Greedy Randomized Adaptive Search Procedures

Article *in* Journal of Global Optimization · March 1995

DOI: 10.1007/BF01096763 · Source: CiteSeer

Journal of Global Optimization, 6, 109–134 (1995)

Greedy Randomized Adaptive Search Procedures

THOMAS A. FEO feo@emx.utexas.edu
Operations Research Group, Department of Mechanical Engineering, The University of Texas, Austin, TX 78712 USA

MAURICIO G.C. RESENDE mgcr@research.att.com
Mathematical Sciences Research Center, AT&T Bell Laboratories, Murray Hill, NJ 07974 USA

Received June 30, 1994

Editor: Panos M. Pardalos

Abstract. Today, a variety of heuristic approaches are available to the operations research practitioner. One methodology that has a strong intuitive appeal, a prominent empirical track record, and is trivial to efficiently implement on parallel processors is GRASP (Greedy Randomized Adaptive Search Procedures). GRASP is an iterative randomized sampling technique in which each iteration provides a solution to the problem at hand. The incumbent solution over all GRASP iterations is kept as the final result. There are two phases within each GRASP iteration: the first intelligently constructs an initial solution via an adaptive randomized greedy function; the second applies a local search procedure to the constructed solution in hope of finding an improvement. In this paper, we define the various components comprising a GRASP and demonstrate, step by step, how to develop such heuristics for combinatorial optimization problems. Intuitive justifications for the observed empirical behavior of the methodology are discussed. The paper concludes with a brief literature review of GRASP implementations and mentions two industrial applications.

Keywords: Combinatorial optimization, search heuristic, GRASP, computer implementation

1. Introduction

Optimization problems that involve a large but finite number of alternatives often arise in industry, government and science. Common examples include designing efficient telecommunication networks, scheduling operations in a semiconductor manufacturing plant, designing effective school zoning, locating strategic energy reserves, routing delivery vehicles, troop deployment, airline crew scheduling, and designing a large experiment. In all of these examples, it is theoretically possible to enumerate all combinations of solutions and evaluate each with respect to the stated objective. The ones that provide the most favorable outcome are deemed *optimal*. However, from a practical perspective, it is infeasible to follow such a strategy of complete enumeration because the number of combinations often grows exponentially with the size of problem.

Much work has been done over the last 40 years to develop optimal seeking methods that do not explicitly require an examination of each alternative. This research has given rise to the field of *combinatorial optimization* (see Papadimitriou and Steiglitz [32]), and an increasing capability to solve ever larger real-world problems. Notable successes have been reported for linear programming [23], specialized versions of the traveling salesman problem [30] and bus driver scheduling [16], to name a few.

```
procedure grasp()
1    InputInstance();
2    for GRASP stopping criterion not satisfied →
3        ConstructGreedyRandomizedSolution(Solution);
4        LocalSearch(Solution);
5        UpdateSolution(Solution,BestSolutionFound);
6    rof;
7    return(BestSolutionFound)
end grasp;
```

Figure 1. A generic GRASP pseudo-code

```
procedure ConstructGreedyRandomizedSolution(Solution)
1    Solution = {};
2    for Solution construction not done →
3        MakeRCL(RCL);
4        s = SelectElementAtRandom(RCL);
5        Solution = Solution ∪ {s};
6        AdaptGreedyFunction(s);
7    rof;
end ConstructGreedyRandomizedSolution;
```

Figure 2. GRASP construction phase pseudo-code

Nevertheless, most problems found in industry and government are either computationally intractable by their nature, or sufficiently large so as to preclude the use of exact algorithms. In such cases, heuristic methods are usually employed to find good, but not necessarily optimal solutions. The effectiveness of these methods depends upon their ability to adapt to a particular realization, avoid entrapment at local optima, and exploit the basic structure of the problem, such as a network or a natural ordering among its components. Furthermore, restart procedures, controlled randomization, efficient data structures, and preprocessing are also beneficial. Building on these notions, various heuristic search techniques have been developed that have demonstrably improved our ability to obtain good solutions to difficult combinatorial optimization problems. The most promising of such techniques include simulated annealing [25], tabu search [19], [20], genetic algorithms [21] and GRASP (Greedy Randomized Adaptive Search Procedures).

In this paper, we define the various components comprising a GRASP and demonstrate, step by step, how to develop such heuristics for combinatorial optimization problems. Intuitive justifications for the observed empirical behavior of the methodology will be discussed. The paper concludes with a brief literature review of GRASP and mentions two industrial applications.

A GRASP is an iterative process, with each GRASP iteration consisting of two phases, a construction phase and a local search phase. The best overall solution is kept as the

```
procedure local(P,N(P),s)
1    for s not locally optimal →
2            Find a better solution t ∈ N(s);
3            Let s = t;
4    rof;
5    return(s as local optimal for P)
end local;
```

Figure 3. GRASP local search phase

result. A generic GRASP pseudo-code is given in Figure 1. Line 1 of the pseudo-code corresponds to problem input. The GRASP iterations take place in lines 2–6, and terminate when some termination criterion, such as maximum number of iterations have occured or solution sought has been found, is satisfied. Line 3 is the GRASP construction phase, while line 4 is the local search phase. If an improved solution is found, the incumbent is updated in line 5. We next present a high-level description of these two phases. In the following section we delve into more detail.

In the construction phase, a feasible solution is iteratively constructed, one element at a time. At each construction iteration, the choice of the next element to be added is determined by ordering all elements in a candidate list with respect to a greedy function. This function measures the (myopic) benefit of selecting each element. The heuristic is adaptive because the benefits associated with every element are updated at each iteration of the construction phase to reflect the changes brought on by the selection of the previous element. The probabilistic component of a GRASP is characterized by randomly choosing one of the best candidates in the list, but not necessarily the top candidate. The list of best candidates is called the *restricted* candidate list (RCL). This choice technique allows for different solutions to be obtained at each GRASP iteration, but does not necessarily compromise the power of the adaptive greedy component of the method. Figure 2 displays pseudo-code for the construction phase of GRASP. The solution to be contructed is initialized in line 1 of the pseudo-code. The loop from line 2 to 7 is repeated until the solution is constructed. In line 3, the restricted candidate list is built. A candidate from the list is selected, at random, in line 4 and is added to the solution in line 5. The effect of the selected solution element s on the benefits associated with every element is taken into consideration in line 6, where the greedy function is adapted.

As is the case for many deterministic methods, the solutions generated by a GRASP construction are not guaranteed to be locally optimal with respect to simple neighborhood definitions. Hence, it is almost always beneficial to apply a local search to attempt to improve each constructed solution. A local search algorithm works in an iterative fashion by successively replacing the current solution by a better solution in the neighborhood of the current solution. It terminates when no better solution is found in the neighborhood. The *neighborhood structure* N for a problem P relates a solution s of the problem to a subset of solutions $N(s)$. A solution s is said to be *locally optimal* if there is no better solution in $N(s)$. Given a neighborhood structure N, a local search algorithm has the general form

size	solution values										
RCL	3016	3017	3018	3019	3020	3021	3022	3023	3024	3025	3026
1									100000		
2							151	6053	93796		
4						75	1676	17744	80503	2	
8				1	50	750	6566	31257	61336	35	5
16				16	282	2485	13274	38329	45547	42	25
32		1	3	72	635	4196	16455	37937	40479	164	58
64		4	18	177	1213	5933	19553	37666	34832	441	163
128		4	36	269	1716	7324	21140	37186	34832	679	281
256	1	5	35	304	1980	7867	21792	36725	29027	1575	689

Figure 4. Sample distributions of GRASP iteration solutions

size RCL	1	2	4	8	16	32	64	128	256
mean (3120 +)	4.00	3.94	3.79	3.53	3.27	3.14	3.00	2.91	2.89

Figure 5. Means of sample distributions of GRASP iteration solutions

as stated in Figure 3. The key to success for a local search algorithm consists of the suitable choice of a neighborhood structure, efficient neighborhood search techniques, and the starting solution.

While such local optimization procedures can require exponential time from an arbitrary starting point, empirically their efficiency significantly improves as the initial solution improves. Through the use of customized data structures and careful implementation, an efficient construction phase can be created which produces good initial solutions for efficient local search. The result is that often many GRASP solutions are generated in the same amount of time required for the local optimization procedure to converge from a single random start. Furthermore, the best of these GRASP solutions is generally significantly better than the single solution obtained from a random starting point.

It is difficult to formally analyze the quality of solution values found by using the GRASP methodology. However, there is an intuitive justification that views GRASP as a repetitive sampling technique. Each GRASP iteration produces a sample solution from an unknown distribution of all obtainable results. The mean and variance of the distribution are functions of the restrictive nature of the candidate list. For example, if the cardinality of the restricted candidate list is limited to one, then only one solution will be produced and the variance of the distribution will be zero. Given an effective greedy function, the mean solution value in this case should be good, but probably suboptimal. If a less restrictive cardinality limit is imposed, many different solutions will be produced implying a larger variance. Since the greedy function is more compromised in this case, the mean solution value should degrade. Intuitively, however, by order statistics and the fact that the samples are randomly produced, the best value found should outperform the mean value. Indeed, often the best solutions sampled are optimal. Figures 4–5 show results of a simulation experiment that illustrates this intuition. The figures show, for different cardinality restriction values (candidate list size $= 1, 2, \ldots, 256$), the distribution of observed solution values $(3017, 3018, \ldots, 3026)$ obtained at each iteration, for 100,000 replications of GRASP

iterations. The simulation uses the code *GRASP-B*, of Resende and Feo [34], to solve satisfiability instance `ssa7552-160` of the 2nd DIMACS Algorithm Implementation Challenge [22]. In the optimization problem, one wants to maximize the number of satisfied clauses. Problem instance `ssa7552-160` is satisfiable and has 1391 variables, 3126 clauses, and 7025 literals. Consequently, the optimal solution value is 3216. The simulation shows that the greedy solution ($|\text{RCL}| = 1$) has the highest mean solution value (3124.00) and the smallest variance (zero). As the restriction is increasingly relaxed, the mean values decrease and the variances increase. With the increase in variance, the number of samples drawn from the set of optimal solutions also increases. No optimal solution is drawn for $|\text{RCL}| = 1, 2$, and 4. Five are drawn for $|\text{RCL}| = 8$, and as many as 689 are drawn for the largest list size of 256.

An especially appealing characteristic of GRASP is the ease with which it can be implemented. Few parameters need to be set and tuned (candidate list size and number of GRASP iterations), and therefore, development can focus on implementing efficient data structures to assure quick GRASP iterations. Finally, GRASP can be trivially implemented on a parallel processor in an MIMD environment. Each processor can be initialized with its own copy of the procedure, the instance data, and an independent random number sequence. The GRASP iterations are then performed in parallel with only a single global variable required to store the best solution found over all processors.

The remainder of the this paper is organized as follows. In Section 2, the GRASP methodology is described in detail. Several GRASP implementations are summarized in Section 3. A summary and discussion of future work are presented in Section 4.

2. Methodology

In this section, we describe a general framework for GRASP, using two classical combinatorial optimization problems (set covering and the maximum independent set) to illustrate the various components of the methodology. We define the two problems and describe the two phases of GRASP with respect to each problem class. Examples are given for the procedures described. We conclude the section by describing computational testing of GRASP codes for set covering and maximum independent set.

2.1. Problem definitions

We begin by defining the two combinatorial optimization problems used in this section to illustrate the phases of a GRASP: the set covering problem and the maximum independent set problem.

2.1.1. Set covering problem

Given n finite sets $P_1, P_2, \ldots, P_n$, let

P_1	P_2	P_3	P_4	
•	•			1
•		•		2
	•		•	3

Figure 6. Set covering problem example

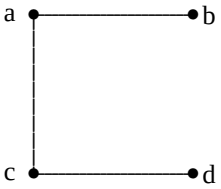

Figure 7. Independent set problem example

$$I = \bigcup_{i=1}^{n} P_i = \{1, 2, \ldots, m\}$$

and $J = \{1, 2, \ldots, n\}$. A set $J^* \subseteq J$ is a *cover* if $\cup_{i \in J^*} P_i = I$. In the *set covering problem* we want to find the minimum cardinality cover.

Consider the example in Figure 6 where four sets $P_1 = \{1,2\}$, $P_2 = \{1,3\}$, $P_3 = \{2\}$, and $P_4 = \{3\}$ are given. There are 7 valid covers for this example: $\{P_1, P_2, P_3, P_4\}$, $\{P_1, P_2, P_3\}$, $\{P_1, P_2, P_4\}$, $\{P_2, P_3, P_4\}$, $\{P_1, P_2\}$, $\{P_1, P_4\}$, $\{P_2, P_3\}$. The optimal covers, of size 2, are: $\{P_1, P_2\}$, $\{P_1, P_4\}$ and $\{P_2, P_3\}$.

2.1.2. Maximum independent set problem

Given a graph $G = (V, E)$ where V is the vertex set and E is the edge set of G, an *independent set* (or *vertex packing* or *stable set*) is a set of vertices whose elements are pairwise nonadjacent. In the *maximum independent set problem* we want an independent set of maximum cardinality.

Consider the example in Figure 7 where a graph with four vertices $\{a, b, c, d\}$ and three edges $\{(a,b), (a,c), (c,d)\}$ is given. The independent sets for this example are $\{a, d\}$, $\{b, c\}$, $\{a\}$, $\{b\}$, $\{c\}$ and $\{d\}$. There are two maximum independent sets: $\{a, d\}$ and $\{b, c\}$.

2.2. GRASP construction phase

During the construction phase of GRASP a solution is built one element at a time, with each element selected at random from a list of candidates determined by an adaptive greedy function. In this subsection, we illustrate the construction phase by defining adaptive

```
procedure ConstructCover(n, P_1, P_2, ..., P_n, α, J*)
1     for j = 1, ..., n → P_j^0 := P_j rof;
2     J* := ∅;
3     for P_j^0 ≠ ∅, ∀ j = 1, ..., n →
4           Γ̄ := max{|P_j^0| : 1 ≤ j ≤ n};
5           P := {j : |P_j^0| ≥ α · Γ̄, 1 ≤ j ≤ n};
6           Select k at random from P;
7           J* := J* ∪ {k};
8           for j = 1, ..., n → P_j^0 := P_j^0 \ P_k^0 rof;
9     rof;
end ConstructCover;
```

Figure 8. Construction phase pseudo-code: set covering

greedy functions and candidate list restriction mechanisms for the two examples described above.

2.2.1. Set covering problem

A set P_i is said to cover the set $I' \subseteq I$ if $P_i \cap I' = I'$. A greedy choice in the set covering problem is to select the set P_i that covers the largest number of yet uncovered elements of set I. Let us use this as the adaptive greedy function to construct a solution for the problem. Instead of making the greedy choice, we allow a set to be in the restricted candidate list if the number of yet uncovered elements that would be covered if that set were to be chosen is within some percentage (α) of the number covered by a greedy choice. This type of candidate list limitation is referred to as *value* restriction. Similarly, we can limit the size of the candidate list by including only the β best elements. This limitation is referred to as a *cardinality* restriction. Note that one may apply both types of restrictions simultaneously to form a candidate list.

Figure 8 illustrates, with pseudo-code, a value-restricted construction phase for the set covering problem. The procedure takes as input the dimension n, sets $P_1, \ldots, P_n$, parameter α, and returns the cover J^*. Steps 1 and 2 initialize sets $P_1^0, \ldots, P_n^0$ and J^*. Steps 4–8 are repeated until all sets $P_j^0,\ j = 1, \ldots, n$, are empty. In step 4, the largest cardinality $\bar{\Gamma}$ of sets $P_j^0,\ j = 1, \ldots, n$, is determined. This cardinality is used in step 5, where the restricted set $\mathcal{P}$ of candidates is built. An element k of set $\mathcal{P}$ is selected at random in step 6 and is added to the cover J^* in step 7. In step 8, the greedy function is adjusted, i.e. elements of set P_k^0 are removed from each set $P_j^0,\ j = 1, \ldots, n$.

Consider the example in Figure 9 and let $\alpha = 40$ percent. The numbers on the bottom row are the number of yet uncovered elements that would become covered if the corresponding set on the top row of the figure were to be selected. The greedy choices, P_4, P_5, or P_6 would therefore cover 3 elements. Since $\alpha = 40$ percent, the value restricted candidate list RCL $= \{P_1, P_4, P_5, P_6, P_7\}$. Suppose, at random, that set P_5 is selected. Then

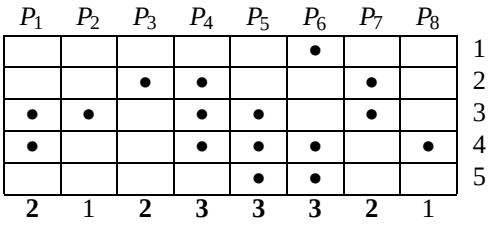

Figure 9. Set covering example: construction phase

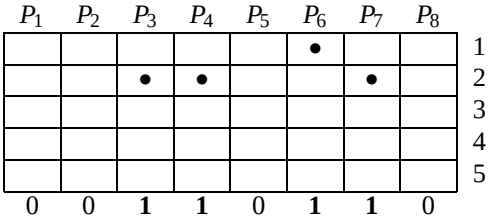

Figure 10. Set covering example: construction phase

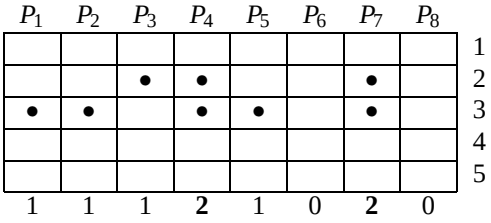

Figure 11. Set covering example: construction phase

elements 3,4 and 5 are covered and we are left with the situation depicted in Figure 10, with RCL $= \{P_3, P_4, P_6, P_7\}$. Next, choosing P_3 would leave the remaining choice as P_6, and the resulting constructed cover would be $J^* = \{P_3, P_5, P_6\}$, of size 3. On the other hand, if P_6 had initially been chosen in place of P_5, we would be in the situation depicted in Figure 11, where choosing P_4 results in a smaller (optimal) cover $J^* = \{P_4, P_6\}$ of size 2.

2.2.2. Maximum independent set problem

In the case of the maximum independent set problem, a GRASP builds an independent set, one vertex at a time, guided by an adaptive greedy function. Let S^* denote the independent set to be constructed. The GRASP begins with $S^* = \{\emptyset\}$. Let $k = 0$, $V_k = V$ and $E_k = E$. A

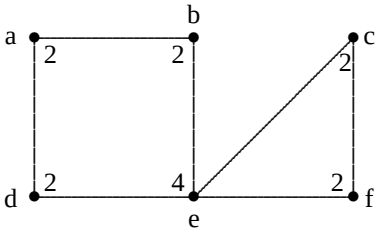

Figure 12. Maximum independent set: construction phase

plausible greedy choice for the maximum independent set is to select the vertex with the smallest degree with respect to the graph induced by the yet unselected vertices that are not adjacent to any previously selected vertex. Let d_v denote the degree of vertex v in graph $G_k = (V_k, E_k)$. The greedy choice is to select a vertex with the smallest degree. Instead of selecting the greedy choice, the GRASP construction phase builds a restricted candidate list of all vertices having small degree, but not necessarily the smallest degree. Let Γ be the smallest degree of vertices in V_k, i.e.,

$$\Gamma = \min\{d_v \mid v \in V_k\},$$

and let $\alpha > 0$ be the restricted candidate parameter. The value restricted candidate list is

$$\text{RCL} = \{v \in V_k \mid \ d_v < (1+\alpha)\Gamma\}.$$

From the candidate list a vertex, say v, is selected at random and placed in the independent set, i.e., $S^* \leftarrow S^* \cup \{v\}$.

The greedy function is adaptive, because with the addition of each new vertex in the independent set, G_{k+1} is different from G_k, and consequently vertex degrees change. G_{k+1} is defined as follows: $V_{k+1} = V_k \setminus \{v\} \setminus \text{adj}(v)$, where adj(v) is the set of vertices in G_k adjacent to v; $E_{k+1} = E \setminus \{(u,w) \mid u \in S^* \text{ or } w \in S^*\}$.

Consider the example of Figure 12. Let $\alpha = 0.6$ in this case. Vertices $\{a,b,c,d,f\}$ each have degree 2, while vertex e has degree 4. Hence, the value restricted candidate list $\text{RCL} = \{a,b,c,d,f\}$. Suppose vertex a were to be selected at random from the RCL. The initial independent set would be $S^* = \{a\}$ and the resulting graph G_1 would be the one depicted in Figure 13. In graph G_1, all vertices have identical degree and consequently $\text{RCL} = \{c,e,f\}$. If vertex c were to be selected, the resulting independent set of size 2 would be $S^* = \{a,c\}$. If instead, b was initially chosen ($S^* = \{b\}$), the resulting graph G_1 would be the one depicted in Figure 14. In that case, the restricted candidate list $\text{RCL} = \{c,d,f\}$. Selecting vertex d and then vertex c results in an optimal independent set $S^* = \{b,c,d\}$.

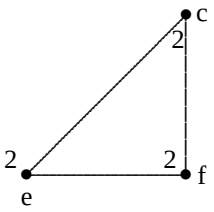

Figure 13. Maximum independent set: construction phase

Figure 14. Maximum independent set: construction phase

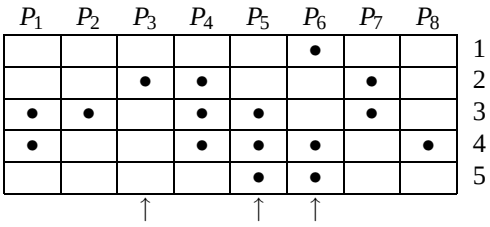

Figure 15. Set covering: local search phase (cover $\{P_3, P_5, P_6\}$)

2.3. GRASP local search phase

We now turn our attention to the local search phase for each of the two examples. We begin with a local search procedure for the set covering problem and then describe a procedure for maximum independent set.

2.3.1. Set covering problem

In the set covering problem, define a k, p-exchange as follows: For all k-tuples in a cover J^*, if possible, exchange the k-tuple with a p-tuple ($p < k$) not in J^*. Consider the example in Figure 15 with cover $J^* = \{P_3, P_5, P_6\}$. Applying the $2,1$-exchange that replaces the 2-tuple $\{P_3, P_5\}$ with the 1-tuple $\{P_4\}$ results in an optimal cover $J^* = \{P_4, P_6\}$ depicted in Figure 16.

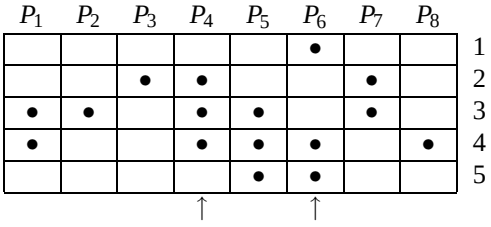

Figure 16. Set covering: local search phase (optimal cover $\{P_4, P_6\}$)

```
procedure local(V,E,S,k)
1     for each k-tuple {v_{i_1},...,v_{i_k}} ∈ S →
2           S' := S \ {v_{i_1},...,v_{i_k}};
3           A := {w ∈ V | (w,v_i) ∉ E, ∀v_i ∈ S'};
4           Apply exhaustive search to graph induced by A to find N;
5           if | N | > k →
6                 S := S' ∪ N;
7                 local(V,E,S,k);
8           fi;
9     rof;
end local;
```

Figure 17. Local search pseudo-code: maximum independent set

2.3.2. Maximum independent set problem

We next describe a k-exchange search procedure for maximum independent set in the graph $G = (V, E)$. The idea here is to take as input an independent set $S \subseteq V$ of size p and consider all k-tuples of vertices in S, for a given parameter k, $0 < k < p$. For each such k-tuple $\{v_{i_1}, \ldots, v_{i_k}\}$, apply an exhaustive search to find a maximum independent set in the graph induced by the vertices of G not adjacent to the vertices in the set $S' = S \setminus \{v_{i_1}, \ldots, v_{i_k}\}$. If the resulting independent set N is larger than S, the set of vertices $S' \cup N$ is an independent set and is larger than S. The procedure can now be applied to the new independent set. This procedure is given in Figure 17.

Consider the example in Figure 18, where a 1-exchange ($v_1 = a$) is carried out on the independent set $\{a, c\}$. There, the set $S' = S \setminus \{a\} = \{c\}$, so the exhaustive enumeration is done of the graph consisting of vertices $\{a, b, d\}$ and edges $\{(a, b), (a, d)\}$, resulting in the maximum independent set $N = \{b, d\}$. Since this set has size 2, the new larger independent set $\{b, c, d\}$ can be built. Applying the local search on this new independent set does not produce an improvement, thus halting the procedure at this local minimum.

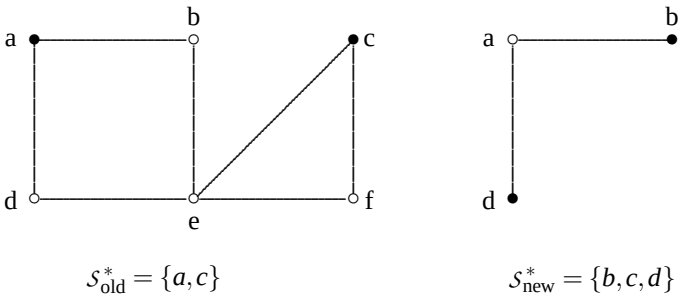

Figure 18. Local search example: maximum independent set

Problem	Size Columns/Rows	Best Known Cover	Optimal?
A_{45}	45/330	30	yes
A_{81}	81/1080	61	unknown
A_{135}	135/3015	105	unknown
A_{243}	243/9801	203	unknown

Figure 19. Experimental results: set covering problem statistics

2.4. Experimental results

To conclude this section, we describe experimental results of running GRASP implementations on the two classes of problems described in this section. The codes used were implemented by Feo and Resende [12] for the set covering problem and Feo, Resende and Smith [13] for the maximum independent set problem. The codes are run on a single 150 MHz MIPS 4400 processor of a Silicon Graphics Challenge computer. Both codes are written in Fortran and were compiled with the `f77` compiler using flags `-O2 -Olimit 800`. Running times were computed with the system routine `etime`.

2.4.1. Set covering problem

Fulkerson, Nemhauser and Trotter [18] proposed a class of small, yet difficult set covering problems that arise when computing the 1-width of incidence matrices of Steiner triple systems. To illustrate a GRASP for set covering, we consider the following instances from this class: A_{45}, A_{81}, A_{135}, and A_{243}. Figure 19 summarizes some statistics for these problems. Of the four instances, only the smallest has a known optimal solution. The GRASP was run with five values of the restricted candidate list parameter α: 0.5, 0.6, 0.7, 0.8, and 0.9. For each parameter setting, 10 runs were carried out for each instance, varying the initial seed of the random number generator. The local search phase consisted of only 1,0-exchanges, i.e., the GRASP eliminated any superfluous columns.

α	cover size	times found	iterations			cpu seconds		
			min	avg	max	min	avg	max
0.5	31	10	5	42.2	111	0.01	0.06	0.15
	30	4	691	2641.5	4606	0.92	3.50	6.11
0.6	31	10	3	23.7	58	0.01	0.04	0.08
	30	8	40	3111.5	6246	0.08	4.33	9.95
0.7	31	10	1	25.7	56	0.00	0.04	0.07
	30	9	594	2982.3	7329	0.72	3.71	8.91
0.8	31	10	1	6.9	27	0.00	0.01	0.03
	30	10	121	1276.0	3589	0.15	1.45	4.08
0.9	31	10	1	7.1	28	0.00	0.01	0.04
	30	5	2799	7129.4	8919	3.27	8.38	10.34

Figure 20. Experimental results: GRASP solution statistics (A_{45})

α	cover size	times found	iterations			cpu seconds		
			min	avg	max	min	avg	max
0.5	63	9	1	5.9	18	0.02	0.05	0.12
	62	1	894	894.0	894	4.95	4.95	4.95
	61	4	1	51.8	154	0.02	0.30	0.86
0.6	63	10	1	5.1	11	0.02	0.04	0.07
	62	3	249	443.7	725	1.34	2.36	3.85
	61	2	418	448.0	478	2.21	2.37	2.52
0.7	63	10	2	2.9	6	0.02	0.03	0.04
	62	5	39	409.0	996	0.20	2.04	5.01
	61	4	197	416.5	736	0.97	2.05	3.60
0.8	63	10	1	3.5	7	0.02	0.03	0.05
	62	3	162	445.3	747	0.76	2.10	3.45
	61	7	20	486.7	893	0.11	2.24	4.08
0.9	63	10	1	4.0	6	0.02	0.03	0.05
	62	3	22	277.3	627	0.12	1.33	2.98
	61	1	718	718.0	718	3.42	3.42	3.42

Figure 21. Experimental results: GRASP solution statistics (A_{81})

	cover	times	iterations			cpu seconds		
α	size	found	min	avg	max	min	avg	max
0.5	107	10	1	86.1	243	0.07	2.00	5.58
	106	9	517	3532.0	8379	11.64	79.84	186.74
	105	1	3787	3787.0	3787	86.26	86.26	86.26
	104	0	-	-	-	-	-	-
0.6	107	10	2	42.5	143	0.09	0.98	3.12
	106	10	863	2704.2	6330	18.52	59.92	159.25
	105	1	5992	5992.0	5992	125.44	125.44	125.44
	104	0	-	-	-	-	-	-
0.7	107	9	2	46.7	110	0.08	0.94	2.19
	106	10	48	765.9	3149	1.00	14.67	59.83
	105	3	1930	3087.0	3773	38.19	60.46	74.06
	104	0	-	-	-	-	-	-
0.8	107	8	3	6.8	18	0.09	0.16	0.35
	106	8	20	121.3	279	0.37	2.16	4.84
	105	10	6	1835.6	5299	0.14	31.18	86.89
	104	2	4635	5747.5	6860	90.33	102.75	115.16
0.9	107	9	1	3.2	6	0.06	0.10	0.15
	106	9	15	51.0	107	0.28	0.87	1.78
	105	10	1	790.3	2402	0.06	12.84	38.55
	104	3	2651	5184.3	8807	42.55	80.97	135.63

Figure 22. Experimental results: GRASP solution statistics (A_{135})

	cover	times	iterations			cpu seconds		
α	size	found	min	avg	max	min	avg	max
0.5	206	8	10	23.6	76	1.40	3.15	9.95
	205	10	8	344.6	1490	1.17	43.18	184.13
	204	5	2430	2895.2	3988	309.55	364.61	496.35
	203	0	-	-	-	-	-	-
0.6	206	10	6	26.6	66	0.95	3.34	7.94
	205	10	25	359.9	1406	3.15	43.39	169.37
	204	2	1475	3520.0	5565	182.25	427.33	672.42
	203	0	-	-	-	-	-	-
0.7	206	10	1	28.0	70	0.29	3.13	7.61
	205	10	73	398.5	1070	7.62	41.84	112.67
	204	4	2927	4527.5	6613	305.15	469.81	673.42
	203	0	-	-	-	-	-	-
0.8	206	10	2	16.2	59	0.33	1.64	5.60
	205	10	44	176.8	353	4.03	16.03	31.11
	204	10	413	2125.4	5453	36.51	189.67	492.53
	203	2	2581	3153.5	3726	236.98	281.95	326.92
0.9	206	8	2	38.8	148	0.32	3.55	13.46
	205	10	2	414.4	1826	0.32	36.08	163.93
	204	3	591	4347.0	7762	49.38	366.29	645.54
	203	0	-	-	-	-	-	-

Figure 23. Experimental results: GRASP solution statistics (A_{243})

$$
\begin{array}{lll}
E(X_{14}) = 4.23\times 10^{3} & & P(X_{14}=0) \leq 0.02 \\
E(X_{15}) = 1.70\times 10^{1} & & P(X_{15}=0) \leq 0.18 \\
E(X_{16}) = 3.19\times 10^{-2} & & P(X_{16}=0) \leq 1.00 \\
E(X_{17}) = 2.18\times 10^{-5} & & P(X_{17}=0) \leq 1.00
\end{array}
$$

Figure 24. Maximum independent sets in $G_{1000,.5}$

Figures 20, 21, 22, and 23 summarize the GRASP runs for instances A_{45}, A_{81}, A_{135} and A_{243}, respectively. The GRASP found the best known solutions for all of the instances considered. Running times for the two smaller instances were less than 10 cpu seconds in all but one run, while the longest run for the largest class took 673.4 seconds. Varying the parameter α from 0.5 to 0.9 changes the behavior of the GRASP from a more randomized to a more greedy procedure ($\alpha = 0$ is a purely random procedure, while $\alpha = 1$ is purely greedy). In most instances, the GRASP with the parameter value $\alpha = 0.8$ is the best performer. For A_{135}, $\alpha = 0.9$ did slightly better.

2.4.2. Maximum independent set problem

For testing the GRASP on the maximum independent set problem, let us consider the family of undirected random graphs $G_{|V|,p}$. These graphs have $|V|$ vertices, and each edge from the set of edges on the complete graph on $|V|$ vertices appears in $G_{|V|,p}$, independently of the inclusion of any other edge, with probability p. This family of graphs has been studied extensively [3]. We consider here 100 instances of random graphs with parameters $|V| = 1000$ and $p = 0.5$, i.e., the class $G_{1000,.5}$. Let X_k be a stochastic variable denoting the number of independent sets of size k in an instance of $G_{1000,.5}$. Figure 24 shows values of the expectation of X_k and bounds on the the probability that $X_k = 0$. The latter indicates that independent sets of size 15 are abundant in $G_{1000,.5}$, while sets of size 16 are rare. In the initial set of runs, we search for a set of 15 or larger, and stop when such a set is found. Then, in a second set of runs, independent sets of size 16 or larger are sought.

For these examples, we introduce a way to decompose the work for a GRASP. The idea is to condition on *favorable* pairs of vertices being in the independent set, and solve a series of smaller, easier problems (each contracted graph having about 250 vertices). We consider the 50 vertices having the smallest degrees, $V_{low} = \{v_{i_1}, v_{i_2}, \ldots, v_{i_{50}}\}$. For all pairs $v_i, v_j \in V_{low}$, such that $(v_i, v_j) \notin E$, compute $\sigma(\{v_i, v_j\})$, the number of vertices not adjacent to either v_i or v_j. The pairs are ordered in decreasing value of σ, and at most 400 pairs are kept for consideration. The problems on the contracted graphs are solved in order, conditioning on the pairs being in the independent set. Consider, as an example, the graph in Figure 25, where we choose to condition on pairs of vertices from the set of vertices having degree 2, i.e. vertices $\{b, c, f\}$. The pairs that we condition on are $\{b, c\}$, $\{b, f\}$, and

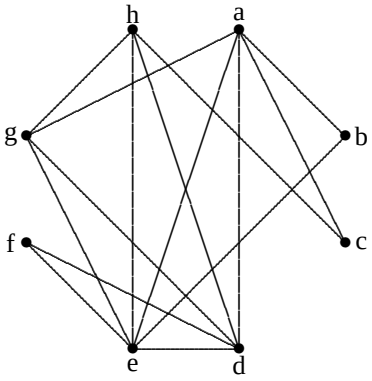

Figure 25. Preprocessing for maximum independent set

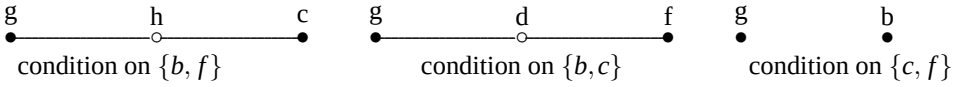

Figure 26. Preprocessing for maximum independent set

$\{c, f\}$. For these pairs, we have $\sigma(\{b,c\}) = |\{d, f, g\}| = 3$, $\sigma(\{b, f\}) = |\{c, g, h\}| = 3$, and $\sigma(\{c, f\}) = |\{b, g\}| = 2$. Figure 26 shows the contracted graphs induced by conditioning on pairs $\{b, f\}$, $\{b, c\}$ and $\{c, f\}$, along with the maximum independent sets of each graph. Together, with the conditioned pairs, we get the independent sets $\{c, f, g\}$, $\{b, c, f, g\}$, and $\{b, c, f, g\}$, of which the set of size 4 is optimal.

In our experiments, for each conditioned instance, at most 100 GRASP iterations are performed, using candidate list parameter $\alpha = 0.1$. Local search is carried out only if the independent set found in the 250 node graph is of size 11 or greater. We use the *k*-exchange local search described in Section 2.3.2 with parameter $k = 2$. Figure 27 summarizes the GRASP runs on the 100 instances of maximum independent set problems on $G_{1000,.5}$. The entries have been sorted in increasing order of running times and are summarized in sets of 10 runs, e.g., the first row summarizes the runs for the ten instances with the fastest running times, the second row for the second ten fastest times, etc. The table lists the minimum, average, and maximum cpu times for setting up the 400 conditioning tuples (preproc seconds), the minimum, average, and maximum number of tuples examined until a set of size 15 or greater is found, and the minimum, average, and maximum cpu times, in seconds, taken by the GRASP to find the independent sets.

Of the one hundred runs stopped when the GRASP found a set of size 15 or greater, independent sets of size 15 were found in 98 instances and of size 16 in two instances. In more than half of the runs, the GRASP took less than one minute of cpu time to terminate. The code was run on the same instances to search for sets of size 16 or greater. There, the code found the two size 16 sets found in the first set of runs, along with another set of size 16, totaling three instances with independent sets of size 16 out of the 100 tested. For

	preproc seconds			tuples examined			GRASP seconds		
instances	min	avg	max	min	avg	max	min	avg	max
1–10	0.41	0.42	0.43	1	2.1	4	0.12	3.77	9.26
11–20	0.40	0.42	0.43	4	4.8	8	9.70	12.36	19.66
21–30	0.39	0.41	0.42	7	8.6	10	20.24	22.93	28.72
31–40	0.41	0.42	0.43	11	12.2	14	29.50	33.11	38.56
41–50	0.40	0.42	0.43	15	17.0	20	41.96	47.86	55.36
51–60	0.41	0.42	0.43	20	24.8	29	59.20	69.89	82.67
61–70	0.41	0.42	0.43	32	34.6	38	88.87	98.64	110.31
71–80	0.40	0.42	0.44	39	50.5	66	110.41	141.66	196.69
81–90	0.40	0.42	0.43	73	88.2	116	203.64	247.24	315.17
91–100	0.41	0.42	0.44	114	173.8	314	324.68	489.28	893.19

Figure 27. Experimental results: GRASP maximum independent set solution statistics

those runs, the preprocessing times were .42, .45, and .45 seconds; the number of tuples examined were 35, 16, and 76; and the GRASP running times were 101.88, 319.39, and 217.77 seconds.

3. Applications

We now turn our attention to a number of GRASP implementations that have appeared in the literature, covering a wide range of applications, including set covering, production planning and scheduling, graph problems, location problems, quadratic assignment problems, and problems in logic. Two industrial implementations of GRASP are also discussed.

3.1. Set covering

Feo and Resende [12] describe a GRASP for solving set covering problems that arise in computing the 1-width of the incidence matrix of Steiner triple systems. The construction mechanism as well as the local search strategy of that GRASP are described in Section 2 of this paper. Computational results are described, where the GRASP quickly produces best known solutions for all of the instances considered.

Bard and Feo [2] describe a unified framework in which product and process demands can be related to manufacturing system requirements. The objective is to determine, in a flexible manufacturing environment, how many of each machine to purchase, as well as what fraction of the time each piece of equipment will be configured for a particular type of operation. A nonlinear cost minimization model is developed and is solved with a depth-first branch and bound routine that employs a GRASP for set covering to find good feasible solutions. The solutions obtained with the GRASP permit early fathoming and greatly contribute to the efficiency of the algorithm.

Feo and Bard [9] use GRASP to solve a sequence of set covering problems in an approach that renders an approximate solution to a minimum cost, multicommodity, network flow problem with integral constraints for airline flight scheduling and maintenance base planning. They demonstrate the procedure with data for the American Airlines Boeing 727 fleet, and show that the new approach is a significant improvement over current solution techniques.

3.2. Production planning and scheduling

Bard and Feo [1], [10] apply GRASP to computer aided process planning, specifically, the selection of tools and cutting paths for milling metal on flexible manufacturing machines. The underlying optimization problem is modeled as an integer program and is solved by branch and bound. Lower bounds are calculated by means of a Lagrangian relaxation technique. Feasible solutions (upper bounds) are found by a GRASP applied to a specialized set covering problem. Overall performance of the method, including quality of solutions and cpu requirements, is judged by examining a wide variety of instances derived from actual manufacturing data.

Laguna and González-Velarde [29] consider the scheduling of parallel machines in a just-in-time production environment. The optimization problem possesses a weighted earliness penalty with deadlines for identical parallel machines. The authors present a hybrid heuristic that combines elements of both tabu search and GRASP methodologies, and uses a branch-and-bound postprocessor. They compare the performance of their method with the modified Smith heuristic of Chand and Scheeberger [6], concluding that their method succeeds in finding solutions that are, on average, 10 percent better than those found by the modified Smith heuristic.

Feo, Venkatraman, and Bard [15] develop a GRASP for a single machine scheduling problem with flow time and earliness penalties. The method compares favorably with methods previously reported in the literature. A dynamic programming (DP) algorithm yields optimal solutions to problems with up to 30 jobs. In a fraction of the time required by the DP implementation, the GRASP code provides optimal solutions to 238 out of the 240 instances tested, while providing solutions that are extremely close to the optimal in the remaining two instances.

Feo, Sarathy, and McGahan [14] write about a single machine scheduling problem with sequence dependent setup costs and linear delay penalties. They develop a GRASP which quickly finds optimal solutions to 20-job problems previously reported in the literature. The method is favorably compared to a tabu search implementation on instances ranging up to 165 jobs. The authors take advantage of the mutation concept found in genetic algorithms to enhance the performance of the local search phase of their GRASP implementation.

Klincewicz and Rajan [27] describe two GRASP heuristics to solve the component grouping problem, a type of set partitioning problem that arises in a number of manufacturing and material logistics applications. In computational results, based on real manufacturing data, the GRASPs produce solutions having objective function values within 4.3 to 9.5 percent (7.4 percent on average) of a lower bound based on a combinatorial argument. Compared

to previously used methods based on a network flow heuristic [33], the first GRASP produced better solutions on all 12 test problems, while the second GRASP produced better solutions on all but one.

Feo, Bard, and Holland [8] present a GRASP implementation for scheduling printed wiring board assembly. The combinatorial optimization problem possesses multiple machines, precedence relationships, start dates, due dates, capacity constraints, set up times, processing times, and resource constraints. A multicriterion objective is considered that includes minimizing weighted tardiness, maximizing revenue (weighted throughput), minimizing cycle times, and flowline balancing. The GRASP is empirically validated in an industrial setting with over 70 processing stations, 140 product types, 4500 components, 126 shifts, 49,000 boards in WIP, and 142,000 boards on demand. The heuristic is shown to outperform rule based methods used previously. This work highlights the ease and effectiveness with which GRASP can be applied to extremely large and complex optimization problems found in practice.

3.3. Graph problems

Feo, Resende and Smith [13] describe a GRASP for finding large independent sets on sparse random graphs. The construction and local search phases of that GRASP are described in Section 2 of this paper. The GRASP is implemented in parallel on a MIMD computer by assigning to different processors the different contracted graphs induced by the conditioning-on-pairs strategy described in Subsection 2.4.2 of this paper. The efficiency (speedup divided by the ratio of processors) of going from one to eight processors was, on average, 93.6 percent. The GRASP was tested on graphs with up 3500 nodes and over 3 million edges and is compared with implementations of simulated annealing, tabu search, and an exact method. The GRASP found larger independent sets, and in substantially less cpu time, than the simulated annealing implementation. GRASP was compared with the tabu search code STABULUS [17] on three classes of random graphs, having 600, 1500, and 3500 vertices. The tabu search code was 1.6 times faster on the 600-node graphs, but was 3.7 times and over 10 times slower on the 1500-node and 3500-node graphs, respectively. On 600-node graphs, the exact method of Carraghan and Pardalos [5] produced optimal solutions on all 25 instances tested, while the GRASP rarely produced optimal solutions. However, to produce the certificate of optimality, the exact method required about 40 times more cpu time than needed by the GRASP to produce independent sets having one vertex less than the optimal size. For a 1000-node graph, the exact method failed to find an optimal solution in 10 cpu days of computing, while GRASP quickly found probably-optimal sets of size 15 or 16 in all 200 instances tested.

Feo and Smith [37] offer a GRASP for coloring sparse graphs. The construction phase builds one color class at a time by identifying maximal independent sets. The local search phase uses a simulated annealing approach starting at a relatively cold temperature. This starting condition keeps the search in the vicinity of the constructed solution while allowing it to wander away from local minima. The GRASP implementation performs well on a wide range of instances including random graphs and graphs of known chromatic number.

Laguna, Feo, and Elrod [28] develop a GRASP implementation for the network 2-partition problem. The heuristic is conceptually simple and straightforward to program. The GRASP is empirically compared to the Kernighan-Lin method [24] which stood for over twenty years as the dominating heuristic procedure. Over 3500 instances are used to compare the running times and solution values provided by the two methods. The instances include a wide variety of random and geometric graphs, as well as smaller examples for which optimal solutions can be found via branch and bound. The comparative study empirically confirms the effectiveness of the GRASP implementation.

3.4. Location problems

Klincewicz [26] compares tabu search and GRASP for solving instances of the discrete p-hub location problem, a problem that has applications in airline and package delivery systems, as well as in certain telecommunications network design problems. In this problem, one is given an n-node graph and a matrix of internodal traffic and is asked to choose p of the n nodes to serve as hubs, which are to be fully interconnected. For all nonhub nodes, one must also determine which hub that node is to be connected to, making it possible to route traffic between any two nodes in the graph. The objective is to minimize the total cost of sending traffic between demand pairs. Computational testing was carried out on real data for airline hub design ($n = 10, 15, 25$, $p = 3, 4$) and a packet network design problem ($n = 52$, $p = 4, 10$). The author concludes that while the tabu search implementation was about twice as fast as the GRASP code in producing the best solution, GRASP found solutions having the best known value more often.

3.5. Quadratic assignment problems

Feo and González-Velarde [11] apply GRASP to a quadratic assignment problem (QAP) that models the positioning of intermodal highway trailers on railcars. The GRASP heuristic is used within a branch and bound scheme to provide optimal solutions. The heuristic is observed to be extremely fast, and by itself, finds optimal solutions to all problem instances furnished over a two-year period by Consolidated Rail Corporation (Conrail).

Li, Pardalos, and Resende [31] propose a GRASP for the classical quadratic assignment problem, where one wants to assign, at minimum cost, n facilities (with interfacility flow demands) to n sites. The cost of assigning facility i to site k and facility j to site l is $f_{i,j} \cdot d_{k,l}$, where $f_{i,j}$ is the flow between facilities i and j, and $d_{k,l}$ is the distance between sites k and l. The GRASP was tested on 88 instances of QAP, most of which are from QAPLIB [4], a library of QAP test problems. The GRASP found the best known solution of almost all of the instances, and improved on the best known solution in a few cases. FORTRAN subroutines of this GRASP are described in [35].

3.6. Problems in logic

Resende and Feo [34] describe several GRASP implementations for the satisfiability problem in logic. In the satisfiability problem one wants to find a truth assignment to Boolean variables to make a given Boolean formula evaluate to **true** or prove that no such assignment exists. The GRASPs tested attempt to find an assignment and are not capable of proving unsatisfiability. The codes were tested on most satisfiable instances of the benchmark collection of the Second DIMACS Algorithm Implementation Challenge [22] and compared with GSAT [36], a code that has recently received much attention due to its ability to find satisfying truth assignments of large formulae. The GRASPs found satisfiable assignments on all 114 instances tested. The GRASPs were faster than GSAT in three out of the five problem classes tested. Furthermore, GSAT failed to produce satisfiable assignments to several formulae for which the GRASPs were successful.

3.7. Industrial applications

GRASP has been directly applied in practice as part of two large scale decision support systems developed and implemented by Optimization Alternatives, an information systems development firm in Austin, Texas.

INSITES^{TM} (Integrated Scheduling, Inventory, and Throughput Evaluation System) provides facility-wide planning and scheduling functions for printed wire board assembly operations. The GRASP used in INSITES is described in Feo, Bard, and Holland [8]. The success of this management information system at Texas Instruments is discussed in Feo, Bard, and Holland [7].

OASIS^{TM} (Optimization Alternatives' Strategic Intermodal Scheduler) controls the logistics operations in an intermodal rail terminal. The system tracks all inventory in the yard and directs parking activities to maximize the utilization of the terminal's parking areas. It issues hostler and packer work orders through a radio frequency (RF) interface to speed operations and handle greater volumes of traffic with less equipment and personnel. It optimizes load plans for both trailers and containers, and thus, improves railcar utilization. The GRASP found in OASIS is used for optimizing the load plans and is based in part on the work of Feo and González-Velarde [11], discussed previously. OASIS is currently in use at several Conrail terminals and will be deployed at all major Conrail and Union Pacific intermodal yards by 1996.

4. Concluding remarks

GRASP possesses characteristics found in and shared by other heuristic search methodologies. Close analogies can be drawn to simulated annealing, tabu search, and genetic algorithms. The implementations of these various approaches are certainly quite different in practice. However, they all share with GRASP fundamental heuristic concepts that can be used to classify their operations. The next two paragraphs give a terse description of simulated annealing, tabu search, and genetic algorithms. The remainder of the conclu-

sion offers several thoughts regarding a classification schema for these and other heuristic methodologies.

Tabu search and simulated annealing contain local search procedures that explore the neighborhood around a current solution for improvements to that solution. Each has the ability to remove itself from local optima in order to find better if not optimal solutions. Simulated annealing uses a straightforward randomization technique. Tabu search in its simplest form uses a short term memory strategy to intelligently direct its search away from neighborhoods already considered. Medium and long term memory strategies are respectively used in tabu search to allow for search intensification and diversification with regard to a known set of promising solutions.

Genetic algorithms (GA) apply crossover and mutation operations to a population of solutions. Crossover mates two solutions in the population by combining attributes of the solutions to form an offspring. The offspring is then mutated by randomly altering a few of its attributes. The offspring is added to the population if its solution value compares favorably with the other solution values in the population, thus resembling natural selection in the theory of evolution.

Categories of fundamental heuristic concepts include: solution construction, solution perturbation, procedure repetition and restart criteria, problem decomposition or conditioning, and procedure stopping rules. For illustrative purposes consider the category of solution perturbation. A local search mechanism, such as a 2-exchange technique or a mutation operation found in a genetic algorithm, are examples of solution perturbation. The basic principle is to move from one solution to another. For each of the categories, a wide variety of mechanisms have been devised and even combined to form hybrid techniques.

Guiding the design of mechanisms in each category are two goals. The first is to find an optimum or near optimum solution. The second is to arrive at such a solution with a minimal amount of computational effort. Given that most combinatorial optimization problems are classified as intractable and have enormous solution spaces, it is very often ineffective to apply the brute force technique of exhaustive enumeration. Thus, one must strategically search for good solutions, biasing the search to consider only a minuscule fraction of all possibilities.

Biases in heuristic mechanisms are sometimes referred to as intelligence. They can be grouped as follows: Random or lexicographic bias – indiscriminate selection of alternatives; Greedy or simple decent bias – selection based on the problem's objective function; Memory bias – selection based on prior selections; Experience or target bias – selection based on prior performance. Consider the following partial illustrations. GRASP uses a greedy bias to guide the construction of each new solution. Simulating annealing uses a random bias to perturb its current solution. Tabu search employs a short term memory bias, while genetic algorithms possess a subtle experience bias analogous to natural selection. Explicit examples of experience bias are also apparent in mechanisms employing the dynamic application of target analysis.

GRASP and the other methods discussed herein have contributed enormously to our ability to empirically find good solutions to otherwise unsolved instances of practical combinatorial optimization problems. Fortunately, these methodologies are not antithetical to one another. They each possess characteristics that can be combined in an enormous num-

```
procedure grasp()
1    InputInstance();
2    for Grasp stopping criterion not satisfied →
3         ConstructGreedyRandomizedSolution(Solution);
4         for local search stopping criterion not satisfied →
5              LocalSearch(Solution);
6              UpdateSolution(Solution,BestSolutionFound);
7              MutateSolution(Solution);
8         rof;
9         UpdateSolution(Solution,BestSolutionFound);
10   rof;
11   return(BestSolutionFound)
end grasp;
```

Figure 28. Adding mutation concept to GRASP local search phase

ber of ways yet to be explored. As an example, consider the hybrid procedure, developed by Feo, Sarathy, and McGahan [14], depicted in Figure 28. The framework is GRASP-based, yet the mutation introduced in Phase 2 is borrowed from the GA methodology. A future direction of research into the design of heuristics should include an expansion of the classification schema started here. The motivation for this work is abundant. First, it will improve our ability to describe and define heuristic methodologies and allow us to conceptually compare different approaches. Second, it will guide the enhancement efforts of existing procedures that will lead to improved hybrid methods. And finally, it may evolve into a theoretical framework capable of blossoming the currently limited discipline of probabilistic analysis of heuristics.

References

1. J.F. Bard and T.A. Feo. Operations sequencing in discrete parts manufacturing. *Management Science*, 35:249–255, 1989.
2. J.F. Bard and T.A. Feo. An algorithm for the manufacturing equipment selection problem. *IIE Transactions*, 23:83–92, 1991.
3. B. Bollobás. *Random graphs*. Academic Press, 1985.
4. R. Burkhard, S. Karisch, and F. Rendl. QAPLIB – A quadratic assignment problem library. *European Journal of Operational Research*, 21, 1991.
5. R. Carraghan and P.M. Pardalos. An exact algorithm for the maximum clique problem. *Operations Research Letters*, 9, 1990.
6. S. Chand and H. Schneeberger. Single machine scheduling to minimize earliness subject to no tardy jobs. *European Journal of Operational Research*, 34:221–230, 1988.
7. T.A. Feo, J. Bard, and S. Holland. Facility-wide planning and scheduling of printed wiring board assembly. Technical report, Operations Research Group, Department of Mechanical Engineering, The University of Texas at Austin, Austin, TX 78712-1063, February 1993.
8. T.A. Feo, J. Bard, and S. Holland. A GRASP for scheduling printed wiring board assembly. Technical report, Operations Research Group, Department of Mechanical Engineering, The University of Texas at Austin, Austin, TX 78712-1063, December 1993.

9. T.A. Feo and J.F. Bard. Flight scheduling and maintenance base planning. *Management Science*, 35:1415–1432, 1989.
10. T.A. Feo and J.F. Bard. The cutting path and tool selection problem in computer-aided process planning. *Journal of Manufacturing Systems*, 8:17–26, 1989.
11. T.A. Feo and J.L. González-Velarde. The intermodal trailer assignment problem. Technical report, Operations Research Group, Department of Mechanical Engineering, The University of Texas at Austin, Austin, TX 78712-1063, April 1992.
12. 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.
13. T.A. Feo, M.G.C. Resende, and S.H. Smith. A greedy randomized adaptive search procedure for a maximum independent set. *Operations Research*, 42(5), 1994.
14. T.A. Feo, K. Sarathy, and J. McGahan. A GRASP for single machine scheduling with sequence dependent setup costs and linear delay penalties. Technical report, Operations Research Group, Department of Mechanical Engineering, The University of Texas at Austin, Austin, TX 78712-1063, January 1994.
15. T.A. Feo, K. Venkatraman, and J.F. Bard. A GRASP for a difficult single machine scheduling problem. *Computers and Operations Research*, 18, 1991.
16. M. Fischetti, S. Martello, and P. Toth. The fixed job scheduling problem with spread-time constraints. *Operations Research*, 35:849–858, 1987.
17. C. Friden, A. Hertz, and D. de Werra. Stabulus: A technique for finding stable sets in large graphs with tabu search. *Computing*, 42:35–44, 1989.
18. D.R. Fulkerson, G.L. Nemhauser, and L.E. Trotter Jr. Two computationally difficult set covering problems that arise in computing the 1-width of incidence matrices of Steiner triple systems. *Mathematical Programming Study*, 2:72–81, 1974.
19. F. Glover. Tabu search – Part I. *ORSA Journal on Computing*, 1:190–206, 1989.
20. F. Glover. Tabu search – Part II. *ORSA Journal on Computing*, 2:4–32, 1990.
21. D.E Goldberg. *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley, 1989.
22. D.S. Johnson and M. Trick, editors. *The Second DIMACS Algorithm Implementation Challenge: Maximum Clique, Coloring, and Satisfiability*. DIMACS Series on Discrete Mathematics and Theoretical Computer Science. American Mathematical Society, 1995.
23. N.K. Karmarkar and K.G. Ramakrishnan. Computational results of an interior point algorithm for large scale linear programming. *Mathematical Programming*, 52:555–586, 1991.
24. B.W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49:291–307, 1970.
25. S. Kirkpatrick. Optimization by simulated annealing: Quantitative studies. *Journal of Statistical Physics*, 34:975–986, 1984.
26. J.G. Klincewicz. Avoiding local optima in the *p*-hub location problem using tabu search and GRASP. *Annals of Operations Research*, 40:283–302, 1992.
27. J.G. Klincewicz and A. Rajan. Using GRASP to solve the component grouping problem. Technical report, AT&T Bell Laboratories, Holmdel, NJ, 1992. To appear in *Naval Research Logistics*.
28. M. Laguna, T.A. Feo, and H.C. Elrod. A greedy randomized adaptive search procedure for the 2-partition problem. *Operations Research*, 42:677–687, 1994.
29. M. Laguna and J.L. González-Velarde. A search heuristic for just-in-time scheduling in parallel machines. *Journal of Intelligent Manufacturing*, 2:253–260, 1991.
30. E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. *The traveling salesman problem*. John Wiley, 1985.
31. Y. Li, P.M. Pardalos, and M.G.C. Resende. A greedy randomized adaptive search procedure for the quadratic assignment problem. In P.M. Pardalos and H. Wolkowicz, editors, *Quadratic assignment and related problems*, volume 16 of *DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 237–261. American Mathematical Society, 1994.
32. C.H. Papadimitriou and K. Steiglitz. *Combinatorial optimization: Algorithms and complexity*. Prentice-Hall, 1982.
33. A. Rajan and M. Segal. Assigning components to robotic workcells for electronic assembly. *AT&T Technical Journal*, 68:93–102, 1989.
34. M.G.C. Resende and T.A. Feo. A GRASP for Satisfiability. Technical report, AT&T Bell Laboratories, Murray Hill, NJ, 1994.

35. M.G.C. Resende, P.M. Pardalos, and Y. Li. FORTRAN subroutines for approximate solution of dense quadratic assignment problems using GRASP. Technical report, AT&T Bell Laboratories, Murray Hill, NJ, 1994.
36. B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard Satisfiability problems. In *Proceedings of the Tenth National Conference on Artificial Intelligence (AAAI-92)*, pages 440–446, July 1992.
37. S.H. Smith and T.A. Feo. A GRASP for coloring sparse graphs. Technical report, Operations Research Group, Department of Mechanical Engineering, The University of Texas at Austin, Austin, TX 78712-1063, January 1991.