Advanced Scatter Search for the Max-Cut Problem

RAFAEL MARTÍ
Departamento de Estadística e Investigación Operativa, Universidad de Valencia, Spain
Rafael.Marti@uv.es

ABRAHAM DUARTE
Departamento de Ciencias de la Computación, Universidad Rey Juan Carlos, Spain.
Abraham.Duarte@urjc.es

MANUEL LAGUNA
Leeds School of Business, University of Colorado at Boulder, USA
laguna@colorado.edu

Abstract

The Max-Cut problem consists of finding a partition of the nodes of a weighted graph into two subsets such that the sum of the weights between both sets is maximized. This is an NP-hard problem that can also be formulated as an integer quadratic program. Several solution methods have been developed since the 1970s and applied to a variety of fields, particularly in engineering and layout design. We propose a heuristic method — based on the scatter search methodology — for finding approximate solutions to this optimization problem. Our solution procedure incorporates some innovative features within the scatter search framework: 1) the solution of the maximum diversity problem to increase diversity in the reference set, 2) a dynamic adjustment of a key parameter within the search, and 3) the adaptive selection of a combination method. We perform extensive computational experiments to first study the effect of changes in critical scatter search elements and then to compare the efficiency of our proposal with previous solution procedures.

Keywords: Max-cut problem, scatter search, metaheuristics, evolutionary algorithms

Version: June 1, 2007

1. Introduction

Consider a graph $G = (V, E)$ with vertex set $V = \{1,\ldots n\}$ and edge set E. Let w_{ij} be the weight associated with edge $(i, j) \in E$. A cut (S, S') is a partition of V into two sets S, $S' = V\text{-}S$ and its value $cut(S, S')$ is given by the expression:

$$cut(S, S') = \sum_{\substack{u \in S \\ v \in S'}} w_{uv} \ .$$

The Max-Cut problem consists of finding a cut in G with maximum value. Karp (1972) showed that the Max-Cut is an NP-hard problem. Table 1 summarizes the approaches (in chronological order) that are most relevant to the work described here.

Reference	Procedure	Comments
Sahni and Gonzales (1976)	Greedy heuristic	Simple and fast
Lovász (1979)	Semidefinite programming	Relaxation from the quadratic formulation
Goemans and Williamson (1995)	Randomized method	Performance guarantee of 0.878
Burer et al. (2001)	CirCut	Rank-2 relaxation heuristic
Festa et al. (2002)	GRASP and VNS	Outperforms previous heuristic methods
Krishnan and Mitchell (2006)	Cut and Price	Exact method for mid-size instances

Table 1. Summary of relevant literature.

The Max-Cut is a classical optimization problem with several practical applications (Chang and Du. 1987; Chen et al. 1983) that has received a great deal of attention in the last two decades. Beginning with the simple approach introduced by Sahni and Gonzales (1976) and some variations (Poljak and Tuza, 1982; Haglin and Venkatesen, 1991), the literature contains a large number of solution methods for this problem. Goemans and Williamson (1995) worked with a relaxation based on semidefinite programming and obtained both, an upper bound on the optimal value — which we will refer to as SDP — and a heuristic method with a performance guarantee of 0.878. This heuristic, however, requires of extremely long running times (e.g., instances with $n = 200$ call for 3 hours of CPU time) and thus a parallel version was proposed by Homer and Peinado (1997). More recently, Burer et al. (2001) introduced the CirCut method, also based on a relaxation of the problem. CirCut experiments show that — in terms of solutions quality — it outperforms all previous procedures in shorter computational time. Finally, Festa et al. (2002) developed six different algorithms based on the variable neighborhood search (VNS), GRASP and Path Relinking (PR) methodologies. In their experimentation, the authors show that their VNS algorithm coupled with PR (referred to as VNSPR) obtains high quality solutions, although at the expense of long computational times. We include both CirCut and VNSPR in the computational experience described in Section 6. In this experimentation we target large graph sizes, with up 3000 vertices. Recent works on exact methods for the Max-Cut problem, such as the Cut and Price procedure by Krishnan and Mitchell (2006), are capable of optimally solving medium size instances (i.e., $n \approx 500$).

The motivation for our work is the implementation of a scatter search (SS) procedure that includes some new elements that could become part of the standard methodology. Just as important, it is our goal of providing a procedure for the Max-Cut problem that is capable of producing high quality approximations in a reasonable amount of computer time. Scatter search (Laguna and Martí, 2003) consists of five methods and

their associated strategies (see Figure 1). Three of them, the Diversification Generation, the Improvement and the Combination Methods, are problem dependent and are designed specifically for the problem being solved. Although it is possible to design "generic" procedures, it is more effective to base the design on the specific characteristics of the problem setting. The other two, the Reference Set Update and the Subset Generation Methods are context independent, and standard implementations are available.

1. Diversification generation
2. Improvement
3. Reference set update

while (termination criteria not satisfied)
{
    4. Subset generation
    5. Combination
    6. Improvement
    7. Reference set update
}

Figure 1. Scatter search framework

The first two steps in Figure 1 — diversification generation (described in Section 2) and improvement (described in Section 3) — yield a population *P* of solutions from which the initial reference set (*RefSet*) is constructed. The initial *RefSet* must balance solution quality and diversity and therefore the standard update in step 3 selects the best |*RefSet*|/2 solutions from *P* and then the |*RefSet*|/2 solutions in *P**RefSet* that are most diverse with respect to those solutions already in the reference set. We will describe a new way of performing this step in Section 5.1.

In our implementation, step 4 is done by generating all pairs of reference solutions that have not been combined before. Details about the combination methods tested in this study are presented in Section 4. The reference set update in step 5 collects all the solutions generated in step 4 and all the solutions in the current reference solutions and — from this pool — it chooses the best solutions (in terms of quality) to form the new reference set. This is the so-called static update by quality (see Laguna and Martí 2003).

When no new solutions are admitted to the reference set after step 7 in Figure 1, the SS methodology dictates that the search either terminates or a *RefSet* rebuilding step is performed. The rebuilding step consists of eliminating all but the best reference solution and reinitializing the process from the first step in Figure 1. In our implementation, we have chosen to terminate the SS method after a pre-specified number of *RefSet* rebuilding steps. Section 6 describes our experimentation to test the effectiveness of the proposed methods as well as to compare the SS algorithm with the state-of-the-art methods for the Max-Cut problem.

2. Diversification Generation Method

Festa et al. (2002) proposed the GRASP (Festa and Resende, 2001) construction C_1 for the Max-Cut problem. It uses two greedy functions that take into account the

contribution of the vertices to the objective function. Specifically, for each vertex v they define $\sigma(v)$ and $\sigma'(v)$ as:

$$\sigma(v) = \sum_{u \in S} w_{vu} \qquad \sigma'(v) = \sum_{u \in S'} w_{vu}$$

C_1 starts by randomly selecting a vertex in S and another vertex in S' and then, at each step, it randomly selects a vertex v in the restricted candidate list (RCL). If $\sigma(v) > \sigma'(v)$ then v is placed in S'; otherwise it is placed in S. RCL consists of the unselected vertices $v \in V' = V \setminus \{S \cup S'\}$ such that $\sigma(v) > \mu$ or $\sigma'(v) > \mu$ where

$$w_{\min} = \min\{\min_{v \in V'} \sigma(v), \min_{v \in V'} \sigma'(v)\}$$
$$w^{\max} = \max\{\max_{v \in V'} \sigma(v), \max_{v \in V'} \sigma'(v)\}$$
$$\mu = w_{\min} + \alpha(w^{\max} - w_{\min}) \qquad 0 \le \alpha \le 1$$

The parameter α is randomly selected between 0 and 1 at the beginning of each construction. We illustrate how this method works when applied to the example of Figure 2. The circles represent the vertices (numbered from 1 to 5) and the numbers next to the edges their respective weights.

C_1 starts by assigning a vertex to S and another to S′. Say for instance that vertices 1 and 5 are selected and are assigned as follows: $S = \{5\}$ and $S' = \{1\}$. The method then computes σ and σ' for the remaining vertices:

$$\sigma'(2) = 0,\ \sigma(2) = 5,\ \sigma'(3) = 9,\ \sigma(3) = 14,\ \sigma'(4) = 0, \text{ and } \sigma(4) = 0$$

It calculates $w_{\min} = 0$, $w^{\max} = 14$ and — considering for example $\alpha = 0.5$ — it obtains $\mu = 7$. The resulting restricted candidate list is RCL = {3, 3′}, where 3 means vertex 3 by virtue of its σ value and 3′ means vertex 3 by virtue of its σ' value. A random selection is made from the RCL, say 3 and therefore vertex 3 is assigned to S'. In the next step, the method computes σ and σ' for the unselected vertices, $\sigma'(2) = 0$, $\sigma(2) = 5$, $\sigma'(4) = 7$, $\sigma(4) = 10$, and obtains RCL = {4, 4′}. We assume that 4′ is randomly selected and the vertex is added to S. In the last step, $\sigma'(2) = 0$ and $\sigma(2) = 13$ and RCL = {2} resulting in the assignment of vertex 2 to S'. Thus the solution is $S = \{4, 5\}$, $S' = \{1, 2, 3\}$ with the value $cut(S, S') = 49$.

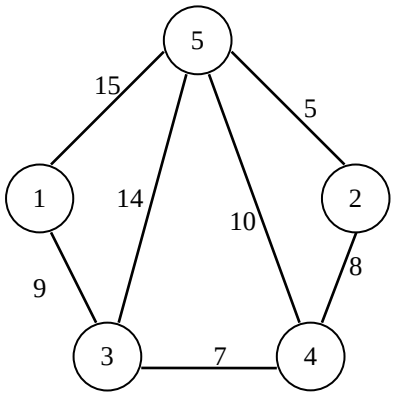

Figure 2. Example of a graph with edge weights

We now propose a new construction method C_2 — also based on the GRASP methodology. It starts by considering that all the vertices are in S' ($S = \emptyset$, $S' = V$, $cut(S, S') = 0$). We define the evaluation of each vertex v as the increase in the value of the cut when v is moved from S' to S.

$$cut(S \cup \{v\}, S' \setminus \{v\}) = cut(S, S') + \Delta cut(v)$$

$$\text{where } \Delta cut(v) = \sigma'(v) - \sigma(v)$$

At each step, C_2 randomly selects a vertex in RCL and moves it from S' to S. In this construction method, RCL is formed by those vertices v in S' with an evaluation $\Delta cut(v) > th$.

$$th = \Delta cut_{\min} + \alpha(\Delta cut^{\max} - \Delta cut_{\min}) \qquad 0 \le \alpha \le 1$$

$$\text{where } \Delta cut^{\max} = \max_{v \in S'} \Delta cut(v),\ \Delta cut_{\min} = \max(0, \min_{v \in S'} \Delta cut(v)).$$

Instead of randomly selecting an α-value at the beginning of each construction, as in C_1, C_2 works with a fixed value of α for all constructions. In the experiments reported in Section 6 we compare the performance of C_2 when employing different values of α.

In the example of Figure 2, a C_2 construction starts with the assignment of all vertices to S'. Then, the method computes σ, σ' and Δcut for all vertices,

$\sigma'(1) = 15 + 9 = 24$	$\sigma(1) = 0$	$\Delta cut(1) = 24$
$\sigma'(2) = 5 + 8 = 13$	$\sigma(2) = 0$	$\Delta cut(2) = 13$
$\sigma'(3) = 9 + 14 + 7 = 30$	$\sigma(3) = 0$	$\Delta cut(3) = 30$
$\sigma'(4) = 7 + 10 + 8 = 25$	$\sigma(4) = 0$	$\Delta cut(4) = 25$
$\sigma'(5) = 15 + 14 + 10 + 5 = 44$	$\sigma(5) = 0$	$\Delta cut(5) = 44$

This results in $\Delta cut_{\min} = 13$, $\Delta cut^{\max} = 44$, and with $\alpha = 0.5$ we obtain $th = 28.5$. Therefore, RCL = {3, 5} and we randomly select vertex 5 and move it from S' to S. In the second step, we first compute:

$\sigma'(1) = 9$	$\sigma(1) = 15$	$\Delta cut(1) = -6$
$\sigma'(2) = 8$	$\sigma(2) = 5$	$\Delta cut(2) = 3$
$\sigma'(3) = 9 + 7 = 16$	$\sigma(3) = 14$	$\Delta cut(3) = 2$
$\sigma'(4) = 7 + 8 = 15$	$\sigma(4) = 10$	$\Delta cut(4) = 5$

Then, $\Delta cut_{\min} = 0$, $\Delta cut^{\max} = 5$ and with $\alpha = 0.5$ we obtain $th = 2.5$. Therefore, RCL = {2, 4} and we randomly select a vertex, for example vertex 4, and move it from S' to S. In the next iteration, $\Delta cut(1) = -6$, $\Delta cut(2) = -13$ and $\Delta cut(3) = -12$, and therefore nothing is selected to be moved to S. The process terminates obtaining $S = \{4, 5\}$, $S' = \{1, 2, 3\}$ and $cut(S, S') = 49$.

To finish this section, we describe a third construction method (C_3) that we developed by employing memory structures instead of randomization. The evaluation of a vertex is modified by the frequency by which it has been selected to belong to *S*, favoring the selection of those vertices with low frequency values. C_3 also starts by considering all the vertices in *S*′. Then, at each step, the procedure selects the vertex with the largest modified evaluation $\Delta cut_f(v)$ and moves it from *S*′ to *S*.

$$\Delta cut_f(v) = \Delta cut(v) - \beta \frac{freq(v)}{MaxFreq} \Delta cut(v)$$

In this expression, *freq*(*v*) is the number of times that vertex *v* has been assigned to *S* in previous constructions. *MaxFreq* is the maximum of *freq*(*v*) for all *v* and β is an input parameter that modifies the behavior of this construction procedure. Performance comparisons among C_1, C_2 and C_3 are presented and discussed in Section 6.

In order to illustrate how C_3 works, we assume that the method has been used to construct several solutions for the example in Figure 2 and that the current frequency counts are: *freq*(1) = 3, *freq*(2) = 1, *freq*(3) = 6, *freq*(4) = 3 and *freq*(5) = 8. Hence, *MaxFreq* is equal to 8 and, with β = 0.75, the following Δcut and Δcut_f values are obtained:

$\Delta cut(1) = 24$	$\Delta cut_f(1) = 24\text{-}0.75(3/8)24 = 17.25$
$\Delta cut(2) = 13$	$\Delta cut_f(2) = 13\text{-}0.75(1/8)13 = 11.78$
$\Delta cut(3) = 30$	$\Delta cut_f(3) = 30\text{-}0.75(6/8)30 = 13.12$
$\Delta cut(4) = 25$	$\Delta cut_f(4) = 25\text{-}0.75(3/8)25 = 17.97$
$\Delta cut(5) = 44$	$\Delta cut_f(5) = 44\text{-}0.75(8/8)44 = 11.00$

These calculations prescribe the selection of vertex 4, since it has the maximum Δcut_f value. The vertex is moved from *S*′ to *S*. The second step results in a maximum Δcut_f of 17.25 associated with vertex 1, which is then moved to *S*. In the next and final step all the vertices in *S*′ have negative Δcut values. The method terminates with the solution $S = \{1, 4\}$, $S' = \{2, 3, 5\}$ and $cut(S,S') = 49$.

3. Improvement Method

Glover and Laguna (1997) introduced the notion of *compound moves*, often called *variable depth methods*, constructed from a series of simpler components. It is well-known that one of the pioneering contributions to this type of moves was Lin and Kernighan (1973). Within the class of variable depth procedures, a special subclass called *ejection chain procedures* has recently proved useful. As described in Glover (1996), "Ejection chain procedures are based on the notion of generating compound sequences of moves, leading from one solution to another, by linked steps in which changes in selected elements cause other elements to be *ejected from* their current state, position or value assignment." In this section we adapt the notion of ejection chains to the Max-Cut problem.

Festa et al. (2002) proposed the local search method LS_1 based on the following neighborhood structure. Given a solution (S, S') they define *move*(v) as moving vertex v from S to S' (if $v \in S$) or from S' to S (if $v \in S'$) and the associated move value as:

$$MoveValue(v) = \begin{cases} \sigma(v) - \sigma'(v) & \text{if } v \in S \\ \sigma'(v) - \sigma(v) & \text{if } v \in S' \end{cases}$$

LS_1 explores, in each iteration, the complete neighborhood; i.e. it examines all the vertices in V to identify the vertex v with the largest move value. The method performs *move*(v) as long as *MoveValue*(v) > 0. After an improving move, the objective function is incremented by *MoveValue*(v). This local search method finishes when no improvement move is found in the neighborhood of the current solution.

In contrast to the *best* strategy implemented in LS_1, we propose a local search method LS_2 based on a *first* strategy. The *best* strategy selects the move with the largest move value among all the moves in the neighborhood. The *first* strategy, on the other hand, scans the list of vertices in search for the first vertex whose movement results in a strictly positive move value.

In the Max-Cut problem, suppose that a vertex $v \in S$ is adjacent to $w \in S'$ and that v is being considered for a move to S'. This possible action triggers the consideration of moving w in the opposite direction and the benefit of the dual move could be calculated. In terms of ejection chains, we may say that the move of v from S to S' caused w to be "ejected" from S' to S (defining a compound move of depth two). Clearly, the benefit of moving a vertex adjacent to w from S to S' could also be evaluated if one would like to consider compound moves of depth three. Longer chains are possible by applying the same logic.

In LS_2, we define *move*2(v) as the ejection chain that starts from moving vertex v. The chain starts by making *move*2(v) = *move(v)*. If this depth-1 move is improving, it is executed and the chain stops. Otherwise, we search for *move(w)* associated with a vertex w adjacent to v allocated in the set where v has been moved. If the compound move of depth two — *move*2(v) = *move(v)*+*move(w)* — is an improving move, the move is executed and the chain stops; otherwise the chain continues until the compound move becomes improving or the depth of the chain reaches the pre-specified limit k. If none of the compounds moves from depth 1 to k examined in *move*2(v) is an improving move, no move is performed and the exploration continues with the next vertex in the list.

A global iteration of LS_2 consists of first ordering the vertices v according to their $\sigma(v)$-$\sigma'(v)$ value (if v is in S, otherwise we use $\sigma'(v)$-$\sigma(v)$) and then scanning them in this order in search for an improving move. In this process, $\sigma(v)$-$\sigma'(v)$ or $\sigma'(v)$-$\sigma(v)$ is not the true value associated with *move*2(v) — unless the chain stops at a depth of one — and it is only used as an indicator of potentially "promising" moves. The list of vertices is reordered only after all the vertices have been examined. LS_2 is a local optimizer and hence it performs only improving moves. The method continues iterating only if in the previous iteration (i.e., the examination of all nodes) at least one improving move has been performed. Otherwise, LS_2 stops.

4. Combination Method

This method uses the subsets generated with the subset generation method (step 4 in Figure 1) to combine the two elements in each subset with the purpose of creating new trial solutions. Given that the combination method is a problem-specific procedure, we present three alternatives for the Max-Cut problem. The next section describes how these three methods are selected within the scatter search algorithm in order to produce new solutions.

We adapt combination method CB_1, proposed in the context of the knapsack problem in Laguna and Martí (2003). The method calculates a score for each variable, based on the objective function value of the two reference solutions being combined. Let x be a binary string of size n representing a solution to our problem, where variable x_i equals 1 if element i is in set S. The score for variable i that corresponds to the combination of reference solutions j and k is calculated with the following formula:

$$score(i) = \frac{cut(j)x_i^j + cut(k)x_i^k}{cut(j) + cut(k)}$$

where $cut(j)$ is the objective function value of solution j and x_i^j is the value of the i^{th} variable in solution j. Then, the trial solution is constructed by using the score as the probability for setting each variable to one, i.e., $P(x_i = 1) = score(i)$. This can be implemented as follows:

$$x_i = \begin{cases} 1 & \text{if } r \le score(i) \\ 0 & \text{if } r > score(i) \end{cases}$$

where r is a uniform random number such that $0 \le r \le 1$. We set $r = 0.5$ as recommended in Laguna and Martí (2003) and obtain two trial solutions from the combination of two reference solutions.

Given two solutions (S,S') and (T,T'), combination method CB_2 first computes the partial solution formed with the intersection (U, U') of both solutions where $U = S \cap T$ and $U' = S' \cap T'$. Then, for each unselected vertex v ($v \in V \setminus \{U \cup U'\}$) the method computes $\sigma(v)$ and $\sigma'(v)$ and places the vertex in the set that produces the best evaluation. That is, if $\sigma(v) > \sigma'(v)$ then v is placed in U'; otherwise it is placed in U. A second solution is obtained as follows. We apply the first step in the same way, computing the partial solution formed with the intersection of both solutions, but then, instead of assigning each vertex to the best set as prescribed by the sigma values, they are assigned randomly. Therefore, CB_2 produces two solutions, one is *quality* oriented and the other is *diversity* oriented.

Festa et al. (2002) proposed a path relinking (PR) algorithm to enhance their GRASP and VNS methods for the Max-Cut problem. PR was first described in Glover and Laguna (1993) and generally operates by starting from an *initiating solution*, selected from a subset of high quality solutions, and generating a path in the neighborhood space that leads toward other solutions called *guiding solutions*. This is accomplished by selecting moves that introduce attributes — into the initiating solution — contained in

the guiding solutions. PR, which can also be considered an extension of the standard combination methods of scatter search, was first adapted in the context of GRASP by Laguna and Martí (1999). Instead of directly producing a new solution when combining two or more original solutions, PR generates paths between and beyond the selected solutions in the neighborhood space. The character of such paths is easily specified by reference to solution attributes that are added, dropped or otherwise modified by the moves executed. Combination method CB_3 consists of a variation of the PR algorithm of Festa et al. (2002). Given two solutions (S,S') and (T,T'), CB_3 first computes the path from (S,S') to the midpoint between (S,S') and (T,T'), and then the path from (T,T') to the midpoint between (T,T') and (S,S'). The final solution in each path is subjected to the improvement method and the two improved solutions are returned as the output of this combination method.

To define the path between solutions (S,S') and (T,T'), we first calculate the set of elements assigned to different sets in both solutions, $Diff = (S \cap T') \cup (S' \cap T)$. In order to prevent symmetries we consider that vertex 1 is always in S (and T). The path from the initiating solution (S,S') to the guiding solution (T,T') is obtained by simply changing, one by one, the vertices in *Diff* from their current assignment to their target assignment. We scan the vertices v in the order given by their $MoveValue(v)$, thus moving first the one with the largest contribution to the objective function. The exploration from (S,S') to (T,T') terminates when we reach the midpoint in the path; i. e. when we have moved $|Diff|/2$ vertices. An attempt to improve the resulting solution is made and the roles of (S,S') and (T,T') are reversed to generate a second solution.

5. Advanced Scatter Search Elements

In this study, we have extended the basic scatter search implementation in three different ways. The first extension consists of a new selection procedure for constructing a reference set from a population of solutions. Traditionally, scatter search implementations have used the criterion of maximizing the minimum distance between the solution under consideration and the solutions already in the reference set. In such a process, diverse solutions are selected one by one from the population P and the distances are updated after each selection. In contrast, we suggest a method of selecting all the diverse solutions at once by way of solving the maximum diversity problem, as described below.

Our second extension consists of a dynamic adjustment of the depth parameter k associated with the ejection chain mechanism. Each solution is represented in such a way that they carry the information related to the particular k value that was used to generate it. In this way, the depth of application of the ejection chain procedure depends on the parameter values that are particular to the solutions being combined (as described below).

The third extension implements a probabilistic selection of the combination methods. The probability of selecting any of the three methods described in the previous section is proportional to the number of high quality solutions that such methods have generated in previous iterations. The next three sections provide the details of our proposed extensions. The implementation framework of our scatter search follows the standard

one developed by Laguna and Martí (2003) — and outlined in Figure 1 — with the addition of the three advanced elements described below.

5.1 Maximum initial diversity in the RefSet

The initial reference set (*RefSet*) in standard scatter search implementations is constructed by performing three steps: (1) generating a population *P* of diverse solutions, (2) selecting the |*RefSet*|/2 best solutions in *P* and adding them to *RefSet*, and (3) adding to *RefSet* the |*RefSet*|/2 most diverse solutions in *P* \ *RefSet*. The selection of solutions in step 2 is made with reference to the objective function values of the solutions in *P*. Thus, the best |*RefSet*|/2 solutions in *P* are added to *RefSet* and deleted from *P*. In standard SS implementation, the third step is performed by selecting one solution at a time. The first solution is the one in *P* that is the most diverse with respect to the solutions currently in *RefSet*. Diversity is typically measured by a function that maximizes the minimum distance between the solution under consideration and the set of solutions — in this case *RefSet* — from which the method wants to move away. Once a diverse solution is selected, the solution is deleted from *P* and added to *RefSet* and the process is repeated |*RefSet*|/2 times.

Instead of the one-by-one selection of diverse solutions, we propose solving the maximum diversity problem (MDP) with the GRASP_C_2 method developed by Duarte and Martí (2007). Since the MDP is a computationally hard problem, we have chosen GRASP_C_2 because it provides a good balance between solution quality and speed, attributes that are important in order to embed it as part of the overall SS framework. The MDP consists of finding, from a given set of elements and corresponding distances between elements, the most diverse subset of a given size. The diversity of the chosen subset is given by the sum of the distances between every pair of elements. The distance between two solutions is given as the number of edges in the cut that are different. To use the MDP within SS, we must recognize that the original set of elements is given by *P* minus the |*RefSet*|/2 best solutions. Also, the most diverse subset that we are asking the MDP to construct is *RefSet*, which is already partially populated with the |*RefSet*|/2 best solutions from *P*. Therefore, we have modified GRASP_C_2 in order to solve the special MDP for which some elements have already been chosen (i.e., the first |*RefSet*|/2 selected due to their quality) and cannot be removed from the solution. Diversity between solutions is measured as the number of edges that are different and GRASP_C_2 is performed for 100 iterations. The most diverse *RefSet* found by GRASP_C_2 is chosen to initiate the scatter search phase that iterates the subset generation, combination, improvement and reference set update methods (steps 4 to 7 in Figure 1).

5.2 Dynamic adjustment of a search parameter

Section 3 introduced an adaptation of ejection chains as a mechanism for creating compound moves within the LS_2 improvement method. The method employs a parameter *k* that limits the length of the chain. Instead of utilizing a fixed value for this parameter throughout the search, we have developed a process to adjust it dynamically as effective values for that parameter are identified. Since solutions generated during the search (either by the diversification or combination method) are subjected to the improvement method (steps 2 and 6 in Figure 1), there is a *k*-value associated with each of them. This value is stored as part of the solution and used during the application of the combination method as follows. Let (*S*,*S*′) and (*T*,*T*′) — with associated k_S and k_T

values — be two reference solutions that are being combined. Also let *cut*(*S*,*S*′) and *cut*(*T*,*T*′) be their respective objective function values. The parameter value used to apply the improvement method to the trial solution that results from combining (*S*,*S*′) and (*T*,*T*′) is:

$$k = \frac{cut(S,S') * k_S + cut(T,T') * k_T}{cut(S,S') + cut(T,T')}$$

Note that this combination of the *k*-values parallels the one used for the variable values in CB_1. Initially, the parameter values are set to a random number between 1 and 5, which we have determined to be effective during preliminary experimentation (see Section 6).

5.3 Adaptive selection of combination method

In Section 4, we described three combination methods. In addition to performing preliminary experimentation to determine which of the competing approaches performs best overall (see Section 6), we introduced a probability-based mechanism to select a combination method every time that the search calls for the combination of two reference solutions. The initial probability of selecting one of the three methods is set to 1/3 at the beginning of the search. That is, initially, the three methods have the same probability of being selected. The probability values are updated at the end of each SS iteration in order to favor those combination methods that produce solutions of higher quality. We consider quality solutions those that are admitted to the reference set and therefore we maintain the counts $q(i)$ of the solutions generated with combination method CB_i that were added to *RefSet* in step 7 of Figure 1. The probability $P(i)$ of choosing combination method CB_i (to perform step 5 in Figure 1) is then given by:

$$P(i) = \frac{33 + q(i)}{99 + q(1) + q(2) + q(3)}$$

Note that at the beginning of the search, the *q*-counts are at zero, however, the probability calculation assumes that 99 reference solutions have been generated and that each combination method has contributed to exactly one third of them.

6. Computational Results

All experiments were performed on a personal computer with a 3.2 GHz Intel Xenon processor and 2.0 GB of RAM. For our computational experiments we employed the following sets of existing test problem instances.

Set 1 — This set consists of instances generated by Helmberg and Rendl (2000) who used *rudy*, a machine independent graph generator by Giovanni Rinaldi, to create 54 instances ranging from n = 800 to 3000. They consist of toroidal, planar and random graphs with weights taking the values 1, 0, or -1. Burer et al. (2001) and Festa et al. (2002) used these graphs in their experiments.

Set 2 — This set contains 30 instances described in Festa et al. (2002). The first ten are small size instances with an average of 128 vertices and 300 edges, the second

ten instances are medium size graphs with 1000 vertices and density equal to 0.60%, and the last ten are the large-size instances with 2744 vertices and density equal to 0.22%. The weight values are either 1, 0, or -1.

In our first preliminary experiment we compare the three construction methods for the Max-Cut problem described in Section 2. We propose a measure of diversity and a measure of quality to compare the performance of the construction methods, C_1, C_2 and C_3, as well as a "pure random" method used as a baseline. C_2 depends on the parameter α — denoted $C_2(\alpha)$ — and we test three different variants: $C_2(0.25)$, $C_2(0.5)$, and $C_2(0.75)$. Similarly, C_3 depends on the parameter β — denoted $C_3(\beta)$ — and we test three different variants: $C_3(0.25)$, $C_3(0.5)$, and $C_3(0.75)$. It has been shown that an effective diversification generation method provides a balance between solution quality and solution diversity. That is, solutions must be of reasonable quality and be sufficiently scattered in the solution space to allow the local search and combination methods to reach different local optima. We have selected 15 representative instances in Set 1, with different sizes and densities, to compare the eight construction methods (considering the variants introduced by the several parameter values). We generated 100 different solutions to each instance with each method.

We use the average objective function value of the 100 solutions as a measure of the quality of each method. We use the average distance between each pair of solutions as a measure of the diversity of each method. The distance between two solutions is calculated as described earlier (Section 5.1), i.e., by counting the number of edges in the cut that are different. Table 2 shows, for each construction variant, the average of the quality measure (Quality), the average of the diversity measure (Diversity) and the average of the CPU time in seconds for all 15 instances used in this experiment.

The results in Table 2 indicate that the $C_3(0.75)$ method is quite balanced, with the best average quality of 8190.6, a relatively high average diversity value of 7787.5 and a modest average computational time of 18.2 seconds. As expected, the Random construction obtains the highest diversity value but the worst average quality. The GRASP construction due to Festa et al. (2002), C_1, obtains a diverse set of solutions of reasonable quality at a large computational expense. Given these results, we selected $C_3(0.75)$ as the diversification generation method for our SS implementation.

	Quality	**Diversity**	**CPU time**
Random	6560.2	9808.9	1.7
C_1	7894.1	8273.1	186.6
$C_2(0.25)$	7695.1	8318.3	22.4
$C_2(0.50)$	7991.5	7764.0	21.0
$C_2(0.75)$	8109.2	7270.9	18.0
$C_3(0.25)$	8150.3	7541.5	17.0
$C_3(0.50)$	8173.0	7688.3	16.8
$C_3(0.75)$	8190.6	7787.5	18.2

Table 2. Quality and diversity of construction methods

In our second preliminary experiment we undertake to compare the effectiveness of the improvement methods described in Section 3. As baseline case, we consider the combination of the existing construction method C_1 and the existing improvement method LS_1. We compare it to the combination of our proposed construction method C_3

and the ejection-chain-based improvement method LS_2 with several *k*-values. To do this, we generate a set of 100 solutions with C_1 and apply the improvement method LS_1 to them. Similarly, we generate a set of 100 solutions with C_3 and apply the improvement method LS_2 to them. In order to study the impact of the parameter *k*, we repeat this experiment for five different *k* values (*k* = 1, 2, 3, 4 and 5). Table 3 shows the average objective function values (Value) obtained by each method, the average percent deviation (Dev) from the semidefinite programming upper bound SDP (Goemans and Williamson, 1995) and the CPU time in seconds needed to construct and improve all 100 solutions.

	$C_1 + LS_1$	$C_3+LS_2(k=1)$	$C_3+LS_2(k=2)$	$C_3+LS_2(k=3)$	$C_3+LS_2(k=4)$	$C_3+LS_2(k=5)$
Value	8210.6	8295.8	8336.1	8347.3	8307.9	8338.4
% Dev.	7.56%	6.60%	6.14%	6.02%	6.46%	6.12%
CPU	1786.7	214.3	393.7	416.5	322.6	573.7

Table 3. Improvement methods

Table 3 shows the superior performance (in terms of both solution quality and computational time) achieved by the combination of the construction method C_3 with the improvement method LS_2 over the combination of C_1 and LS_1. This table also shows that, in general, computational time increases with the value of *k*. In terms of average quality, all the C_3+LS_2 variants are statistically equivalent. However, in an instance-by-instance comparison, they tend to complement each other by alternating in finding the best local optimum. This is why we implemented the mechanism described in Section 5.2. In particular, we generate *k* values between 1 and 5 and randomly assign them to the solutions generated in step 1 of Figure 1. Subsequently, the *k* values associated with trial solution created by the combination method are determined as explained in Section 5.2.

In our next preliminary experiment we compare the performance of the different combination methods: CB_1 — based on scores calculated by considering solution quality, CB_2 — based on an incremental construction from the intersection of the combined solutions, and CB_3 — based on the path relinking methodology. We consider three variants — one for each combination method (SS_CB_1, SS_CB_2 and SS_CB_3) — of our SS with the diversification and improvement methods chosen above.: We also consider the version of scatter search — labeled SS below — in which the three combination methods are selected according to their success in previous iterations, as described in Section 5.3. In all versions, we use $|P| = 100$ and $|RefSet| = 10$, settings that have been shown effective in Laguna and Martí (2003). We use the same 15 instances considered in the previous preliminary experiments. Table 4 shows, for these four methods, the average objective function of the best solution found (Value) the average percent deviation from the SDP upper bound (Dev) and the number of best solutions found (Best).

	SS_CB_1	SS_CB_2	SS_CB_3	SS
Value	8451.73	8452.20	8453.60	8459.60
% Dev.	4.73%	4.72%	4.69%	4.62%
Best	4	6	6	13

Table 4. Comparison of four SS variants

Table 4 shows that the three combination methods provide similar results when used in exclusivity and that they complement each other well when used as a group. As part of this experiment, we tracked the evolution of the best solution found with the SS. Figure 3 depicts — for each SS iteration — the value of the best solution in the *RefSet* (Best RefSet) the value of the best solution obtained with the combination of the solutions in the *RefSet* (Best Comb) and the value of the best solution resulting from the application of the improvement method to the combined solutions (Best Imp). The first "Best RefSet" value shown in Figure 3 corresponds to the value after performing step 3 in Figure 1. All other values are obtained after the execution of the while-loop in Figure 1.

Figure 3 shows the contribution of the combination and improvement methods to the best solution found. In the first SS iterations, the combination and improvement methods are able to produce solutions of significantly better quality than those currently in the *RefSet*. As the search progresses, the improvement upon the reference solutions becomes marginal. This is the typical behavior of SS (and most metaheuristic searches). When no new solutions are admitted to the reference set a rebuilding step is performed. The first rebuilding step in Figure 3 takes place at iteration 9.

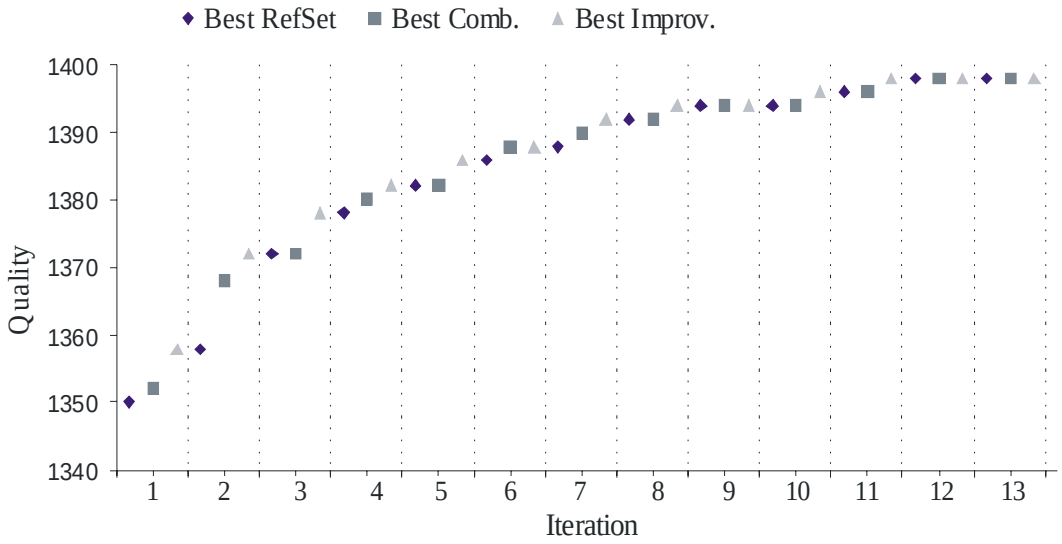

Figure 3. Best value evolution

In the next experiment we compare our SS method with the Variable Neighborhood Search coupled with Path Relinking (VNSPR) due to Festa et al. (2002) and with the CirCut heuristic due to Burer et al. (2001). Festa et al. (2002) proposed six different heuristics for this problem and showed by way of experimentation that the VNSPR outperforms the others. In that experimentation, the solution methods were run for extremely long times. Specifically, the VNSPR required from 2.8 hours on a 1000-vertex instance to 54.9 hours on a 2000-vertex instance. In our tests we use the VNSPR parameters recommended by its authors but limit the execution of the methods to 0.5 hours approximately and report the best solution found within this time. As part of this experiment, we also compare our advanced scatter search method (SS) with the standard scatter search (Basic SS) to measure the contribution of the advanced elements proposed in Section 5. The standard SS algorithm follows the same template as the advanced except that it uses the following elements: the initial population of solutions is generated with C_1, improved with LS_1 and combined with CB_1. Both methods, Basic SS and SS,

terminate after 5 rebuilding steps. Table 5 summarizes the results of applying the solution methods to the 24 instances of Set 1 that Festa et al. (2002) used in their experimentation. This table shows, for each instance and each method, the relative percent deviation from the SDP upper bound and the CPU time in seconds.

Table 5 shows that Basic SS, SS and CirCut methods provide better solutions than VNSPR in the allotted computational time. In the 24 instances reported in this table, Basic SS, SS and CirCut exhibit an average percent deviation of 6.5%, 5.6% and 5.8% respectively achieved in 1042.3, 549.1 and 147.0 seconds of computer time, which compares favorably with the 9.7% deviation of the VNSPR achieved in 2060.3 seconds. SS produces solutions of slightly better quality than the solutions obtained by the CirCut method, although SS consumes more CPU time. SS obtains 16 best solutions while CirCut obtains 12 out of 24 instances. This table also shows the contribution of the advanced elements proposed in Section 5, since the Basic SS method on average finds inferior solutions than the advanced SS even if we allow the basic procedure to run longer.

	Basic SS		SS		VNSPR		CirCut	
	Dev	Time	Dev	Time	Dev	Time	Dev	Time
G1	4.1%	569.0	3.7%	139.0	4.5%	1864.0	3.8%	204.6
G2	4.1%	643.9	3.8%	167.2	4.5%	2001.0	3.9%	210.4
G3	4.0%	619.3	3.7%	180.1	4.3%	1914.0	3.8%	218.6
G11	11.3%	335.1	10.3%	171.8	20.6%	1822.0	11.0%	25.2
G12	12.7%	326.5	11.1%	241.5	20.1%	1804.0	11.4%	26.2
G23	11.6%	315.5	10.3%	227.5	19.4%	1810.0	10.7%	25.5
G14	4.3%	412.9	3.9%	186.5	6.5%	1839.0	4.0%	71.8
G15	4.4%	417.0	3.7%	142.8	6.1%	1915.0	3.8%	68.7
G16	4.3%	426.6	4.0%	161.9	6.3%	1826.0	3.9%	68.2
G22	6.0%	1889.0	5.5%	1335.8	6.7%	3534.0	5.5%	312.4
G23	6.3%	1832.0	5.7%	1021.7	7.2%	2242.0	5.7%	285.5
G24	6.0%	1877.6	5.8%	1191.0	6.7%	2664.0	5.8%	288.4
G32	12.7%	1804.3	10.3%	900.6	19.7%	2072.0	11.0%	88.1
G33	12.7%	1807.6	11.3%	925.6	20.0%	1941.0	11.7%	80.4
G34	12.7%	1803.9	11.4%	950.2	19.9%	2008.0	11.5%	89.1
G35	4.7%	1802.4	4.4%	1257.5	6.2%	2012.0	4.2%	224.0
G36	4.7%	1805.8	4.2%	1391.9	6.5%	1894.0	4.2%	251.4
G37	4.7%	1804.8	4.3%	1386.8	5.9%	2153.0	4.3%	238.6
G43	5.6%	1098.5	5.2%	405.8	5.8%	2023.0	5.2%	122.4
G44	6.0%	1036.6	5.3%	355.9	6.1%	2054.0	5.4%	111.3
G45	5.6%	1251.4	5.3%	354.3	5.8%	2114.0	5.3%	114.8
G48	0.0%	405.2	0.0%	20.1	8.0%	1962.0	0.0%	113.7
G49	0.0%	365.0	0.0%	35.1	7.8%	1982.0	0.0%	111.9
G50	1.8%	366.2	1.0%	16.8	7.6%	1998.0	1.8%	176.4
Avg.	6.5%	1042.3	5.6%	549.1	9.7%	2060.3	5.8%	147.0

Table 5. Comparison on 21 Set 1 instances

In the next experiment we compare the performance of SS and CirCut over time on 10 large instances (n = 2744) of Set 2. These two methods were run for 2 hours and the best solution found was reported every 200 seconds. The results of this experiment are shown in Figure 4.

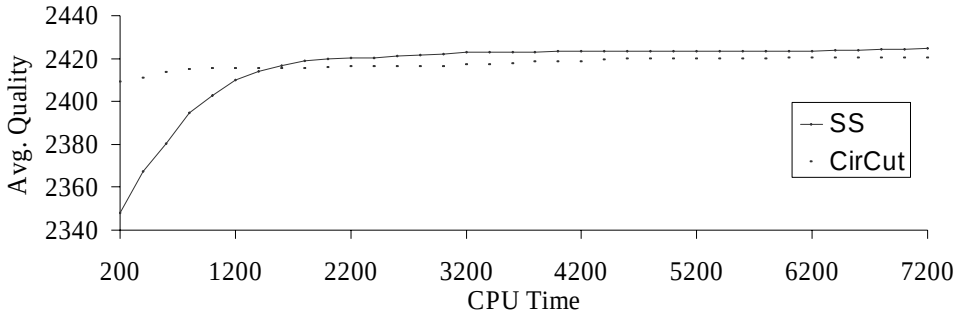

Figure 4. Average best solution value over time

Figure 4 shows that CirCut is capable of obtaining high quality solutions from the very beginning of the search (i.e., within the first 200 seconds). SS requires 1200 seconds to improve upon the solutions found by CirCut but it then maintains its lead during the rest of the execution time. We do not include the VNSPR method in this plot because within the time limit of 7200 seconds considered in this experiment, the average value of the best solution found with the VNSPR is 1906.57. According to Festa et al. (2002) VNSPR requires significantly longer running times than competing procedures in order to obtain high quality solutions. We have verified that in about 50 hours of computer time VNSPR would obtain solutions of a similar or slightly worse quality than those obtained with SS and CirCut.

In our final experiment we compare our SS approach with the CirCut method on the complete set of instances (Set 1 and Set 2). Table 6 shows, for both methods, the average objective function value of the best solution found (Value), the average percent deviation from the best known solution (Dev), the number of best solutions found (Best) and the CPU time in seconds. We do not show the deviation from the SDP upper bound because it is not possible to compute it for the larger instances. In this experiment we run the CirCut method for a number of iterations such that the final running time approaches the one used by SS, which terminates after 5 *RefSet* rebuilding steps.

	54 Set 1 instances		**30 Set 2 instances**	
	CirCut	SS	CirCut	SS
Value	279999	280126	34086	34132
Dev.	0.18%	0.10%	0.16%	0.09%
Best	24	35	18	23
CPU	626.60	620.96	513.77	537.33

Table 6. Comparison of best methods

Table 6 shows that both methods, SS and CirCut, are able to obtain high quality solutions for the Max-Cut problem in relatively short computational times. SS obtains 0.1 percent deviation in Set 1 and 0.09 in Set 2, while CirCut obtains 0.18 in Set 1 and 0.16 in Set 2. Moreover, SS is able to match 35 out of 54 best known solutions in Set 1 and 23 out of 30 in Set 2, while CirCut obtains 24 and 18 respectively.

7. Conclusions

Max-Cut is a computationally difficult optimization problem, which has served us well as test case for a few new strategies that we are proposing to embed in the standard scatter search framework. It is always difficult to choose a classical optimization problem to try new ideas because state-of-the-art procedures for such problems tend to

be the result of years of effort and hence are highly efficient. We feel comfortable with what we have accomplished here because we have not only managed to compete with the state-of-the-art but in fact have pushed the envelope a little further. Obviously, the results that we obtained with our SS implementation are not all due to the strategies that we wanted to test and that we describe in Section 5. Performance was definitely enhanced by the context-specific methods that we developed for the Max-Cut problem. However, our preliminary experiments do show the merit of the mechanisms in Section 5 that we hope other researchers might find effective and eventually could become standard in future SS implementations.

Acknowledgments

This research has been partially supported by the *Ministerio de Educación y Ciencia* of Spain (Grant Refs. TIN2005-08943-C02-02, TIN2006-02696), by the Comunidad de Madrid – Universidad Rey Juan Carlos project (Ref. URJC-CM-2006-CET-0603) and by the Generalitat Valenciana (Ref. GV/2007/047).

References

Burer, S., R.D.C. Monteiro and Y. Zang (2001) “Rank-two relaxation heuristics for max-cut and other binary quadratic programs”, *SIAM Journal on Optimization* 12, 503-521.

Chang K.C. and D.-Z. Du. (1987), "Eficient algorithms for layer assignment problems", *IEEE Transaction on Computer-Aided Design*, 6:67-78.

Chen, R., Y. Kajitani, and S. Chan (1983), "A graph-theoretic via minimization algorithm for two layer printed circuit boards", *IEEE Transaction on Circuits and Systems* 30, 284-299.

Festa, P. and M.G.C. Resende (2001) "GRASP: an annotated bibliography". M.G.C. Resende, P. Hansen, eds. Essays and Surveys in Metaheuristics. Kluwer Academic Publishers, Boston, MA 325-367.

Festa, P., P.M. Pardalos, M.G.C. Resende and C.C. Ribeiro (2002) “Randomized heuristics for the max-cut problem”, *Optimization methods and software* 7, 1033-1058.

Glover, F. (1996) "Ejection chains, reference structures and alternating methods for traveling salesman problems" *Discrete Applied Mathematics* 65, 223-253

Glover, F. and M. Laguna (1997) *Tabu Search*. Kluwer Academic Publisher.

Goemans, M.X. and D.P. Williams (1995), Improved approximation algorithms for the max-cut and satisfability problems using semidefinite programming, *Journal of the ACM* 42, 1115-1145.

Haglin, D.J. and S.M. Venkatesen (1991), “Approximation and intractability results for the maximum cut problem and its variants”, *IEEE Transactions on Computers* 40, 110-113

Helmberg, C. and F. Rendl (2000) “A spectral bundle method for semidefinite programming”, *SIAM Journal on Optimization* 10, 673-696.

Hofmeister, T. and H. Lefmann (1995), “A combinatorial design approach to max cut”, Lecture Notes in Computer Science, 1046, 441-452

Homer, S. and M. Peinado (1997), “Design and performance of parallel and distributed approximation algorithms for max cut”, *Journal of Parallel and Distributed Computing* 46, 48-61

Karp, R.M. (1972) “Reducibility among combinatorial problems”, In: R. Miller and J. Tatchers (Eds.), *Complexity of computer computations*, 85-103, Plenun Press, New-York.

Laguna, M. and R. Martí (1999) "GRASP and Path Relinking for 2-Layer Straight Line Crossing Minimization", *INFORMS Journal on Computing* 11(1) 44-52

Laguna, M. and R. Martí (2003) *Scatter Search: Methodology and Implementations in C*, Kluwer Academic Publishers: Boston.

Lin, S. and B. Kernighan (1973), “An effective heuristic algorithm for the traveling salesman problem”, *Operations Research* 21, 498-516.

Poljak, S. and Z. Tuza (1982), “A polynomial algorithm for constructing a large bipartite subgraph, with an application to a satisfiability problem”, *Canadian Journal of Mathematics* 34, 519-524

Sahni, S. and T. Gonzales (1976), “ P-Complete approximation problem”, *Journal of the Association for Computing Machinery* 46, 48-61