A GREEDY RANDOMIZED ADAPTIVE SEARCH PROCEDURE FOR THE FEEDBACK VERTEX SET PROBLEM*

PANOS M. PARDALOS†, TIANBING QIAN‡, AND MAURICIO G.C. RESENDE§

Abstract. A Greedy Randomized Adaptive Search Procedure (GRASP) is a randomized heuristic that has produced high quality solutions for a wide range of combinatorial optimization problems. The NP-complete Feedback Vertex Set (FVS) Problem is to find the minimum number of vertices that need to be removed from a directed graph so that the resulting graph has no directed cycle. The FVS problem has found applications in many fields, including VLSI design, program verification, and statistical inference. In this paper, we develop a GRASP for the the FVS problem. We describe GRASP construction mechanisms and local search, as well as some efficient problem reduction techniques. We report computational experience on a set of test problems using three variants of GRASP.

Key words. Combinatorial optimization, feedback vertex set, local search, GRASP, probabilistic algorithm, computer implementation, heuristics, integer programming

1. Introduction. The feedback vertex set (FVS) problem can be stated as follows: Given a directed graph $G(V,E)$, where V denotes the set of n vertices and E the set of arcs, find a minimum cardinality subset S of V such that every directed cycle in G contains at least one vertex in S. In other words, we wish to determine how to remove the minimum number of vertices from the original graph such that the resulting graph has no directed cycle. A *cutset* is a set of vertices whose removal from the graph eliminates all directed cycles of the graph. The minimum cardinality cutset is called the *feedback vertex set*. The FVS problem is known to be NP-complete [8, 19] and has found applications in many diverse areas, including program verification [17], deadlock prevention [18], and Bayesian inference [20].

The FVS problem can be formulated as a set covering problem. Let C_G denote the set of all cycles in the graph G and define

$$x_j = \begin{cases} 1 & \text{if vertex } j \text{ is in the feedback vertex set} \\ 0 & \text{otherwise.} \end{cases}$$

The minimum number of vertices that need to be removed so that the resulting graph is cycle-free can be found by solving the following integer programming (set covering) problem:

$$\min \sum_{j=1}^{n} x_j$$

such that

$$\sum_{j \in C_j} x_j \geq 1, \;\; C_j \in C_G$$

$$x_j \in \{0,1\}, \;\; j = 1, \ldots, n.$$

Among the classical NP-complete problems, the FVS problem has been regarded as "one of the least understood problems" [18]. Solvable special cases are studied in [17, 18]. From

*October 7, 1997– URL = ftp://www.research.att.com/~mgcr/doc/gfvs.ps.Z

†Center for Applied Optimization, Department of ISE, 303 Weil Hall, University of Florida, Gainesville, FL 32611 USA. e-mail: pardalos@ufl.edu

‡Department of Management Sciences, The University of Iowa, Iowa City, IA 52242 USA. e-mail: tqian@dollar.biz.uiowa.edu

§Information Sciences Research Center, AT&T Labs Research, Florham Park, NJ 07932 USA. e-mail: mgcr@research.att.com

the point of view of approximation algorithms, Erdös and Pósa [3] proposed an algorithm with a ratio of approximation of $2\log n$, which was later improved by Monien and Schultz [11] to a ratio of $\sqrt{\log n}$. Recently, Bafna et al. [1] proposed an algorithm for the FVS problem on undirected graphs with approximation ratio of 2 (see also [2]). The best approximation ratio for the directed version is $O(\log\ n\ \log\log\ n)$ and is due to Seymour [16]. Other approximations can be found in Qian, Ye, and Pardalos [13] and references therein. Although these approximation algorithms guarantee a solution of a certain quality, often in practice heuristic methods can provide better quality solutions in reasonable CPU time. Since there usually is no theoretical guarantee of the quality of the solution obtained by a heuristic, often computational experimentation is used to determine its effectiveness. In this paper, we present and analyze empirically a Greedy Randomized Adaptive Search Procedure (GRASP) to solve large instances of the FVS problem.

A GRASP is a randomized heuristic that has found successful applications for a variety of combinatorial optimization problems [4, 10, 14]. GRASP is a multistart method having two phases, a construction phase and a local search phase. During the construction phase, a feasible solution is constructed iteratively, one element at a time. Each element of the solution is randomly selected from a restricted candidate list (RCL) that contains elements that are well-ranked according to some greedy function. Once the solution is constructed, there is no guarantee that it is a locally optimal solution with respect to the adopted neighborhood definition. The local search phase tries to improve the constructed solution and produces a solution that is locally optimal with respect to the specified neighborhood structure. This two-stage process is applied repeatedly, and the best solution found is kept as an approximation of the optimal. We call this solution the GRASP solution.

In this paper, we present a new algorithm, using the GRASP metaheuristic, for finding approximate solutions of large instances of the FVS problem. Several degree-related greedy functions are used in the construction of the RCL. To improve the efficiency of the algorithm, we incorporate problem reduction techniques [9] into the construction phase of GRASP. The iterative construction of the solution alternates between the choice of a vertex from the RCL and the optimal solution preserving reduction. The motivation is that when we add a new vertex to the partial solution, the problem size is reduced. If there is a solution preserving reduction that can further reduce the problem size, then the construction process may be sped up. More importantly, we have observed empirically that the risk of the constructed solution being trapped in a local minimum is considerably reduced. For the local search component of GRASP, checking acyclicity of a graph is nontrivial. Thus, it is relatively expensive to conduct local search for the FVS problem. By utilizing the graph reduction techniques, we design some simple procedures to produce a minimal vertex set.

In addition to the new algorithm described, the large FVS instances presented in this paper, along with the approximate solutions found by our algorithm, can serve as a suite of benchmark problems for testing future implementations.

The remainder of the paper is organized as follows. Section 2 discusses various phases of the proposed GRASP. Section 3 reports computational experience. Concluding remarks are given in Section 4.

Throughout this paper, we use the following notation. For a digraph $G(V,E)$ we define $\mathtt{in}(i) = \{j \mid (j,i) \in E\}$, $\mathtt{out}(i) = \{j \mid (i,j) \in E\}$, and $\mathtt{adj}(i) = \mathtt{in}(i) \cup \mathtt{out}(i)$. $G(i)$ is the greedy function value of vertex i and S denotes a feedback vertex set.

2. GRASP. As sketched in Section 1, a GRASP possesses two phases, a construction and a local improvement phase. To describe the construction phase, we need to provide an adaptive greedy function, a construction mechanism for the RCL, and a probabilistic selection procedure. These components are interlinked, forming an iterative procedure that constructs

```
procedure GRASP(RCLSize, MaxIter, RandomSeed)
1     BestSolutionFound = ∅;
2     do k = 1,...,MaxIter →
3           x =ConstructGreedyRandomizedSolution();
4           x =LocalSearch(x);
5           UpdateSolution(BestSolutionFound,x);
6     od;
7     return(BestSolutionFound)
end GRASP;
```

FIG. 2.1. *A generic GRASP pseudo-code*

```
procedure ConstructGreedyRandomizedSolution()
1     S = ∅;
2     G' =ReduceInstanceSize(G);
3     do k =1,...,n →
4           MakeRCL();
5           s = SelectIndex(RandomSeed);
6           S = S ∪ {s};
7           G' = UpdateGraph(G');
8           G' =ReduceInstanceSize(G');
9     od;
end ConstructGreedyRandomizedSolution;
```

FIG. 2.2. *GRASP construction phase pseudo-code*

a solution, one vertex at a time, biased by the adaptive greedy function. A local search procedure is then applied to possibly improve the constructed solution. Figure 2.1 shows a GRASP in pseudo code. The main loop of the GRASP consists of lines 2–6. A solution is randomly generated by the adaptive greedy function in line 3 with local search taking place in line 4. The best solution found is updated in line 5.

We next discuss the different components of the proposed GRASP. A new feature of this GRASP is the introduction of reduction techniques in the construction and local search phases. The modified generic scheme of our construction phase is outlined in Figure 2.2. Initially, the feedback vertex set S is emptry and several of solution-preserving reductions (described in Subsection 2.2) are applied to the input graph G. The main loop in the construction phase is in lines 3–9. At most $n = |V|$ vertices can be selected to be in the feedback vertex set S, so the loop is repeated at most n times. A restricted candidate list (RCL) is set up in line 4 containing any yet unselected vertex having a large greedy function value. The greedy functions are defined in Subsection 2.1. A vertex is seclected, at random, from the RCL in line 5 and the feedback vertex set is updated to include vertex s in line 6. In line 7, `UpdateGraph` removes vertex s and all edges incident to s from G'. The solution preserving reductions are applied on G' in line 8.

In the remainder of this section, we discuss the greedy adaptive functions (Subsection 2.1), the reduction heuristics (Subsection 2.2), the construction mechanism of the restricted candidate list (Subsection 2.3), and the local search phase (Subsection 2.4).

2.1. Greedy functions. Several greedy functions are tested in this study. Usually, the greedy function of a vertex is linked to its contribution to the objective function achieved by selecting that vertex. However, in the case of the FVS problem, it seems hard to directly link

```
procedure ReduceInstanceSize(G,S)
1     Apply reduction 1;
2     Apply reduction 2;
3     Apply reduction 3;
4     for (at least one reduction succeeds) →
5           Apply reduction 1;
6           Apply reduction 2;
7           Apply reduction 3;
8     rof;
9     return(reduced graph G');
end ReduceInstanceSize;
```

FIG. 2.3. *Solution preserving reduction pseudo-code*

the effect of cycle reduction to any vertex. Intuitively speaking, the larger the degree of a vertex, the more likely that the deletion of this vertex will cut off more cycles. Therefore, it is natural to link the greedy function to the degree of a vertex in one way or another. The three greedy functions employed in this study are as follows:

1. $G_A(i) = |\mathtt{in}(i)| + |\mathtt{out}(i)|$.
2. $G_B(i) = |\mathtt{in}(i)| * |\mathtt{out}(i)|$.
3. $G_C(i) = \max(|\mathtt{in}(i)|, |\mathtt{out}(i)|)$.

Roughly speaking, G_A puts equal weight on both in-degree and out-degree, G_B favors the balance between in- and out-degree, and G_C only considers the largest value of the two degrees. As will be demonstrated later, there is considerable difference among these three greedy functions in terms of solution quality.

2.2. Problem reduction techniques. A solution preserving reduction of a graph G is a contraction of G (i.e. removal of a subset of vertices and edges from G) such that the original graph and the reduced graph have the same FVS. Three solution preserving reductions are used in the GRASP implementation. They are:

1. If $\mathtt{out}(i) = 0$ or $\mathtt{in}(i) = 0$, then $i \in S$. The reduction is
   (a) $V = V \setminus \{i\}$
   (b) $E = E \setminus \{(x,y) \mid x = i \text{ or } y = i\}$.
2. If $(i,i) \in E$, then $i \in S$. The reduction is
   (a) $V = V \setminus \{i\}$
   (b) $E = E \setminus \{(i,j) \text{ or } (j,i), \text{ for } \forall j \in V\}$.
3. If $\mathtt{in}(i) = 1$ and $(j,i) \in E$, then the reduction is
   (a) $V = V \setminus \{i\}$
   (b) $\mathtt{out}(j) = \mathtt{out}(j) \cup \mathtt{out}(i)$
   (c) $E = E \cup \{(j,k) \mid k \in \mathtt{out}(i)\} \setminus \{(i,k) \mid k \in \mathtt{out}(i)\}$

   If $\mathtt{out}(i) = 1$ and $(i,j) \in E$, then the reduction is
   (a) $V = V \setminus \{i\}$
   (b) $\mathtt{in}(i) = \mathtt{in}(i) \cup \mathtt{in}(j)$
   (c) $E = E \cup \{(k,j) \mid k \in \mathtt{in}(i)\} \setminus \{(k,i) \mid k \in \mathtt{in}(i)\}$.

The above reductions are implemented in the procedure `ReduceInstanceSize`, outlined in Figure 2.3. When a graph is acyclic, it can be shown that `ReduceInstanceSize` will return an empty cutset and an empty reduced graph. Also, note that these reductions are recursive in nature, i.e. they can be applied to a graph repeatedly. A useful application of these reduction procedures is to determine whether a directed graph is acyclic or not. If $\mathtt{ReduceInstanceSize}(G) = \emptyset$, then G is acyclic. This result is used in the local search phase

```
procedure LocalSearch(G'(V',E'),S)
1     for (cutset S contains redundant vertices) →
2         for i = 1,...,|S|  →
3             V'' = V' ∪ {S \ {s_i}};
4             E'' = E' ∪ {(v,w) ∈ E | v or w ∈ {S \ {s_i}};
5             G'''(V''',E''') =ReduceInstanceSize(G''(V'',E''));
6             if (V''' = ∅) →
7                 S = S \ {s_i};
8                 break;
9             fi;
10        rof;
11    rof;
end LocalSearch;
```

FIG. 2.4. *GRASP local search phase pseudo-code*

of GRASP as will be described later. For a detailed analysis of these reduction procedures, see also Levy and Lowe [9].

2.3. Construction of the RCL. There are several ways to build an RCL. In one approach, the RCL is made up of the k candidates with the largest value of the adaptive greedy function [4]. In another, it is made up of a variable number of well ranked elements. In this study, since the greedy function is directly linked to the degree of the vertex, there is usually a large number of vertices having the same greedy function value. Therefore, we choose all vertices that have greedy function values greater than α times the largest greedy function value, for α such that $0 \leq \alpha \leq 1$.

2.4. Local search phase. In contrast to the simplicity of the greedy functions, the local search phase for the FVS problem is more complicated and expensive. The main difficulty associated with the local search is verifying whether a graph is acyclic or not. Therefore, we limit our local search to attempt to eliminate redundant elements of the cutset, thus resulting in a minimal cutset. The pseudo-code of the local search heuristic is presented in Figure 2.4, where S is the FVS returned by the algorithm and s_i is the i-th element in the current cutset S.

The local search heuristic works as follows. For any given cutset, it checks whether each vertex of the cutset is redundant. This is done by excluding each vertex from the cutset once, then applying the reduction heuristics to each reduced graph. If in one iteration, no reduction heuristic can be successfully applied, then the reduced graph is cyclic and therefore this vertex is not redundant. Otherwise, the heuristic will return an empty reduced graph, indicating that the vertex examined is redundant and can be dropped from the current cutset.

We observed empirically that the above procedure usually improves the construction phase solution. The typical improvement is 1 – 3 vertices, with a 1-vertex improvement occurring most often. Based on this observation and the cost of the local search, it may be recommended that the above procedure be applied only to cutsets of high quality. One approach is to apply this local search only to constructed solutions that are better than the average constructed solution.

3. Experimental results. In this section, we report on computational experience with three variants of GRASP (A, B, and C). GRASP variants A, B, and C corresponds to greedy functions G_A, G_B, and G_C, respectively, as defined in Subsection 2.1.

```
20 196 MHZ IP25 Processors
CPU: MIPS R10000 Processor Chip Revision:  2.4
FPU: MIPS R10010 Floating Point Chip Revision:  0.0
Data cache size:  32 Kbytes
Instruction cache size:  32 Kbytes
Secondary unified instruction/data cache size:  1 Mbyte
Main memory size:  1024 Mbytes, 2-way interleaved
```

FIG. 3.1. *Hardware configuration (partial output of system command* `hinv`*)*

TABLE 3.1
Funke & Reinelt test problems

			GRASP A			GRASP B			GRASP C		
name	$\|V\|$	$\|E\|$	sol'n	itr	time	sol'n	itr	time	sol'n	itr	time
r_25_10	25	57	3	1	0.001	3	1	0.001	3	1	0.001
r_25_20	25	127	11	1	0.003	11	2	0.004	11	4	0.010
r_25_30	25	172	12	1	0.002	12	5	0.012	12	1	0.003
r_30_10	30	84	5	1	0.001	5	1	0.001	5	1	0.001
r_30_20	30	154	11	7	0.020	11	1	0.003	11	1	0.003
r_30_30	30	238	16	2	0.011	16	1	0.007	16	12	0.060
r_35_10	35	111	7	2	0.003	7	1	0.002	7	1	0.002
r_35_20	35	246	17	1	0.009	17	2	0.010	17	1	0.008
r_35_30	35	356	21	9	0.070	21	1	0.009	21	1	0.008
		sum	103	25	0.120	103	15	0.049	103	23	0.096

The experiments were carried out using two sets of test problems. [1] The first data set was used in Funke and Reinelt [7]. Those small instances are randomly generated with varying number of vertices and varying densities and have known optimal solutions. The second set of test problems are randomly generated, having from 50 to 1000 vertices and varying densities.

The experiment was conducted on a Silicon Graphics Power Challenge computer (196 MHz MIPS R10000 processor), whose hardware configuration is summarized in Figure 3.1. The code was compiled on the SGI Fortran compiler `f77` using compiler flags `-O3 -n32 -static -mips4`. Processes were limited to a single processor. CPU times in seconds were computed by calling the system routine `etime()`.

Reported CPU times exclude problem input time, which is negligible for these test problems, and give the total time taken to find the GRASP solution. The number of iterations reported corresponds to the reported times, i.e. we report the iteration where the GRASP solution was found. Note that the time actually taken by the GRASP is usually longer than what is reported in the tables. This is so because each GRASP run is repeated $\texttt{MaxIter} = 100$ iterations. Even though the CPU time for each local search may vary, a good approximation of the total running time is to take the average time per iteration and multiply it by `MaxIter`.

The performance of most heuristics depends on parameter settings. GRASP requires few parameters to be set. To facilitate reproducibility, as well as to investigate the robustness of the approach, we limit our runs in this experiment to a single set of parameter settings. We use $\texttt{MaxIter} = 100$ and $\alpha = 0.8$. Even though, CPU time could be reduced by selectively applying the local search only to good constructed solutions, in our computational experiments, we apply the local search at every GRASP iteration.

The runs on the Funke-Reinelt data set are summarized in Table 3.1. On this class of

[1] FVS datasets: `ftp://www.research.att.com/~mgcr/data/gfvs-data.tar.gz`

TABLE 3.2
Randomly generated dense test problems with 50 nodes

problem		GRASP A			GRASP B			GRASP C		
$\|E\|$	seed	sol'n	itr	time	sol'n	itr	time	sol'n	itr	time
500	1	29	11	0.189	29	2	0.027	29	39	0.738
600	2	32	15	0.343	32	63	1.394	32	65	1.451
700	3	33	4	0.099	33	1	0.020	34	28	0.814
800	4	36	16	0.487	36	5	0.149	36	7	0.201
900	5	36	40	1.432	36	100	3.429	37	80	2.938
1000	6	38	3	0.098	38	5	0.192	38	7	0.276
1100	7	39	8	0.356	39	21	0.918	39	65	3.255
1200	8	41	17	0.931	40	41	2.226	41	10	0.540
1300	9	41	10	0.618	41	3	0.219	41	45	2.769
1400	10	42	9	0.639	42	17	0.974	42	40	2.748
	sum	367	133	5.192	366	258	9.548	369	386	15.730

TABLE 3.3
Randomly generated sparse test problems with 50 nodes

problem		GRASP A			GRASP B			GRASP C		
$\|E\|$	seed	sol'n	itr	time	sol'n	itr	time	sol'n	itr	time
100	11	3	1	0.001	3	1	0.001	3	1	0.001
150	12	9	1	0.003	9	1	0.003	9	2	0.008
200	13	13	1	0.005	13	2	0.010	13	3	0.020
250	14	17	1	0.009	17	4	0.026	17	13	0.111
300	15	19	20	0.185	19	2	0.014	20	4	0.043
350	16	21	26	0.273	21	26	0.273	22	42	0.520
400	17	24	54	0.745	25	5	0.062	24	59	0.825
500	18	25	43	0.513	25	1	0.011	25	60	0.797
550	19	12	35	0.153	12	13	0.056	12	16	0.068
600	20	19	2	0.014	19	1	0.012	19	11	0.113
	sum	162	184	1.901	163	56	0.468	164	211	2.506

test problems, all three GRASP variants found optimal solutions for all instances [6] in very few GRASP iterations. [2]. The last row of table gives the sum of all solution values, number of iterations, and CPU times. Observe that GRASP B required the fewest overall number of iterations as well as the least CPU time.

To test the effectiveness of GRASP on larger instances, we generated random digraphs with up to 1000 vertices and 50,000 edges using the FORTRAN random graph generator `mkdigraph.f`. [3] This generator takes as input the number of nodes, arcs, and the random number seed. The generator should produce the same digraphs on different computers, since it uses the portable FORTRAN random number generator of Schrage [15]. The problem characteristics as well as a summary of the GRASP runs are listed in Tables 3.2– 3.9.

Several observations can be made on the properties of FVS as well as the behavior of the proposed GRASP.

- It is observed that, for a fixed number of vertices, the size of the cutset increases with the problem density, which is expected. Also note that for relatively dense graphs (with average more than 10 edges per vertex), more than half of the vertices need to be removed for the reduced graph to become acyclic. Although these two observations are based on heuristic solutions, it is reasonable to conjecture the behavior of the optimal cutset is similar.
- With regard to performance of the three heuristics, when the cardinality of the graph

[2] GRASP solution: `ftp://www.research.att.com/~mgcr/sol/gfvs-soln.tar.gz`
[3] `mkdigraph` source code: `ftp://www.research.att.com/~mgcr/src/mkdigraph.f.gz`

TABLE 3.4
Randomly generated dense test problems with 100 nodes

problem		GRASP A			GRASP B			GRASP C		
$\|E\|$	seed	sol'n	itr	time	sol'n	itr	time	sol'n	itr	time
1000	21	56	30	2.069	56	32	2.314	58	57	4.033
1100	22	58	84	6.922	58	34	2.548	61	8	0.610
1200	23	60	17	1.493	61	6	0.492	63	65	5.819
1300	24	63	60	5.845	63	47	4.667	66	79	7.704
1400	25	66	1	0.101	64	67	7.413	67	1	0.131
1500	26	64	8	1.002	64	40	4.680	67	45	5.018
1600	27	70	9	1.157	68	19	2.534	71	18	2.311
1700	28	68	97	14.352	68	28	3.514	71	80	10.959
1800	29	71	12	1.840	68	82	13.049	73	13	1.980
1900	30	71	79	13.514	71	8	1.280	74	85	14.099
	sum	647	397	48.295	641	363	42.491	671	451	52.664

TABLE 3.5
Randomly generated sparse test problems with 100 nodes

problem		GRASP A			GRASP B			GRASP C		
$\|E\|$	seed	sol'n	itr	time	sol'n	itr	time	sol'n	itr	time
200	31	9	1	0.004	9	2	0.011	9	6	0.027
300	32	17	5	0.063	17	1	0.018	17	51	0.656
400	33	23	18	0.292	23	12	0.189	23	38	0.678
500	34	34	40	1.201	33	17	0.500	35	87	2.571
600	35	39	73	2.863	39	23	0.877	40	37	1.406
700	36	44	48	2.188	45	43	1.923	47	7	0.393
650	37	41	26	1.040	40	9	0.295	42	26	1.134
450	38	26	36	0.804	25	94	1.912	27	19	0.437
350	39	19	3	0.040	19	2	0.025	19	78	1.165
250	40	12	16	0.098	12	1	0.006	12	21	0.136
	sum	264	266	8.593	262	204	5.756	271	370	8.603

is small, there is not much difference among them. As the number of vertices increases, GRASP B dominates the other two variants and almost always produces the best solutions in the least number of GRASP iterations. This suggests that the product of in- and out-degrees (i.e. a good balance between in- and out-degrees) is a better indicator as to whether a vertex is in the cutset or not than are the sum or the maximum values. For a fixed number of nodes, as the number of arcs increases, the relative performance of GRASP B with respect to both GRASP A and GRASP C improves.

- Since the GRASP iterations are independent of each other, GRASP can be easily parallelized [5, 12]. On the largest instance solved (1000 nodes and 50,000 arcs), 100 GRASP iterations required over 6 hours of CPU time. Since a single GRASP iteration for that instance took little over 200 CPU seconds, if the GRASP iterations are distributed over many processors, the total running time can be considerably reduced.

4. Concluding remarks. In this paper, we describe three GRASP variants for finding approximate solutions of the feedback vertex set problem on general directed graphs. The heuristics make use of solution preserving reductions in both the construction phase and the local search. Experimental results on a large set of test problems suggest that the procedures produce good quality solutions in little CPU time. A parallel version of the GRASP can be used to solve larger instances of the FVS problem.

TABLE 3.6
Randomly generated dense test problems with 500 nodes

problem		GRASP A			GRASP B			GRASP C		
$\|E\|$	seed	sol'n	itr	time	sol'n	itr	time	sol'n	itr	time
5000	41	267	90	175.682	262	65	133.340	285	41	86.652
5500	42	286	61	138.679	281	7	16.330	300	98	225.992
6000	43	298	32	84.170	292	62	167.989	316	2	5.508
6500	44	308	23	68.559	304	71	219.365	322	18	52.944
7000	45	324	38	127.295	317	1	3.452	339	38	134.813
7500	46	330	8	31.699	324	46	194.024	340	49	201.431
8000	47	337	66	302.234	326	31	147.918	348	88	410.672
8500	48	345	93	472.520	336	96	498.786	356	40	206.155
9000	49	351	94	537.407	344	70	413.597	365	71	404.133
10000	50	357	49	325.288	352	86	580.664	370	18	115.891
	sum	3203	554	2263.530	3138	535	2375.470	3341	463	1844.190

TABLE 3.7
Randomly generated sparse test problems with 500 nodes

problem		GRASP A			GRASP B			GRASP C		
$\|E\|$	seed	sol'n	itr	time	sol'n	itr	time	sol'n	itr	time
1000	51	33	33	3.603	34	4	0.403	34	7	0.713
1500	52	74	1	0.269	71	25	7.480	78	95	27.862
2000	53	115	31	15.626	111	4	2.585	121	49	23.390
2500	54	158	19	13.795	152	70	60.324	166	30	21.994
3000	55	189	64	59.587	183	43	41.998	198	74	68.457
3500	56	213	3	3.693	208	15	20.203	222	50	58.037
4000	57	236	46	66.146	230	23	34.679	250	75	111.120
4500	58	255	97	162.414	250	17	28.882	272	86	147.536
2000	59	123	28	14.040	117	16	8.254	129	83	41.536
2500	60	188	9	8.245	183	80	73.458	201	68	64.290
	sum	1584	331	347.418	1539	297	278.266	1671	617	564.935

Acknowledgments. The authors would like to thank Professor Meinrad Funke for providing one of the data sets used in our experiments and an anonymous referee for comments that improved the presentation of this paper.

REFERENCES

[1] V. BAFNA, P. BERMAN, AND T. FUJITO, *Approximating feedback vertex set for undirected graphs within ratio 2*, 1994. Manuscript.

[2] A. BECKER AND G. GEIGER, *Approximation algorithms for the loop cutset problem*, in Proc. of the 10th Conference on Uncertainty in Artificial Intelligence, 1979, pp. 60–68.

[3] P. ERDÖS AND L. PÓSA, *On the maximal number of disjoint circuits of a graph*, Publ. Math. Debrechen, 9 (1962), pp. 3–12.

[4] T. A. FEO AND M. G. C. RESENDE, *Greedy randomized apaptive search procedures*, Journal of Global Optimization, 6 (1995), pp. 109–133.

[5] T. A. FEO, M. G. C. RESENDE, AND S. H. SMITH, *A greedy randomized adaptive search procedure for maximum independent set*, Operations Research, 42 (1994), pp. 860–878.

[6] M. FUNKE, 1996. Personal communication.

[7] M. FUNKE AND G. REINELT, *A polyhedral approach to the feedback vertex set problem*, 1996. Manuscript.

[8] M. R. GAREY AND D. S. JOHNSON, *Computers And Reducibility – A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.

[9] H. LEVY AND L. LOWE, *A contraction algorithm for finding small cycle cutsets*, Journal of Algorithms, 9 (1988), pp. 470–493.

[10] Y. LI, P. M. PARDALOS, AND M. G. C. RESENDE, *A greedy randomized adaptive search procedure for the quadratic assignment problem*, in Quadratic Assignment and Related Problems, P. Pardalos and H. Wolkowicz, eds., vol. 16 of DIMACS Series on Discrete Mathematics and Theoretical Computer

TABLE 3.8
Randomly generated dense test problems with 1000 nodes

problem		GRASP A			GRASP B			GRASP C		
$\|E\|$	seed	sol'n	itr	time	sol'n	itr	time	sol'n	itr	time
10000	61	545	16	166.831	531	6	61.532	568	46	509.558
15000	62	650	27	571.371	638	56	1216.152	684	91	2011.868
20000	63	719	88	3209.624	708	21	766.150	744	2	73.515
25000	64	764	95	5060.780	755	5	255.889	790	25	1351.301
30000	65	801	17	1213.047	792	30	2161.844	821	42	3052.569
20000	66	720	55	2063.221	713	95	3772.719	745	71	2717.697
30000	67	802	17	1142.092	791	4	304.570	819	33	2369.979
40000	68	844	96	11944.590	836	38	4816.231	857	7	856.952
50000	69	867	6	1277.027	862	36	8080.269	881	5	1057.569
45000	70	859	43	7366.830	852	64	10727.460	872	54	9186.821
	sum	7571	460	34015.413	7478	355	32162.816	7781	376	23187.829

TABLE 3.9
Randomly generated sparse test problems with 1000 nodes

problem		GRASP A			GRASP B			GRASP C		
$\|E\|$	seed	sol'n	itr	time	sol'n	itr	time	sol'n	itr	time
3000	71	151	6	7.334	148	11	13.966	159	76	93.645
3500	72	188	43	69.292	185	72	127.191	199	94	153.218
4000	73	225	2	4.112	217	1	2.198	238	93	197.254
4500	74	274	100	283.969	261	40	105.853	283	76	190.505
5000	75	303	77	239.386	299	79	245.280	327	40	131.053
5500	76	339	78	284.014	330	30	118.884	362	66	244.803
6000	77	380	6	24.561	368	27	118.739	404	11	49.170
6500	78	407	16	77.992	394	25	128.487	430	53	264.913
7000	79	431	75	432.295	420	45	259.267	465	41	242.707
8000	80	471	15	122.550	458	7	57.529	494	43	367.425
	sum	3169	418	1545.505	3080	337	1177.394	3361	593	1934.693

Science, American Mathematical Society, Providence, R.I., 1994, pp. 237–261.

[11] B. MONIEN AND R. SCHULTZ, *Four approximation algorithms for the feedback vertex set problem*, Proc. of the 7th Conference on Graph Theoretic Concepts of Computer Science, (1981), pp. 315–390.

[12] P. M. PARDALOS, L. S. PITSOULIS, AND M. G. C. RESENDE, *A parallel GRASP implementation for the quadratic assignment problem*, in Parallel Algorithms for Irregularly Structured Problems – Irregular'94, A. Ferreira and J. Rolim, eds., Kluwer Academic Publishers, 1995, pp. 111–130.

[13] T. QIAN, Y. YE, AND P. M. PARDALOS, *A pseudo ε-approximation algorithm for FVS*, in State of the Art in Global Optimization, C. Floudas and P. Pardalos, eds., Kluwer Academic Publishers, Dordrecht, Boston, London, 1996, pp. 341–351.

[14] M. G. C. RESENDE AND T. A. FEO, *A GRASP for satisfiability*, in The Second DIMACS Implementation Challenge, M. Trick and D. Johnson, eds., vol. 26 of DIMACS Series on Discrete Mathematics and Theoretical Computer Science, American Mathematical Society, 1996, pp. 499–520.

[15] L. SCHRAGE, *A more portable Fortran random number generator*, ACM Transactions on Mathematical Software, 5 (1979), pp. 132–138.

[16] P. D. SEYMOUR, *Packing directed circuits fractionally*, Combinatorica, 15 (1995), pp. 182–188.

[17] A. SHAMIR, *A linear time algorithm for finding minimum cutsets in reduced graphs*, SIAM Journal On Computing, 8 (1979), pp. 654–655.

[18] C. WANG, E. LLOYD, AND M. SOFFA, *Feedback vertex sets and cyclically reducible graphs*, Journal of the ACM, 32 (1985), pp. 296–313.

[19] M. YANNAKAKIS, *Node and edge-deletion NP-complete problems*, in Proc. of the 10th Annual ACM Symp. on Theory of Computing, 1978, pp. 253–264.

[20] B. 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*, in Proc. of the 5th Annual ACM-SIAM Symp. on Discrete Algorithms, 1994, pp. 344–354.