RANDOMIZED HEURISTICS FOR THE MAX-CUT PROBLEM

P. FESTA[a]*, P. M. PARDALOS[b]†, M. G. C. RESENDE[c]‡ and C. C. RIBEIRO[d]§

[a] *University of Napoli FEDERICO II, Napoli, Italy*;
[b] *University of Florida, Gainesville, FL, USA*;
[c] *AT&T Labs Research, Florham Park, NJ, USA*;
[d] *Catholic University of Rio de Janeiro, Brazil*

Abstract

Given an undirected graph with edge weights, the MAX-CUT problem consists in finding a partition of the nodes into two subsets, such that the sum of the weights of the edges having endpoints in different subsets is maximized. It is a well-known NP-hard problem with applications in several fields, including VLSI design and statistical physics. In this paper, a greedy randomized adaptive search procedure (GRASP), a variable neighborhood search (VNS), and a path-relinking (PR) intensification heuristic for MAX-CUT are proposed and tested. New hybrid heuristics that combine GRASP, VNS, and PR are also proposed and tested. Computational results indicate that these randomized heuristics find near-optimal solutions. On a set of standard test problems, new best known solutions were produced for many of the instances.

1 INTRODUCTION

Given an undirected graph $G = (V, E)$, where $V = \{1, \ldots, n\}$ is the set of vertices and E is the set of edges, and weights w_{ij} associated with the edges $(i, j) \in E$, the MAX-CUT problem consists in finding a subset of vertices S such that the weight of the cut $(S, \bar{S})$ given by

$$w(S, \bar{S}) = \sum_{i \in S, j \in \bar{S}} w_{ij}$$

is maximized. The decision version of the MAX-CUT problem was proved to be NP-complete by Karp [27]. Applications are found in VLSI design and statistical physics,

*E-mail: paola.festa@unina.it
†E-mail: pardalos@ufl.edu
‡Corresponding author. E-mail: mgcr@att.com
§E-mail: celso@inf.puc-rio.br

see e.g. [4, 10, 11, 33] among others. The reader is referred to Poljak and Tuza [35] for an introductory survey.

The MAX-CUT problem can be formulated as the following integer quadratic program:

$$\max \frac{1}{2} \sum_{1 \le i < j \le n} w_{ij}(1 - y_i y_j)$$

subject to

$$y_i \in \{-1, 1\} \ \ \forall\, i \in V.$$

Each set $S = \{i \in V : y_i = 1\}$ induces a cut $(S, \bar{S})$ with weight

$$w(S, \bar{S}) = \frac{1}{2} \sum_{1 \le i < j \le n} w_{ij}(1 - y_i y_j).$$

In recent years, several continuous and semidefinite programming relaxations of the above formulation have been considered. The idea that the MAX-CUT problem can be naturally relaxed to a semidefinite programming problem was first observed by Lovász [29] and Shor [41]. Goemans and Williamson [22] proposed a randomized algorithm that uses semidefinite programming to achieve a performance guarantee of 0.87856 if the weights are nonnegative. Since then, many approximation algorithms for NP-hard problems have been devised using SDP relaxations [22, 26, 34].

More recent algorithms for solving the semidefinite programming relaxation are particularly efficient, because they explore the structure of the MAX-CUT problem. One approach along this line is the use of interior-point methods [6, 16, 17]. In particular, Benson, Ye, and Zhang [6] used the semidefinite relaxation for approximating combinatorial and quadratic optimization problems subject to linear, quadratic, and Boolean constraints. They proposed a dual potential reduction algorithm that exploits the sparse structure of the relaxation.

Other nonlinear programming approaches have also been presented for the MAX-CUT semidefinite programming relaxation [24, 25]. Homer and Peinado [25] reformulated the constrained problem as an unconstrained one and used the standard steepest ascent method on the latter. A variant of the Homer and Peinado algorithm was proposed by Burer and Monteiro [7]. Their idea is based on the constrained nonlinear programming reformulation of the MAX-CUT semidefinite programming relaxation obtained by a change of variables.

More recently, Burer, Monteiro, and Zhang [8] proposed a rank-2 relaxation heuristic for MAX-CUT and described a computer code, called `circut`, that produces better solutions in practice than the randomized algorithm of Goemans and Williamson.

The remainder of this paper is organized as follows. In Section 2 we propose various randomized heuristics for finding approximate solutions of the MAX-CUT problem, based on the instantiation of several metaheuristics and their hybrids. Computational results are reported in Section 3. Concluding remarks are given in the last section.

2 RANDOMIZED HEURISTICS

Recent surveys on randomized metaheuristics can be found in [32]. Almost all randomization effort in implementations of the GRASP (greedy randomized adaptive search procedure) metaheuristic [12, 13, 37] involves the construction phase. On the other hand, strategies such as VNS (Variable Neighborhood Search) and VND (Variable Neighborhood Descent) [23, 31] rely almost entirely on the randomization of the local search to escape from local optima. With respect to this issue, probabilistic strategies such as GRASP and VNS may be considered as complementary and potentially capable of leading to effective hybrid methods. A first attempt in this direction was done by Martins et al. [30]. The construction phase of their hybrid heuristic for the Steiner problem in graphs follows the greedy randomized strategy of GRASP, while the local search phase makes use of two different neighborhood structures as a VND strategy. Their heuristic was later improved by Ribeiro, Uchoa, and Werneck [39], one of the key components of the new algorithm being another strategy for the exploration of different neighborhoods. Ribeiro and Souza [38] also combined GRASP with VND in a hybrid heuristic for the degree-constrained minimum spanning tree problem. Canuto, Resende, and Ribeiro [9] used path-relinking in a GRASP for the prize collecting Steiner tree problem.

In this paper, we designed, implemented, and tested several pure and hybrid heuristics:

- a pure GRASP;
- a GRASP that uses path-relinking for intensification;
- a pure VNS;
- a VNS that uses path-relinking for intensification;
- a GRASP that uses VNS to implement the local search phase; and
- a GRASP that uses VNS to implement the local search phase and path-relinking for intensification.

In the algorithms described in the next subsections, we combined the main characteristics of some of the state-of-the-art heuristics, in an attempt to take advantage of their best properties in terms of computation time and solution quality.

2.1 A pure GRASP

GRASP is a randomized multistart iterative method proposed in Feo and Resende [12, 13]. For a comprehensive study of GRASP strategies and variants, the reader is referred to the survey chapter by Resende and Ribeiro [37], as well as to the annotated bibliography of Festa and Resende [14] for a survey of applications. Generally speaking, GRASP is a randomized heuristic having two phases: a construction phase and a local search phase. The construction phase adds one element at a time to a set that ends up with a representation of a feasible solution. At each iteration, an element is

```
procedure GRASP(MaxIterations)
1     for i = 1,...,MaxIterations do
2           Build a greedy randomized solution x;
3           x ← LocalSearch(x);
4           if i = 1 then x* ← x;
5           else if w(x) > w(x*) then x* ← x;
6     end;
7     return (x*);
end GRASP;
```

Figure 1: Pseudo-code of a generic GRASP.

randomly selected from a *restricted candidate list*, whose elements are among the best ordered, according to some greedy function. Once a feasible solution is obtained, the local search procedure attempts to improve it by producing a locally optimal solution with respect to some neighborhood structure. The construction and the local search phases are repeatedly applied. The best solution found is returned as an approximation of the optimal. Figure 1 depicts the pseudo-code of a generic GRASP heuristic.

The construction phase makes use of an adaptive greedy function, a construction mechanism for the restricted candidate list, and a probabilistic selection criterion. The greedy function takes into account the contribution to the objective function achieved by selecting a particular element. In the case of the MAX-CUT problem, it is intuitive to relate the greedy function to the sum of the weights of the edges in each cut. More formally, let $(S,\bar{S})$ be a cut. Then, for each vertex $v \notin S \cup \bar{S}$ we define $\sigma_S(v) = \sum_{u \in S} w_{vu}$ and $\sigma_{\bar{S}}(v) = \sum_{u \in \bar{S}} w_{vu}$. The greedy function, $g(v) = \max\{\sigma_S(v), \sigma_{\bar{S}}(v)\}$, measures how much additional weight will result from the assignment of vertex v to S or $\bar{S}$. The greedy choice consists in selecting the vertex v with the highest greedy function value. If $\sigma_S(v) > \sigma_{\bar{S}}(v)$, then v is placed in $\bar{S}$; otherwise it is placed in S. To define the construction mechanism for the restricted candidate list, let

$$w_{min} = \min\{\min_{v \in V'} \sigma_S(v),\ \min_{v \in V'} \sigma_{\bar{S}}(v)\}$$

and

$$\begin{aligned} w^{max} &= \max\{\max_{v \in V'} \sigma_S(v),\ \max_{v \in V'} \sigma_{\bar{S}}(v)\} \\ &= \max_{v \in V'}\{g(v)\}, \end{aligned}$$

where $V' = V \setminus \{S \cup \bar{S}\}$ is the set of vertices which are not yet assigned to either subset S or subset $\bar{S}$. Denoting by $\mu = w_{min} + \alpha \cdot (w^{max} - w_{min})$ the cut-off value, where α is a parameter such that $0 \leq \alpha \leq 1$, the restricted candidate list is made up by all vertices whose value of the greedy function is greater than or equal to μ. A vertex is randomly selected from the restricted candidate list.

The local search phase is based on the following neighborhood structure. Let $(S,\bar{S})$ be the current solution. To each vertex $v \in V$ we associate either the neighbor $(S \setminus$

```
procedure LocalSearch(x = {S, S̄})
1     change ← .TRUE.
2     while change do;
3           change ← .FALSE.
4           for v = 1, ..., |V| while .NOT.change circularly do
5                 if v ∈ S and δ(v) = σ_S(v) − σ_S̄(v) > 0
6                 then do S ← S \ {v}; S̄ ← S̄ ∪ {v}; change ← .TRUE. end;
7                 if v ∈ S̄ and δ(v) = σ_S̄(v) − σ_S(v) > 0
8                 then do S̄ ← S̄ \ {v}; S ← S ∪ {v}; change ← .TRUE. end;
9           end;
10    end;
11    return (x = {S, S̄});
end LocalSearch;
```

Figure 2: Pseudo-code of the local search phase.

$\{v\}, \bar{S} \cup \{v\}$) if $v \in S$, or the neighbor $(S \cup \{v\}, \bar{S} \setminus \{v\})$ otherwise. The value

$$\delta(v) = \begin{cases} \sigma_S(v) - \sigma_{\bar{S}}(v), & \text{if } v \in S, \\ \sigma_{\bar{S}}(v) - \sigma_S(v), & \text{if } v \in \bar{S}, \end{cases}$$

represents the change in the objective function associated with moving vertex v from one subset of the cut to the other. All possible moves are investigated. The current solution is replaced by its best improving neighbor. The search stops after all possible moves have been evaluated and no improving neighbor was found. The pseudo-code of the local search procedure is given in Figure 2.

2.2 Hybrid GRASP with path-relinking

Path-relinking is an enhancement to the basic GRASP procedure, leading to significant improvements in solution quality. Path-relinking was originally proposed by Glover [18] as an intensification strategy exploring trajectories connecting elite solutions obtained by tabu search or scatter search [19, 20, 21]. Starting from one or more elite solutions, paths in the solution space leading towards other guiding elite solutions are generated and explored in the search for better solutions. This is accomplished by selecting moves that introduce attributes contained in the guiding solutions. Successful applications of path-relinking combined with GRASP are described in [1, 2, 9, 28, 36, 39]. Implementation strategies are described and investigated in detail by Resende and Ribeiro [37].

We now briefly describe the integration of path-relinking into the pure GRASP algorithm described in Subsection 2.1. In this context, path-relinking is applied to pairs (x, z) of solutions, where x is the locally optimal solution obtained by local search (*initial solution*) and z (*guiding solution*) is randomly chosen from a pool with a limited

procedure PR(x,*EliteSet*)
1 Let $(S,\bar{S})$ be the partition defined by x;
2 Randomly select a solution $z \in EliteSet$;
3 Compute the symmetric difference $\Delta(x,z) = \{i = 1,\ldots,|V| : x_i \neq z_i\}$;
4 $w^* \leftarrow \max\{w(x), w(z)\}$;
5 $\bar{x}, y \leftarrow x$;
6 **while** $|\Delta(x,z)| \geq 2$ **do**
7 $i^* = \text{argmax}\{\sigma_S(i) - \sigma_{\bar{S}}(i), \forall i \in \bar{S} \cap \Delta(x,z)$; $\sigma_{\bar{S}}(i) - \sigma_S(i), \forall i \in S \cap \Delta(x,z)\}$;
8 Place vertex i^* in the other partition and set $y_{i^*} \leftarrow 1 - y_{i^*}$;
9 **if** $w(y) > w^*$ **then do** $w^* \leftarrow w(y)$; $\bar{x} \leftarrow y$ **end**;
10 $\Delta(x,z) \leftarrow \Delta(x,z) \setminus \{i^*\}$;
11 **end**;
12 **return** $(\bar{x})$;
end PR;

Figure 3: Pseudo-code of the path-relinking heuristic.

number `MaxElite` of high quality solutions found along the search. The pseudo-code for the path-relinking procedure is shown in Figure 3.

Each solution $y = (S,\bar{S})$ is represented by its characteristic vector y, such that $y_i = 1$ if vertex $i \in S$; $y_i = 0$ otherwise. The path-relinking procedure starts by computing the set $\Delta(x,z) = \{i = 1,\ldots,n : x_i \neq z_i\}$ of variables with different values in the initial and guiding solutions. Each iteration of this procedure has two steps. In the first step, we evaluate the incremental cost $\delta(i)$ resulting from changing the subset of the partition in which vertex i is currently placed, for each $i \in \Delta(x,z)$ (see the description of the local search procedure in Section 2.1). In the second step, the vertex

$$i^* = \text{argmax}\{\sigma_S(i) - \sigma_{\bar{S}}(i), \forall i \in \bar{S} \cap \Delta(x,z)\ ;\ \sigma_{\bar{S}}(i) - \sigma_S(i), \forall i \in S \cap \Delta(x,z)\}$$

with the largest incremental cost is selected, the value of variable y_{i^*} is flipped, we set $\Delta(x,z) \leftarrow \Delta(x,z) \setminus \{i^*\}$, and a new iteration resumes. The relinking procedure stops when the guiding solution is attained. The best solution $\bar{x}$ found along this trajectory is returned.

The pool of elite solutions is originally empty. The best solution $\bar{x}$ found along the relinking trajectory is considered as a candidate to be inserted into this pool. If the pool already has `MaxElite` solutions and the candidate is better than the best elite solution, then $\bar{x}$ replaces the worst elite solution. If the candidate is better than the worst elite solution, but not better than the best, it replaces the worst if it is sufficiently different (see Section 3) from all elite solutions. If the pool is not full, the candidate is simply inserted. Figure 4 depicts the pseudo-code of the proposed GRASP with path-relinking hybrid algorithm.

```
procedure GRASP+PR(MaxIterations)
1     for i = 1,...,MaxIterations do
2           Construct a greedy randomized solution x;
3           x ← LocalSearch(x);
4           if i = 1 then do EliteSet ← {x}; x* ← x;
5           else do
6                 x̄ ← PR(x,EliteSet);
7                 Update EliteSet with x̄;
8                 if w(x̄) > w(x*) then x* ← x̄;
9           end;
10    end;
11    return (x*);
end GRASP+PR;
```

Figure 4: Pseudo-code of the hybrid GRASP with path-relinking.

2.3 A pure VNS

The variable neighborhood search (VNS) metaheuristic, proposed by Hansen and Mladenović [23], is based on the exploration of a dynamic neighborhood model. Contrary to other metaheuristics based on local search methods, VNS allows changes of the neighborhood structure along the search.

VNS explores increasingly distant neighborhoods of the current best found solution x. Each step has three major phases: neighbor generation, local search, and jump. Let N_k, $k = 1,\ldots,k_{max}$ be a set of pre-defined neighborhood structures and let $N_k(x)$ be the set of solutions in the kth-order neighborhood of a solution x. In the first phase, a neighbor $x' \in N_k(x)$ of the current solution is applied. Next, a solution x'' is obtained by applying local search to x'. Finally, the current solution jumps from x to x'' in case the latter improved the former. Otherwise, the order of the neighborhood is increased by one and the above steps are repeated until some stopping condition is satisfied. The pseudo-code of a typical VNS procedure is illustrated in Figure 5.

In the case of the MAX-CUT problem, the kth-order neighborhood is defined by all solutions that can be derived from the current one by selecting k vertices and transfering each of them from one subset of the partition to the other. The same local search strategy used within the pure GRASP algorithm described in Section 2.1 is used in the VNS heuristic.

2.4 Hybrid VNS with path-relinking

As is the case for GRASP, VNS also can be hybridized with path-relinking. At the end of each major VNS cycle, an intensification phase using path-relinking is carried out. Figure 6 shows the pseudo-code for this hybrid heuristic.

```
procedure VNS(MaxIterations, k_max)
1     for i = 1,... ,MaxIterations do
2           k ← 1;
3           Generate a starting solution x at random;
4           while k ≤ k_max do
5                 Generate x' ∈ N_k(x) at random;
6                 x'' ← LocalSearch(x');
7                 if w(x'') > w(x)
8                 then do x ← x''; k ← 1;
9                 else k ← k + 1;
10                end
11          end;
12    end;
13    x* ← x;
14    return (x*);
end VNS;
```

Figure 5: Pseudo-code of a generic VNS heuristic.

2.5 Hybrid GRASP with VNS

This hybrid procedure is simply obtained by replacing the local search phase of the GRASP procedure described in Section 2.1 (line 4 of the pseudo-code in Figure 1) by the VNS procedure presented in Section 2.3. To speed up the search, a smaller value of k_{max} is used in the VNS. The resulting pseudo-code is depicted in Figure 7.

2.6 Hybrid GRASP with VNS and path-relinking

Finally, path-relinking intensification can be added to the GRASP with VNS, resulting in the hybrid GRASP with VNS and path-relinking heuristic, whose pseudo-code is shown in Figure 8.

3 EXPERIMENTAL RESULTS

In this section, we describe computational experience with the heuristics proposed in this paper. We describe the computer environment, discuss implementation details, describe the instances, and report on the experimental evaluation of the different algorithms.

The computational experiments were performed on an SGI Challenge computer, with 28 196-Mhz MIPS R10000 processors and 7.6 Gb of memory. All runs were done using a single processor. Our codes were written in Fortran 77 and compiled with the SGI MIPSpro F77 compiler using flags `-O3 -r4 -64`. The rank-2 relaxation heuristic `circut` was compiled with the SGI MIPSpro 7 F90 compiler using flags `-O3 -r4 -64`.

```
procedure VNS+PR(MaxIterations, k_max)
1     for i = 1,... ,MaxIterations do
2           k ← 1;
3           Generate a starting solution x at random;
4           while k ≤ k_max do
5                 Generate x' ∈ N_k(x) at random;
6                 x'' ← LocalSearch(x');
7                 if w(x'') > w(x)
8                 then do x ← x''; k ← 1;
9                 else k ← k + 1;
10                end;
11          end;
12          if i = 1
13          then do EliteSet ← {x}; x* ← x;
14          else do
15                x̄ ← PR(x,EliteSet);
16                Update EliteSet with x̄;
17                if w(x̄) > w(x*) then x* ← x̄;
18          end;
19    end;
20    return (x*);
end VNS+PR;
```

Figure 6: Pseudo-code of VNS with path-relinking.

```
procedure GRASP+VNS(MaxIterations, k_max)
1     for i = 1,...,MaxIterations do
2           k ← 1;
3           Build a greedy randomized solution x;
4           while k ≤ k_max do
5                 Generate x' ∈ N_k(x) at random;
6                 x'' ← LocalSearch(x');
7                 if w(x'') > w(x)
8                 then do x ← x''; k ← 1;
9                 else k ← k + 1;
10                end;
11          end;
12          if i = 1 then x* ← x;
13          else if w(x) > w(x*) then x* ← x;
14    end;
15    return (x*);
end GRASP+VNS;
```

Figure 7: Pseudo-code of the hybrid GRASP with VNS.

Processing times were measured with the system function `etime`. The portable random number generator of Schrage [40] was used.

Our initial objective was to compare our heuristics with the randomized algorithm of Goemans and Williamson [22], to show that the solutions produced by our randomized heuristics are of much better quality than theirs and can be found in a fraction of the time taken by their algorithm. Recently, however, Burer, Monteiro, and Zhang [8] showed that `circut`, a Fortran 90 implementation of their rank-2 relaxation heuristic for MAX-CUT, produces higher quality approximate solutions in practice than the randomized algorithm of Goemans and Williamson. In addition, running times were shown to be small. For this reason, in this section we compare our heuristics directly with `circut`. We compiled version 0.612 of `circut` on our computer and used it to solve all but one of the test problems used to test our heuristics. We set the `circut` parameters to their default values with the exception of $(N,M) = (50,10)$, the most intensive parameter settings used in [8].

We implemented the following six heuristics described in Section 2:

1. `g`: A pure GRASP, where `MaxIterations` independent GRASP iterations are executed. Each iteration uses the restricted candidate list parameter α selected from the uniform distribution interval $[0,1]$. During a GRASP iteration the value of α does not change.

2. `gpr`: The pure GRASP `g` with forward path-relinking (path-relinking from the local search solution to a randomly chosen elite solution, see [37]) executed after each GRASP local search phase.

```
procedure GRASP+VNS+PR(MaxIterations, k_max)
1     for i = 1,...,MaxIterations do
2           k ← 1;
3           Build a greedy randomized solution x;
4           while k ≤ k_max do
5                 Generate x' ∈ N_k(x) at random;
6                 x'' ← LocalSearch(x');
7                 if w(x'') > w(x)
8                 then do x ← x''; k ← 1;
9                 else k ← k + 1;
10                end;
11          end;
12          if i = 1
13          then do EliteSet ← {x}; x* ← x;
14          else do
15                x̄ ← PR(x,EliteSet);
16                Update EliteSet with x̄;
17                if w(x̄) > w(x*) then x* ← x̄;
18          end;
19    end;
20    return (x*);
end GRASP+VNS+PR;
```

Figure 8: Pseudo-code of GRASP with VNS and path-relinking.

3. `vns`: A pure VNS with `MaxIterations` cycles, each starting with a randomly constructed initial solution, and maximum neighborhood parameter $k_{max} = 100$.

4. `vnspr`: The variable neighborhood search `vns` with forward path-relinking (path-relinking from a locally optimum VNS solution to a randomly chosen elite solution) done after each VNS cycle.

5. `gvns`: The pure GRASP `g` using VNS (with $k_{max} = 15$) as the local search phase.

6. `gvnspr`: `gvns` with forward path-relinking (path-relinking from a locally optimum VNS solution to a randomly chosen elite solution) done after the VNS local search.

Path-relinking is performed within `gpr`, `vnspr`, and `gvnspr`. The maximum size of the elite set was set to 30. Recall that the characteristic vector x of a solution $\{S, \bar{S}\}$ is such that $x_i = 1$ if vertex $i \in S$ and $x_i = 0$ if vertex $i \in \bar{S}$. For inclusion in the elite set, we use a strategy suggested by Fleurent and Glover [15]. In the case that the candidate solution is better than the worst elite set solution but not better than the best, the characteristic vector of the candidate $\bar{x}$ is compared to the characteristic vectors of all elite set solutions. If the candidate differs from all elite solutions by more than 1%, it replaces the worst solution in the elite set.

The experiments consisted of three parts.

In the first part, we tested `circut` and the six randomized heuristics on test problems `G1`, `G2`, `G3`, `G14`, `G15`, `G16`, `G22`, `G23`, `G24`, `G35`, `G36`, `G37`, `G43`, `G44`, `G45`, `G48`, `G49`, and `G50`. These test problems were created by Helmberg and Rendl [24] using a graph generator written by Rinaldi and were used by Burer and Monteiro [7], Benson et al. [5], and Burer, Monteiro, and Zhang [8] for testing their algorithms. They consist of toroidal, planar, and randomly generated graphs of varying sparsity and sizes. These graphs vary in size from 800 to 3000 nodes and in density from 0.17% to 6.12%.

We first ran the randomized heuristics `g`, `gvns`, and `vns` on the Helmberg and Rendl instances using the random number generator seed 270001 for a single iteration. Our objective was to show that the value guaranteed to be achieved by the randomized algorithm of Goemans and Williamson can be easily achieved by our randomized heuristics. Since only one iteration of each algorithm was done, path-relinking was not used. The weights of optimal cuts for these instances are not known. Therefore, we compare the solutions found by our randomized heuristics with the value 0.87856 of the SDP upper bound (which is at least as large as the value guaranteed to be achieved by the randomized algorithm of Goemans and Williamson). Table 1 summarizes these results. We make the following observations about the results in Table 1:

- The pure GRASP (`g`), GRASP with VNS local search (`gvns`), as well as pure VNS (`vns`) found, in their first iteration, a cut with weight at least as large as 0.87856 of the SDP upper bound on all the 18 instances.

- As expected, the processing time increased when going from `g` to `gvns` to `vns`. Pure GRASP times varied from less than 0.5 second to less than 7 seconds. Pure VNS times went from a little over 10 seconds to over 200 seconds.

Table 1: Experimental results for Helmberg and Rendl [24] instances. Total solution times and cut values for 1 iteration of heuristics `g`, `gvns`, and `vns`. Times are in seconds on an SGI Challenge computer (196Mhz R10000 processor). All cut values are larger than 0.87856 of the SDP bound.

Problem			Randomized heuristics									SDP	.879 SDP
			`g`			`gvns`			`vns`				
Name	$\|V\|$	density	time	cut	cut/bound	time	cut	cut/bound	time	cut	cut/bound	bound	bound
G1	800	6.12%	2.10	11420	0.95	6.09	11475	0.95	40.95	11549	0.96	12078	10612
G2			1.98	11488	0.95	3.30	11499	0.95	37.32	11575	0.96	12084	10617
G3			2.02	11419	0.95	4.72	11507	0.95	16.98	11577	0.96	12077	10611
G14	800	1.58%	0.50	3011	0.94	1.81	3009	0.94	12.89	3040	0.95	3187	2800
G15			0.46	2978	0.94	3.57	3008	0.95	18.09	3017	0.95	3169	2785
G16			0.46	2970	0.94	1.78	2983	0.94	10.30	3017	0.95	3172	2787
G22	2000	1.05%	6.29	13027	0.92	43.00	13156	0.93	56.98	13087	0.93	14123	12408
G23			6.29	13121	0.93	41.98	13181	0.93	141.23	13190	0.93	14129	12414
G24			6.55	13059	0.92	46.25	13097	0.93	192.81	13209	0.93	14131	12415
G35	2000	0.64%	3.83	7539	0.94	13.23	7564	0.95	142.54	7593	0.95	8000	7029
G36			3.83	7530	0.94	22.12	7556	0.94	186.05	7584	0.95	7996	7025
G37			3.36	7511	0.94	17.32	7576	0.95	205.48	7598	0.95	8009	7037
G43	1000	2.10%	1.32	6537	0.93	6.03	6583	0.94	36.78	6599	0.94	7027	6174
G44			1.35	6522	0.93	5.19	6559	0.93	40.55	6559	0.93	7022	6170
G45			1.27	6524	0.93	6.54	6553	0.93	24.30	6555	0.93	7020	6168
G48	3000	0.17%	3.61	5902	0.98	11.33	6000	1.00	49.98	6000	1.00	6000	5272
G49			2.48	5924	0.99	7.90	5932	0.99	52.48	5874	0.98	6000	5272
G50			3.94	5812	0.97	19.31	5838	0.97	75.58	5820	0.97	5988	5261

Table 2: Experimental results for Helmberg and Rendl [24] instances. Best solution found in 1000 iterations for `g`, `gpr`, `vns`, `vnspr`, `gvns`, and `gvnspr`. The rank-2 relaxation heuristic `circut` of Burer, Monteiro, and Zhang [8] uses parameters $(N,M) = (50,10)$. SDP UB is the best known semidefinite programming upper bound. For each of the seven heuristics, as well as the SDP bound, the last two rows of this table list the sum of the cuts/bounds over the 24 instances and those sums normalized by the total SDP upper bounds.

Problem			Cut values							
Name	$\|V\|$	density	circut	g	gpr	gvns	gvnspr	vns	vnspr	SDP UB
G1	800	6.12%	11624	11540	11563	11589	11589	11621	11621	12078
G2			11617	11567	11567	11598	11598	11615	11615	12084
G3			11622	11551	11585	11596	11596	11622	11622	12077
G11	800	0.63%	560	552	564	560	564	560	564	627
G12			552	546	552	550	556	554	556	621
G13			574	572	580	576	578	580	580	645
G14	800	1.58%	3058	3027	3041	3044	3044	3055	3055	3187
G15			3049	3013	3034	3031	3031	3043	3043	3169
G16			3045	3013	3028	3031	3031	3043	3043	3172
G22	2000	1.05%	13346	13185	13203	13246	13246	13295	13295	14123
G23			13317	13203	13222	13258	13260	13290	13290	14129
G24			13314	13165	13242	13255	13255	13276	13276	14131
G32	2000	0.25%	1390	1370	1392	1382	1394	1386	1396	1560
G33			1360	1348	1362	1356	1368	1362	1376	1537
G34			1368	1348	1364	1360	1368	1368	1372	1541
G35	2000	0.64%	7670	7567	7588	7605	7605	7635	7635	8000
G36			7660	7555	7581	7604	7604	7632	7632	7996
G37			7666	7576	7602	7601	7608	7643	7643	8009
G43	1000	2.10%	6656	6592	6621	6622	6622	6659	6659	7027
G44			6643	6587	6618	6634	6634	6642	6642	7022
G45			6652	6598	6620	6629	6629	6646	6646	7020
G48	3000	0.17%	6000	6000	6000	6000	6000	6000	6000	6000
G49			6000	6000	6000	6000	6000	6000	6000	6000
G50			5880	5862	5880	5854	5880	5868	5880	5988
		Sum	150623	149337	149809	149981	150060	150395	150441	157743
		% of SDP UB	95.49	94.67	94.97	95.08	95.13	95.34	95.37	100.00

Table 3: Experimental results for Helmberg and Rendl [24] instances. Total solution times for the rank-2 relaxation heuristic `circut` of Burer, Monteiro, and Zhang [8] and 1000 iterations of heuristics `g`, `gpr`, `vns`, `vnspr`, `gvns`, and `gvnspr`. The last column lists time when heuristic `vnspr` made its last solution improvement. For each of the seven heuristics, as well as the time to best for heuristic `vnspr`, the last two rows of this table list the sum of the processing times over the 24 instances and those sums normalized by the time taken by the heuristic `circut`. Times are in seconds on an SGI Challenge computer (196Mhz R10000 processor).

			Processing time (seconds)							
Problem			Total							To best
Name	$\|V\|$	density	circut	g	gpr	gvns	gvnspr	vns	vnspr	vnspr
G1	800	6.12%	352	2111	2111	4647	4684	22713	22732	2536
G2			283	2037	2067	4625	4570	22689	22719	21504
G3			330	2053	2048	4566	4567	23878	23890	11183
G11	800	0.63%	74	276	285	1212	1222	10077	10084	686
G12			58	275	284	1174	1184	10845	10852	1572
G13			62	278	287	1175	1185	10469	10479	3652
G14	800	1.58%	128	478	489	2323	2337	16742	16734	13474
G15			155	478	488	2485	2495	17175	17184	14315
G16			142	478	488	2360	2369	16539	16562	16014
G22	2000	1.05%	493	6667	6724	32114	32175	197689	197654	44384
G23			457	6795	6749	31238	31065	193741	193707	27662
G24			521	6760	6697	31006	31143	195766	195749	152171
G32	2000	0.25%	221	1962	2017	8030	8079	82331	82345	12891
G33			198	1979	2036	7946	7995	76250	76282	72782
G34			237	1967	2113	7999	7954	79363	79406	52380
G35	2000	0.64%	440	3671	3654	19495	19573	167056	167221	96028
G36			400	3692	3646	20609	20701	167102	167203	63522
G37			382	3670	3631	19910	20125	170633	170786	47564
G43	1000	2.10%	213	1361	1379	5637	5584	35366	35324	3152
G44			192	1377	1377	5628	5645	34567	34519	14238
G45			210	1369	1387	5670	5683	34202	34179	10822
G48	3000	0.17%	119	5723	5881	15318	15495	64538	64713	50
G49			134	6165	6273	14447	14497	64596	64749	711
G50			231	4967	5095	16215	16217	146965	147132	846
		Sum	6033	66589	67206	265829	266544	1861292	1862205	684139
		Sum w.r.t. circut	1.00	11.04	11.14	44.06	44.18	308.51	308.67	113.40

The heuristic `circut` and the six randomized heuristics were run a single time on the Helmberg and Rendl test instances. Instances `G11`, `G12`, `G13`, `G32`, `G33`, and `G34`, which have negative weights, were added to the the set of test problems. For each instance, we ran each of our heuristics a total of 1000 iterations, i.e. with `MaxIterations` = 1000. The random number generator seed 270001 was used on all runs. For each instance, Table 2 shows the cut weights found by `circut` and each of our six randomized heuristics, as well as the best known semidefinite programming upper bound (SDP UB). Table 3 lists times (in seconds) to run `circut` using parameters $(N,M) = (50,10)$ and to run 1000 iterations with each randomized heuristic.

On this class of problems, we make the following observations:

- All heuristics were, on average, between 4.5% and 5.4% off from the semidefinite programming upper bound.
- With only two exceptions, all heuristics found cuts greater than 0.87856 of the SDP upper bound. The two exceptions occured with the pure GRASP (`g`) on instances `G33` and `G34`.
- For the randomized heuristics `g`, `gvns`, and `vns`, the incorporation of path-relinking was beneficial, improving some of the solutions, with little additional computational burden.
- At the expense of increased running times, the use of VNS in the local search phase of GRASP was beneficial. Likewise, at the expense of increased running times, using a pure VNS strategy with larger neighborhoods improved upon GRASP with VNS (with a smaller neighborhood).
- Among the randomized heuristics, the variable neighborhood search with path-relinking (`vnspr`) found the best cuts. Heuristic `circut` found slightly better solutions than the randomized heuristic `vnspr` on 13 of the 24 instances. On seven of the 24 instances, `vnspr` found slightly better cuts, while on the remaining four instances, cuts of the same weight were found by `circut` and `vnspr`. Overall, the quality of the solutions found by `circut` and `vnspr` differed by less than 0.12%.
- In terms of solution quality, `circut` and `vnspr` seemed to be sensitive to problem characteristics. For problems with $|V| = 800$, `circut` found better solutions for the densest classes (with densities 1.58% and 6.12%), while `vnspr` found better solutions for the sparsest class (with density 0.63%). Likewise, for problems with $|V| = 2000$, `circut` found better solutions for the densest classes (with densities 0.64% and 1.05%), while `vnspr` found better solutions for the sparsest class (with density 0.25%). For the three problems with $|V| = 1000$, all with density 2.1%, `circut` found better solutions on two instances, while `vnspr` found the better solution on the other instance. For the largest and sparsest instances ($|V| = 3000$ with density 0.17%) both algorithms found equally good solutions. The solutions found by both heuristics for `G48` and `G49` were optimal.
- Running times for 1000 iterations of the randomized heuristics went from a factor of 11 with respect to the running time of `circut` to over a factor of 300. Even

when one considers the time to best, vnspr is still over two orders of magnitude slower than circut.

Since the running times per iteration of our six randomized heuristics vary substantially, we plot in Figure 9 the empirical distributions of the random variable *time-to-target-solution-value* considering instances G11, G12, and G13, using the target values those found by the pure GRASP in the 1000 iteration runs, i.e. 552, 546, and 572, respectively. We performed 200 independent runs of each heuristic using random number generator seeds 270001, 270002, ..., and 270200 and recorded the time taken to find a solution at least as good as the target solution. As in [3], to plot the empirical distribution we associate with the i-th sorted running time (t_i) a probability $p_i = (i - \frac{1}{2})/200$, and plot the points $z_i = (t_i, p_i)$, for $i = 1, \ldots, 200$. We make the following observations about the runs shown in Figure 9:

- The pure GRASP (g) is the heuristic that most benefited from path-relinking. Running times for g varied from less than one second to about 2000 seconds, while with GRASP with path-relinking (gpr), in 80%, 90%, and 100% of the 200 runs on G13, G12, and G11, respectively, the target solution was found in less than 10 seconds.

- The second heuristic to most benefit from path-relinking was GRASP with VNS as the local search procedure (gvns). Heuristic gvns found the target solution in less than 10 seconds on 17%, 20%, and 38% of the 200 runs on instances G13, G12, and G11 runs, respectively, while with path-relinking (gvnspr) found the target solution in less than 10 seconds on 50%, 50%, and 87% of the runs on G13, G12, and G11 runs, respectively.

- Though not as much as GRASP and GRASP with VNS local search, pure VNS also benefited slightly from path-relinking.

- Overall, pure GRASP with path-relinking was the fastest heuristic to find suboptimal solutions with cut weights at least as large as the target values.

In the second part of the experiments, we report on instance pm3-8-50, from the benchmark problem set of the 7th DIMACS Implementation Challenge [1]. This instance has $|V| = 512$ nodes and density 1.17%. It was generated by M. Jünger and F. Liers using the Ising model of spin glasses. The best known solution prior to this paper was 456 and the best known upper bound is 461. The SDP upper bound of 527 is far from the optimal. Burer, Monteiro, and Zhang [8] report a solution of 454 using circut with parameters $(N, M) = (8, 100)$. Using variable neighborhood search with path-relinking (vnspr), we were able to improve the best known solution for this instance. We ran the algorithm 60 times, using random number generator seeds 270001, 270002, ..., and 270060, for a maximum of 1000 iterations. In 16 of these 60 runs, vnspr found a solution of weight 456. On the remaining 44 runs, new best known solutions of weight 458 were found. We recorded the time taken to find a solution of weight 458

[1] http://dimacs.rutgers.edu/Challenges/Seventh/

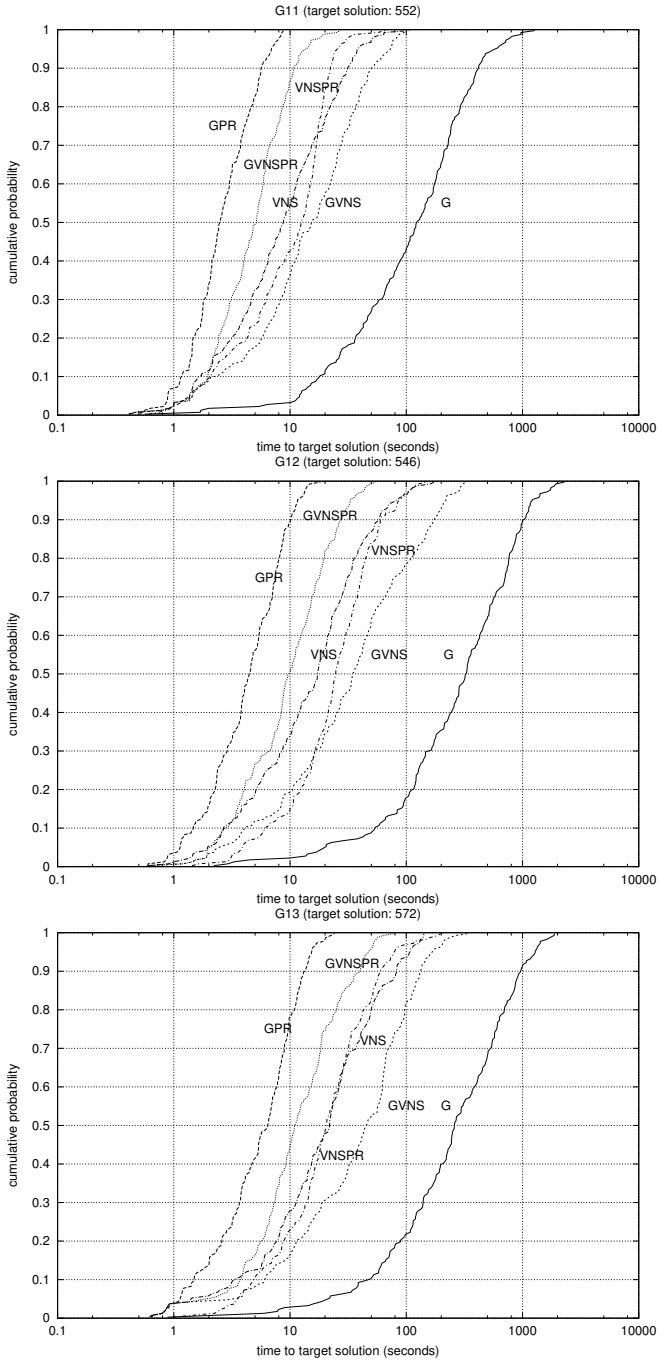

Figure 9: Empirical probability distributions of time to target solution for the six randomized heurisitics on problems `G11`, `G12`, and `G13`. Target solution is the solution found by the pure GRASP on the 1000 iteration run.

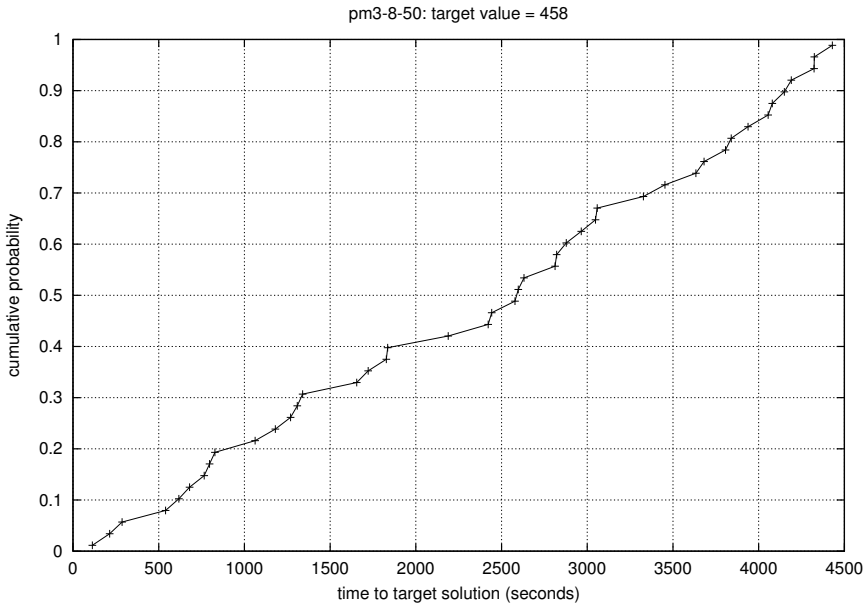

Figure 10: Empirical probability distributions of time to target solution value for `vnspr` on problem `pm3-8-50`. Target solution value is the new best known solution with value 458.

and plotted the empirical distribution of the random variable *time-to-target-solution-value* in Figure 10. To plot the empirical distribution, we associate with the i-th sorted running time (t_i) a probability $p_i = (i - \frac{1}{2})/44$, and plot the points $z_i = (t_i, p_i)$, for $i = 1, \ldots, 44$.

Finally, we considered ten instances from MAX-CUT problems arising in physics, proposed by Burer, Monteiro, and Zhang [8]. These instances correspond to cubic lattice graphs modeling Ising spin glasses. We used ten instances `sg3dl101000`, `sg3dl102000`, ..., `sg3dl1010000` with $|V| = 1000$ and density 0.60% and ten larger and sparser instances `sg3dl141000`, `sg3dl142000`, ..., `sg3dl1410000` with $|V| = 2744$ and density 0.22%. To the best of our knowledge, SDP upper bounds are not known for these instances. We ran `circut` using the most intensive parameter settings used in [8], i.e. $(N, M) = (50, 10)$, and the randomized heuristics `gpr`, `gvnspr`, and `vnspr` for 1000 iterations. Table 4 summarizes the results. For each instance, the table lists the best cut weights found by each heuristic, as well as the total processing times and the time when `vnspr` made its last solution improvement. Sums of cut weights found by each heuristic over the instances are listed. Sums of processing times, as well as those sums normalized by the sum of the processing times of `circut`, are also listed for each heuristic. We make the following observations about these runs:

- The processing time for `circut` was the smallest. For the three randomized heuristics `gpr`, `gvnspr`, and `vnspr`, cut weights increased with processing times.

- On the smaller set of problems, runs for `circut` were 5 to 180 times faster than 1000 iterations of the randomized heuristics. On the larger set of problems, runs for `circut` were 13 to 530 times faster than 1000 iterations of the randomized heuristics. Even when considering the time when `vnspr` stopped making solution improvements, `circut` was still on average about 52 and 279 times faster than `vnspr`, on the smaller and larger set of instances, respectively.
- Overall, `vnspr` found the best cuts, followed by `circut`, `gvnspr`, and `gpr`.
- On the set of ten smaller instances `vnspr` found the best solution for nine instances, while `circut` found the best solution for only two instances. On one instance there was a tie.
- On the set of ten larger instances `vnspr` found the best solution for seven instances, while `circut` found the best solution for four instances. On two instances both heuristics found solutions with the same weight.

Table 4: Experimental results for Ising spin glasses cubic lattice graphs of Burer, Monteiro, and Zhang [8]. Best cuts weights found for `circut` using parameters $(N,M) = (50,10)$, and in 1000 iterations of `gpr`, `gvnspr`, and `vnspr`. Total processing times are given for the four heuristics. The last column lists time when last improvement was made by heuristic `vnspr`. At the bottom of each table, the sum of cut vales found and the sum of processing times over each set of ten instances is given, as is the sum of processing times, normalized by the sum of processing times for heuristic `circut`.

							Processing time (seconds)				
Problem			Cut values				Total				To best
Name	$\|V\|$	density	circut	gpr	gvnspr	vnspr	circut	gpr	gvnspr	vnspr	vnspr
sg3dl101000	1000	0.60%	880	884	884	892	106	564	2269	20409	2417
sg3dl102000			892	896	896	900	116	569	2308	20873	12715
sg3dl103000			882	878	878	884	112	568	2269	20574	5158
sg3dl104000			894	884	890	896	103	570	2300	19786	1906
sg3dl105000			882	868	874	882	106	565	2241	19160	4765
sg3dl106000			886	870	880	880	119	564	2228	17872	590
sg3dl107000			894	890	892	896	115	568	2362	21044	10569
sg3dl108000			874	876	878	880	104	565	2243	19760	6805
sg3dl109000			890	884	896	898	121	569	2282	20930	3098
sg3dl1010000			886	888	886	890	111	567	2271	20028	10717
		Sum	8860	8818	8854	8898	1113	5669	22773	200436	58740
					Time w.r.t. `circut`		1.0	5.1	20.5	180.1	52.8

							Processing time (seconds)				
Problem			Cut values				Total				To best
Name	$\|V\|$	density	circut	gpr	gvnspr	vnspr	circut	gpr	gvnspr	vnspr	vnspr
sg3dl141000	2744	0.22%	2410	2378	2388	2416	382	5009	18775	188390	171255
sg3dl142000			2416	2382	2410	2416	351	4981	19008	187502	129819
sg3dl143000			2408	2390	2394	2406	377	4971	19076	190028	53439
sg3dl144000			2414	2382	2400	2418	356	4968	18985	198809	120112
sg3dl145000			2406	2374	2390	2416	388	5012	18969	196725	190947
sg3dl146000			2412	2390	2406	2420	331	4965	18990	189366	89151
sg3dl147000			2410	2384	2394	2404	381	5028	18814	187902	155018
sg3dl148000			2418	2378	2396	2418	332	4940	18665	194838	3005
sg3dl149000			2388	2362	2372	2384	333	4985	19224	193627	32903
sg3dl1410000			2420	2390	2406	2422	391	4959	19334	196456	64588
		Sum	24102	23810	23956	24120	3622	49818	189840	1923643	1010237
					Time w.r.t. `circut`		1.0	13.8	52.4	531.1	278.9

4 CONCLUDING REMARKS

In this paper we proposed, implemented, and tested six randomized heuristics for the MAX-CUT problem. The heuristics are derived from a greedy randomized adaptive search procedure (GRASP), variable neighborhood search (VNS), and path-relinking (PR).

Without adding much additional computational burden, path-relinking is able to improve the basic GRASP, the basic VNS, as well as the GRASP with VNS local search. GRASP benefits the most from path-relinking, with VNS enjoying the least benefit. GRASP with path-relinking was the fastest among the six randomized heuristics to converge to a solution with the weight at least as good as a specified sub-optimal value. The VNS with path-relinking found the best-quality solutions, but required the longest running times.

The randomized heuristics can quickly find solutions that are competitive with those found by the randomized algorithm of Goemans and Williamson [22], such as implemented by Benson, Ye, and Zhang [6] and, at the expense of additional processing time, can find solutions that come within 5% of the semidefinite programming (SDP) upper bound. For many sparse instances, better cuts than those found by the rank-2 relaxation heuristic of Burer, Monteiro, and Zhang [8] where identified. On problems arising from physics, one of the randomized heuristics (variable neighborhood search with path-relinking) improved the best known lower bound for `pm-3-8-50`, a problem from the 7th DIMACS Implementation Challenge. On Ising spin glasses problems on lattice graphs, the variable neighborhood search with path-relinking found smaller cut weights than `circut` on only five of 20 instances, again at the expense of longer processing times.

As shown in [1], the random variable *time-to-target-solution-value* in GRASP with path-relinking fits a two-parameter exponential distribution. Consequently, one can expect good speed-ups in a parallel implementation of this algorithm.

We experimented with only one neighborhood parameter setting for GRASP with VNS local search, as well as with pure VNS. A more systematic investigation of different parameter settings may yield algorithm with improved performance.

Finally, the algorithms described in this paper can be hybridized with previously described methods. For example, local search could be applied on the cuts produced by the Goemans and Williamson approximation algorithm and path-relinking could be incorporated into `circut`.

References

[1] R.M. Aiex, S. Binato, and M.G.C. Resende. Parallel GRASP with path-relinking for job shop scheduling. *Parallel Computing*, 2002. To appear.

[2] R.M. Aiex, M.G.C. Resende, P.M. Pardalos, and G. Toraldo. GRASP with path-relinking for the three-index assignment problem. Technical report, AT&T Labs-Research, 2000.

[3] R.M. Aiex, M.G.C. Resende, and C.C. Ribeiro. Probability distribution of solution time in GRASP: An experimental investigation. *Journal of Heuristics*, 8:343–373, 2002.

[4] F. Barahona, M. Grötschel, M. Jürgen, and G. Reinelt. An application of combinatorial optimization to statistical optimization and circuit layout design. *Operations Research*, 36:493–513, 1988.

[5] S. Benson, Y. Ye, and X. Zhang. Mixed Linear and Semidefinite Programming for Combinatorial and Quadratic Optimization. *Optimization Methods and Software*, 11/12:515–544, 1999.

[6] S. Benson, Y. Ye, and X. Zhang. Solving large-scale sparse semidefinite programs for combinatorial optimization. *SIAM J. on Optimization*, 10:443–461, 2000.

[7] S. Burer and R.D.C. Monteiro. A projected gradient algorithm for solving the Max-Cut SDP relaxation. *Optimization Methods and Software*, 15:175–200, 2001.

[8] S. Burer, R.D.C. Monteiro, and Y. Zhang. Rank-two relaxation heuristics for MAX-CUT and other binary quadratic programs. *SIAM J. on Optimization*, 12:503–521, 2001.

[9] S.A. Canuto, M.G.C. Resende, and C.C. Ribeiro. Local search with perturbations for the prize-collecting Steiner tree problem in graphs. *Networks*, 38:50–58, 2001.

[10] K.C. Chang and D.-Z. Du. Efficient algorithms for layer assignment problems. *IEEE Trans. on Computer-Aided Design*, CAD-6:67–78, 1987.

[11] R. Chen, Y. Kajitani, and S. Chan. A graph-theoretic via minimization algorithm for two-layer printed circuit boards. *IEEE Trans. on Circuits and Systems*, CAS-30:284–299, 1983.

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

[14] P. Festa and M.G.C. Resende. GRASP: An annotated bibliography. In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys on Metaheuristics*, pages 325–367. Kluwer Academic Publishers, 2002.

[15] C. Fleurent and F. Glover. Improved constructive multistart strategies for the quadratic assignment problem using adaptive memory. *INFORMS Journal on Computing*, 11:198–204, 1999.

[16] K. Fujisawa, M. Fojima, and K. Nakata. Exploiting sparsity in primal-dual interior-point methods for semidefinite programming. *Mathematical Programming*, 79:235–253, 1997.

[17] K. Fujisawa, M. Fukuda, M. Fojima, and K. Nakata. Numerical evaluation of SDPA (Semidefinite Programming Algorithm. In *High performance optimization*, pages 267–301. Kluwer Academic Publishers, 2000.

[18] F. Glover. Tabu search and adaptive memory programing – Advances, applications and challenges. In R.S. Barr, R.V. Helgason, and J.L. Kennington, editors, *Interfaces in Computer Science and Operations Research*, pages 1–75. Kluwer, 1996.

[19] F. Glover. Multi-start and strategic oscillation methods – Principles to exploit adaptive memory. In M. Laguna and J.L. Gonzáles-Velarde, editors, *Computing Tools for Modeling, Optimization and Simulation: Interfaces in Computer Science and Operations Research*, pages 1–24. Kluwer, 2000.

[20] F. Glover and M. Laguna. *Tabu search*. Kluwer Academic Publishers, 1997.

[21] F. Glover, M. Laguna, and R. Martí. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 39:653–684, 2000.

[22] M.X. Goemans and D.P. Williams. Improved approximation algorithms for Max-Cut and Satisfiability Problems using Semidefinite Programming. *J. of the ACM*, 42:1115–1145, 1995.

[23] P. Hansen and N. Mladenović. Developments of variable neighborhood search. In C.C. Ribeiro and P. Hansen, editors, *Essays and Surveys in Metaheuristics*, pages 415–439. Kluwer Academic Publishers, 2002.

[24] C. Helmberg and F. Rendl. A spectral bundle method for semidefinite programming. *SIAM J. on Optimization*, 10:673–696, 2000.

[25] S. Homer and M. Peinado. Two distributed memory parallel approximation algorithms for Max-Cut. *J. of Parallel and Distributed Computing*, 1:48–61, 1997.

[26] S.E. Karisch, F. Rendl, and J. Clausen. Solving graph bisection problems with semidefinite programming. *SIAM J. on Computing*, 12:177–191, 2000.

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

[28] M. Laguna and R. Martí. GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, 11:44–52, 1999.

[29] L. Lovász. On the Shannon capacity of a graph. *IEEE Trans. of Information Theory*, IT-25:1–7, 1979.

[30] S.L. Martins, M.G.C. Resende, C.C. Ribeiro, and P. Pardalos. A parallel GRASP for the Steiner tree problem in graphs using a hybrid local search strategy. *Journal of Global Optimization*, 17:267–283, 2000.

[31] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers and Operations Research*, 24:1097–1100, 1997.

[32] P.M. Pardalos and M.G.C. Resende, editors. *Handbook of Applied Optimization*. Oxford University Press, 2002.

[33] R.Y. Pinter. Optimal layer assignment for interconnect. *J. of VLSI Computat. Syst.*, 1:123–137, 1984.

[34] S. Poljak, F. Rendl, and H. Wolkowicz. A recipe for semidefinite relaxation for 0-1 quadratic programming. *J. of Global Optimization*, 7:51–73, 1995.

[35] S. Poljak and Z. Tuza. The Max-Cut problem: A survey. In W. Cook, L. Lovász, and P. Seymour, editors, *Special Year on Combinatorial Optimization*, DIMACS Series in Discrete Mathematics and Computer Science. American Mathematical Society, 1995.

[36] M.G.C. Resende and C.C. Ribeiro. A GRASP with path-relinking for permanent virtual circuit routing. Technical report, AT&T Labs Research, 2001.

[37] M.G.C. Resende and C.C. Ribeiro. Greedy randomized adaptive search procedures. In F. Glover and G. Kochenberger, editors, *State-of-the-Art Handbook of Metaheuristics*. Kluwer, 2002.

[38] C.C. Ribeiro and M.C. Souza. Variable neighborhood search for the degree constrained minimum spanning tree problem. *Discrete Applied Mathematics*, 118:43–54, 2002.

[39] C.C. Ribeiro, E. Uchoa, and R.F. Werneck. A hybrid GRASP with perturbations for the Steiner problem in graphs. *INFORMS Journal on Computing*, 14:228–246, 2002.

[40] L. Schrage. A more portable Fortran random number generator. *ACM Transactions on Mathematical Software*, 5:132–138, 1979.

[41] N.Z. Shor. Quadratic optimization problems. *Soviet J. of Computer and Systems Science*, 25:1–11, 1987.