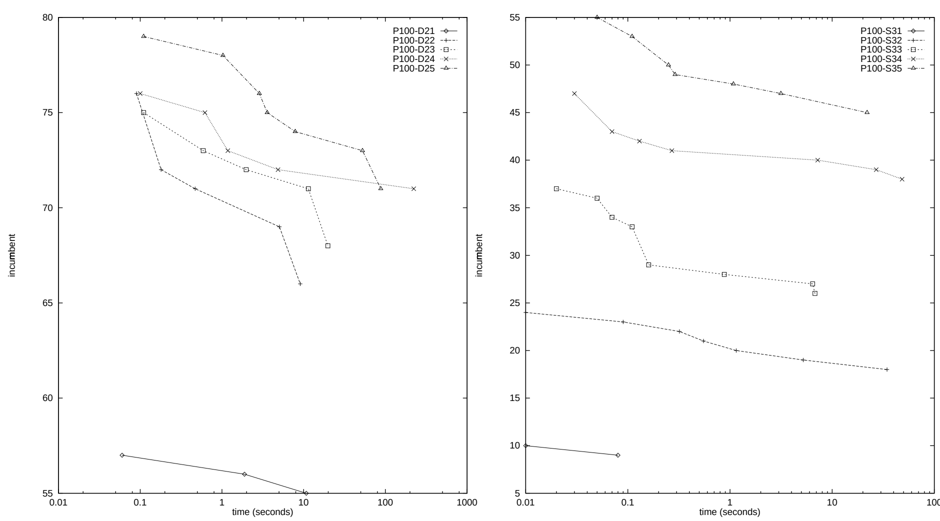

FIGURE 13. FVS: dense and sparse 100-node graphs

For each problem considered in the experiment, Tables 2 and 3 show the size of feedback vertex cutset of the incumbent solution as a function of GRASP iterations. In addition to the size of the input graph, Table 3 also lists the size of the transformed feedback vertex set problem that is solved to solve the feedback arc set problem. The incumbent solutions at iterations 2, 8, 32, 128, 512, and 2048 are listed. Figures 12–18 show all incumbents as a function of execution time (in seconds).

ACKNOWLEDGMENTS

The authors would like to thank Tianbing Qian for implementing a preliminary version of `gfvs`. The research of Paola Festa was supported in part by the Italian Ministry

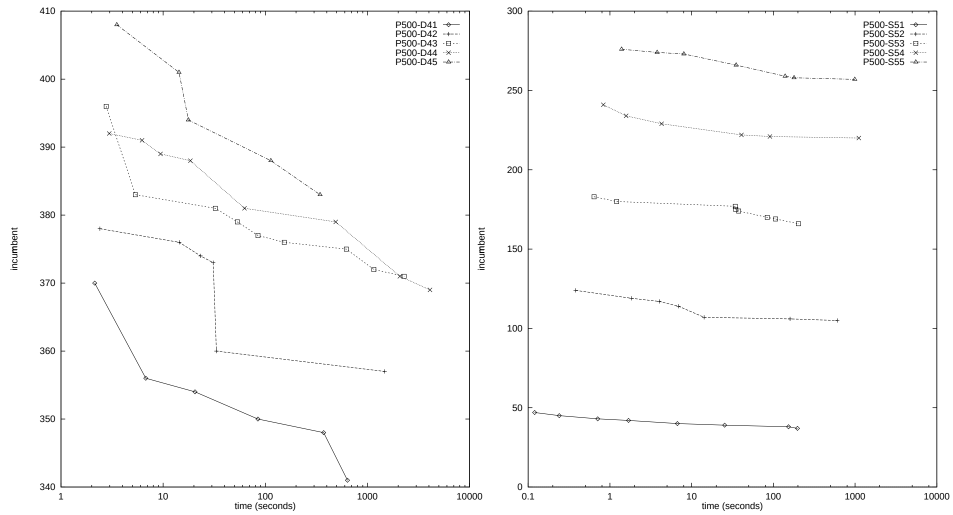

FIGURE 14. FVS: dense and sparse 500-node graphs

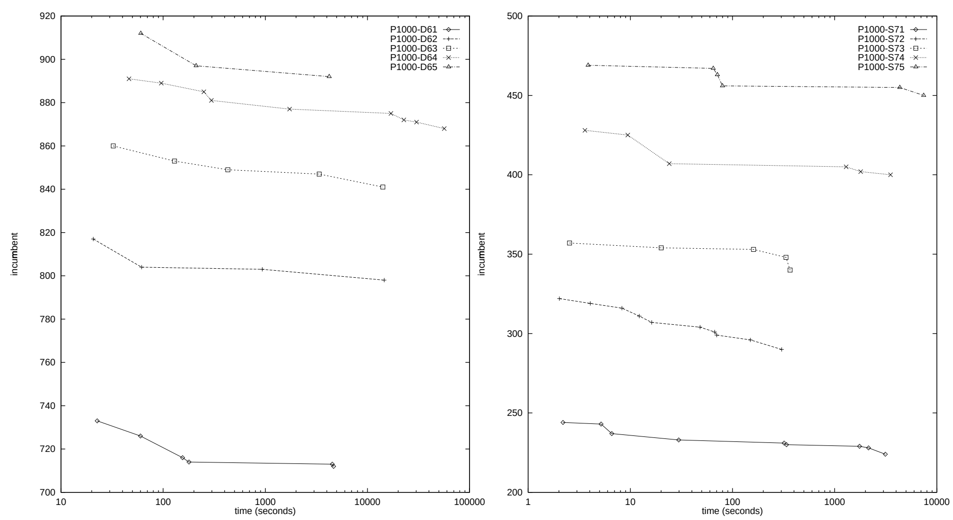

FIGURE 15. FVS: dense and sparse 1000-node graphs

of Scientific Research (MURST) – Research Project MOST 97. The research of Panos Pardalos was supported in part by DIMACS and by National Science Foundation Grant EIA98-72509.

REFERENCES

[1] R. Bar-Yehuda, D. Geiger, J. Naor, and R. M. Roth. Approximation algorithms for the vertex feedback set problem with applications to constraint satisfaction and Bayesian inference. *SIAM J. on Computing*, 27:942–959, 1998.

[2] G. Even, S. Naor, B. Schieber, and M. Sudan. Approximating minimum feedback sets and multicuts in directed graphs. *Algorithmica*, 20:151–174, 1998.

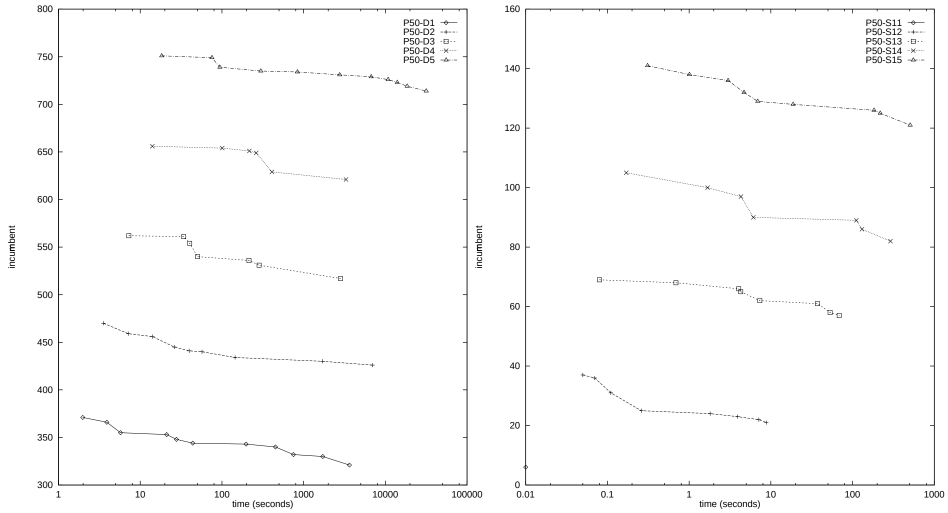

FIGURE 16. FAS: dense and sparse 50-node graphs

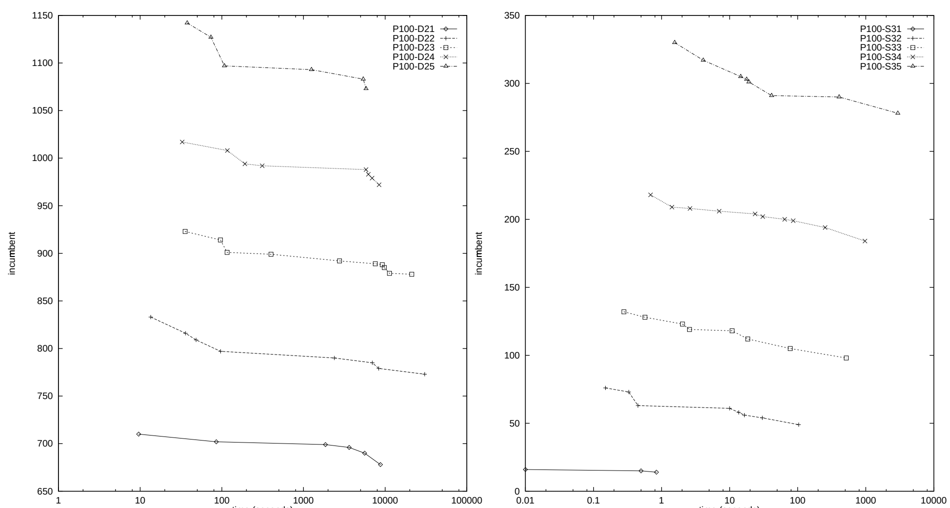

FIGURE 17. FAS: dense and sparse 100-node graphs

[3] T. A. Feo and M. G. C. Resende. Greedy randomized adaptive search procedures. *J. of Global Optimization*, 6:109–133, 1995.

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

[5] P. Festa, P. M. Pardalos, and M. G. C. Resende. Feedback set problems. In *Handbook of Combinatorial Optimization*, volume 4. Kluwer Academic Publishers, 1999.

[6] M. R. Garey and D. S. Johnson. *Computers and intractability – A guide to the theory of NP-completeness*. W.H. Freeman and Company, 1979.

[7] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity Of Computer Computations*, pages 85–103. Plenum Press, 1972.

[8] H. Levy and L. Lowe. A contraction algorithm for finding small cycle cutsets. *Journal Of Algorithms*, 9:470–493, 1988.

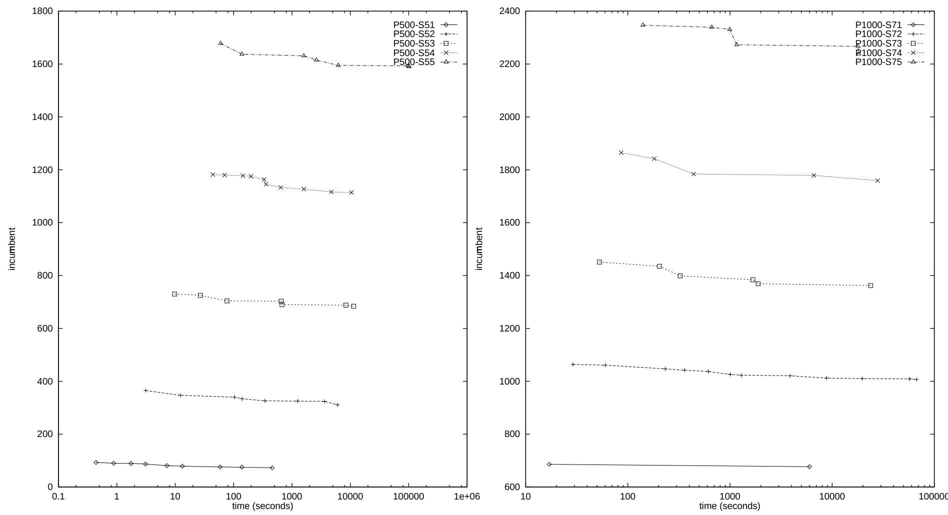

FIGURE 18. FAS: sparse 500-node and 1000-node graphs

[9] P. M. Pardalos, T. Qian, and M. G. C. Resende. A greedy randomized adaptive search procedure for feedback vertex set. *J. of Combinatorial Optimization*, 2:399–412, 1999.

[10] A. Shamir. A linear time algorithm for finding minimum cutsets in reduced graphs. *SIAM Journal on Computing*, 8:645–655, 1979.

[11] C. Wang, E. Lloyd, and M. Soffa. Feedback vertex sets and cyclically reducible graphs. *Journal of the Association for Computing Machinery*, 32:296–313, 1985.

[12] M. Yannakakis. Node and egde-deletion NP-complete problems. In *Proceedings of the 10-th Annual ACM Symposium on Theory of Computing*, pages 253–264. Association of Computing Machinery, 1978.

(P. Festa) MATHEMATICS AND COMPUTER SCIENCE DEPARTMENT, UNIVERSITY OF SALERNO, 84081 BARONISSI (SA), ITALY.

E-mail address, P. Festa: `paofes@udsab.dia.unisa.it`

(P. M. Pardalos) CENTER FOR APPLIED OPTIMIZATION, DEPARTMENT OF INDUSTRIAL AND SYSTEMS ENGINEERING, UNIVERSITY OF FLORIDA, GAINESVILLE, FL 32611 USA.

E-mail address, P. M. Pardalos: `pardalos@ufl.edu`

(M. G. C. Resende) INFORMATION SCIENCES RESEARCH, AT&T LABS RESEARCH, FLORHAM PARK, NJ 07932 USA.

E-mail address, M. G. C. Resende: `mgcr@research.att.com`