Scatter Search for Binary Optimization Problems

Francisco Gortázar* Abraham Duarte* Manuel Laguna† Rafael Martí‡

*Departamento de Ciencias de la Computación, Universidad Rey Juan Carlos
Tulipán s/n, 28933, Madrid, Spain
{francisco.gortazar,abraham.duarte}@urjc.es

†Leeds School of Business, University of Colorado at Boulder
995 Regent Drive, Boulder, Colorado
laguna@colorado.edu

‡Departamento de Estadística e Investigación Operativa, Universitat de València
Dr. Moliner 50, 46100, Valencia, Spain
rafael.marti@uv.es

1 Introduction

The purpose of this paper is to develop a black-box method for solving an important class of combinatorial optimization problems. Specifically, we tackle problems whose solutions can be represented with a vector of binary variables. The main contribution of our work is the development of a procedure based on the scatter search methodology that can be used for searching the solution space of optimization problems whose solutions can be represented as binary vectors. The procedure is general in the sense that it is not customized to solve a particular problem in this class. When solving an optimization problem, a common practice is to develop a context-dependent method (i.e., a procedure that explicitly exploits the structure and properties of the problem in order to conduct an efficient search). Most heuristic and metaheuristic implementations are based on this paradigm, i.e., developing a specialized method for a specific problem. Our current goal is different, as we attempt to create a method that finds solutions of reasonable quality for a class of problems based on a context-independent paradigm.

Black-box solvers (also referred to as general-purpose optimizers) based on metaheuristics have found their home in commercial implementations. We have identified three software products based on metaheuristic technologies, which are able to solve general classes of binary problems. The Premium Solver by Frontline Systems, Inc (http://www.frontsys.com/), the OptQuest library by Opttek Systems, Inc (http://www.opttek.com/) and Evolver by Palisade Corporation (http://www.palisade.com/). In this paper we first propose a black-box solver for binary problems and then compare it with Solver, OptQuest and Evolver when solving two well known binary optimization problems: the max-cut and the maximum diversity problem.

2 Scatter Search Procedure

Scatter Search (SS) is a metaheuristic that explores solution spaces by evolving a set of reference points. It consists of five methods and their associated strategies. Three of them, the Diversification Generation, the Improvement and the Combination Methods, are typically problem-dependent and are designed for the problem or the class of problems being solved (we devoted a subsection to each of them). For the other two, the Reference Set Update and the Subset Generation Methods, we employ standard implementations [7] as most of the scatter search implementations.

We have adapted SS to develop a black-box solver for optimization problems with binary variables. We assume that a problem instance consists of finding a set of values for $x = (x_1, x_2, ..., x_n)$ - where x_i=0 or 1 - in order to maximize an unknown objective function. The user has the choice of specifying whether the problem is unconstrained or constrained. For unconstrained problems, any binary vector of n elements is a feasible solution. For constrained problems, the solution evaluator returns a Boolean value indicating whether the solution is feasible or not. In other words, the solution method does not know the level of infeasibility or what specific variables need to be given a value of zero to make the solution feasible. The solution method, however, does know that for this type of problems, feasibility may be eventually achieved by switching variable values from 1 to 0.

2.1 Diversification Generation Method

The search starts with the application of a diversification generation method that results in a population P of $PSize$ points from which a subset is selected as the initial reference set ($RefSet$). To obtain solutions with different structures we apply three different generators of binary vectors and create $PSize/3$ solutions with each one of them.

We first create $PSize/3$ solutions with a systematic generator [7] and then compute $score(i)$ for each variable x_i to estimate its contribution to the objective function value in order to generate the remaining $2PSize/3$ solutions considering both quality and diversity. The score calculation is such that a large (small) score value for a variable indicates that, historically, objective function values of higher quality are obtained when this variable takes on the value of 1 (0).

The second generator, G2, constructs a solution step by step, starting with all variables set to 0, and switching in each step one variable from 0 to 1. The variables are randomly selected and the probability of changing a selected variable from 0 to 1 is given by $Prob(x_i = 1) = 0.1 + score(i)$. The addition of 0.1 to the computation of the probability reflects a bias to change the variable value from 0 to 1 when the score is close to 0.5. For unconstrained problems, G2 performs steps as long as the solution under construction improves. For constrained problems, G2 performs steps as long as the solution under construction remains feasible.

The third generator, G3, can be viewed as a destructive method. It was suggested in [5] and adapted in [2]. It starts with all variables set to 1 and at each step it switches the value of one variable from 1 to 0. Variables are probabilistically selected according to their score values. The complement of the scores (i.e., $1 - Prob(x_i = 1)$) is used to calculate the probability of changing the value from 1 to 0. The stopping criterion is customized for both type of problems in a similar way as in G2.

The initial $RefSet$ must balance solution quality and diversity, and we follow the standard reference set update method, selecting the best $b/2$ solutions (where $b = |RefSet|$) from P and then the $b/2$ solutions in P that are most diverse with respect to those already in the $RefSet$ (computing the maximum of the minimum distances between each solution and the solutions already in $RefSet$ as it is customary in scatter search). Diversity is measured according to the following distance function between solutions. The distance between solutions $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_n)$ is given by $d(x, y) = \sum_{i=1}^{n} |x_i - y_i|$.

We follow the standard subset generation method and the procedure generates all pairs of reference solutions that have not been combined before. We limit the application of the improvement method to promising solutions. Specifically, we store the trial solutions resulting from the application of the combination method in a temporary $Pool$, and we apply the improvement method to the $b/2$ best solutions in $Pool$. The reference set is then updated by selecting the best b solutions from the union of the $RefSet$ and $Pool$ (where the improved solutions replaced the original ones in $Pool$). If the combination method is incapable of creating solutions that can be admitted to the $RefSet$, the reference set is rebuilt. The rebuilding consists of keeping the best $b/2$ solutions intact and replacing the worst $b/2$ solutions in the $RefSet$ with new diverse solutions from P. The method stops after a pre-established $MaxTime$ CPU seconds.

2.2 Improvement Method

The improvement method consists of a local search procedure. It is applied to the best $b/2$ solutions in the $RefSet$. A global iteration of the improvement method consists of three steps. First, we construct the candidate list $CL = \{x_i | (x_i = 0 \wedge score(i) \geq th_1) \vee (x_i = 0 \wedge score(i) \leq th_2)\}$ of elements (variables) to be changed. CL contains the variables with a value of 0 and a large score value (close to 1), as well as the variables with a value of 1 and a small score value (close to 0). Therefore, according to the score information we should flip (from 1 to 0 and from 0 to 1) the value of the variables in CL.

In the second step we order the elements (variables) in CL according to their score value (where those with the largest score are located first) and then, in the third step, we scan them in this order in search for improving moves. We perform several iterations in the third step alternating flip and swap moves, where a swap consists of exchanging the values of two different variables. In the first iteration we try flip moves with all the elements in CL (examined in order), changing their value if it improves the objective function. The CL is reconstructed at this point. Then, in the second iteration we consider swap moves with all the elements in CL (examined in order), in which we try to exchange the value of each variable with the value of another variable. To do this we implement a first strategy, which scans the list of variables in search for the first one whose movement results in a strictly positive change of the objective function value. After this process, the CL is reconstructed again. Further iterations are performed alternating between flip and swap moves. The method stops after *MaxImpIter* iterations or before, if no improvements are achieved after trying all flips and swaps.

2.3 Combination Method

The reference set is a collection of b solutions that are used to generate new solutions by way of applying a combination method. In order to design a context-independent methodology that performs well across a wide collection of different binary problems, we propose a set of six combination methods from which one is probabilistically selected according to its performance in previous iterations (i.e., the selection process is reactive). This method was successfully applied by [1].

Solutions in the $RefSet$ are ordered according to their objective-function value (where the best solution occupies the first place). When a solution obtained with a combination method CM_i qualifies to be the jth member of the current $RefSet$, we add $b-j+1$ to $success(CM_i)$. Therefore, combination methods that generate good solutions accumulate higher success values and increase proportionally their probability of being selected. To avoid initial biases, this mechanism is activated after the first $InitIter$ combinations, and before this, selections are made completely at random. A description of the combination methods follows. These methods generate the new trial solution z from the combination of two reference solutions x and y. It should be mentioned that the *score* value, initially computed with the solutions in P, is updated during the entire search process, thus providing an estimation of the contribution of each variable (when it takes on the value of 1) to the objective function value.

The CM_1 combination method first computes the partial solution z formed with the "union" of x and y, that is $z_i = 1$ if $x_i = 1$ or $y_i = 1$, otherwise $z_i = 0$. It then performs a series of steps switching some variables from 1 to 0 in a similar way as the G3 generator. Specifically, at each step, the method uses the *score* values to select probabilistically one variable with the value of 1 to switch it to 0. For unconstrained (constrained) problems, it continues in this fashion while the solution improves (remains feasible). The CM_2 combination method is similar to CM_1 with the only difference that the variable selection (to switch the value from 1 to 0 in z) is performed completely at random, that is, without considering the *score* values.

The CM_3 combination method is an adaptation of the one proposed by [7] in the context of the knapsack problem. The method calculates a weight for each variable, based on the objective function value of the two reference solutions being combined. The weight for variable i that corresponds to the combination of reference solutions x and y is calculated with the following formula:

$$weight(i) = \frac{f(x)x_i + f(y)y_i}{f(x) + f(y)} \tag{1}$$

In this formula $f(x)$ is the objective function value of solution x and x_i is the value of the ith variable. Then, the trial solution z is constructed by using the weight as the probability for setting each variable to one, i.e., $Prob(z_i = 1) = weight(i)$.

The CM_4 combination method first computes the partial solution z formed with the "intersection" of x and y. A number of steps are then performed in which a variable with the value of zero is chosen. The probability of selecting variable i is proportional to $weight(i)$. (Note that the larger the weight the more attractive it is, at least from the history of the search, to set the value of the variable to one.) The difference between CM_3 and CM_4 is that CM_3 starts the process of switching variable values from 0 to 1 with all the variables set to 0 while CM_4 starts with the partial solution z that represents the intersection of the reference solutions being combined. CM_4 stops in the same

way as CM_1, depending on the type of problem.

The CM_5 combination method is very similar to CM_4. Starting from the partial solution constructed with the intersection of the reference solutions, variables that are set to 0 are chosen to change their value to 1. The only difference with CM_4 is that variables with a value of 0 are selected completely at random, that is, ignoring their weight values.

The CM_6 combination method starts with all the variable values set to 0. Then, it applies the G2 constructive method with the restriction that the z_i variables that are candidates to be switched to 1 are those with a value of 1 in x or y (i.e., those for which $x_i + y_i \geq 1$).

3 Experimental Results

This section describes the computational experiments that we have performed to test the efficiency of our context-independent scatter search procedure as well as comparing it with various methods from the literature. We have implemented the methods in Java SE 6 and all the experiments were conducted on a Pentium 4 computer at 3 GHz with 2 GB of RAM.

We have used two combinatorial optimization problems to test our procedure. Solutions to these problems are naturally represented as binary vectors: the max-cut problem and the maximum diversity problem.

- 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 on the edges connecting the two subsets is maximized. A solution to this problem can be represented as a binary vector with cardinality equal to the number of nodes in the graph (where the value 0 or 1 indicates that the associated node belongs to one or other subset). The max-cut problem falls within the class of binary unconstrained problems. We will compare our method with the recent specialized SS algorithm [8] that was shown to outperform existing methods. We consider 94 instances from two sources. The Hartmann problems are 10 instances with 125 nodes and 375 edges all with weight values equal to -1 or 1. The second set consists of 84 instances (n = 50 to 300) generated with rudy - a machine independent graph generator by Giovanni Rinaldi. The toroidal, planar and random graphs have weights taking the values of -1, 0, or 1.
- The *maximum diversity problem* (MDP) consists of selecting a subset of k elements from a set of n elements in such a way that the sum of the distances between the chosen elements is maximized. Clearly, a solution to this problem can be represented as a binary string x, where variable x_i takes on the value of 1 if element i is selected and 0 otherwise, $i = 1, \ldots, n$. There is only one constraint in this problem that forces that exactly k variables in the string are assigned the value of 1. This problem belongs to the class of binary constrained problems. We will compare our method with the recent specialized SS algorithm in [4] since it has been shown to outperform previous approaches. We consider 92 instances from two sources. The first one with 40 instances, referred to as Glover, for which the values are calculated as the Euclidean distances from randomly generated points with coordinates in the 0 to 100 range. The second one with 52 instances, referred to as Silva, for which the values are integer numbers randomly generated between 50 and 100. The value of n ranges from 50 to 300 and the value of k ranges from $0.1n$ to $0.3n$.

In each experiment we compute for each instance, the overall best solution value, BestValue, obtained by all executions of all methods. Then, for each method, we compute the relative percent deviation between the best solution value (Value) obtained with the method and BestValue for that instance. We report the average percent deviation (Dev.) across all the instances considered in each particular experiment. We also report, for each method, the number of instances (#Best) for which the value of the best solution obtained with this method matches BestValue. In addition, we calculate the Rank statistic - proposed by [9] - associated with each method. For each instance, the n_rank of a method M is defined as the number of methods that found a better solution than the one found by M. In case of ties, all the methods receive the same n_rank, equal to the number of methods strictly better than all of them. The value of Rank is the sum of the n_rank values for all the instances in the experiment, thus, the lower the Rank the better the method.

In our preliminary experimentation with 90 representative instances (42 Max-cut and 48 MDP) we set the parameters (th_1, th_2) equal to (0,1) and the $MaxImpIter$ parameter equal to 30. We compare our scatter search implementation (Black-Box SS) against the three well known commercial solvers mentioned in the introduction: the Evolutionary Premium Solver by Frontline Systems in the Solver Software Development Kit (version 7.2), OptQuest by OptTek Systems (version 6.2) and Evolver by Palisade Corporation in the Evolver Development Kit (version 4.1.2). We also compare with the best known solutions mainly obtained with the specialized SS algorithms also included in these experiments. Tables 1 and 2 report the Dev., #Best and Rank values for the five methods considered on the max-cut and MDP problems respectively. The termination limit was set to 30 seconds for all the methods.

Table 1: Max-cut (94 instances).

Method	Dev.	#Best	Rank
OptQuest	17.49%	0	358
Evolver	17.11%	3	249
Solver	8.10%	5	172
Black-Box SS	4.08%	17	82
Specialized SS [8]	0.00%	90	0

Table 2: Maximum diversity problem (92 instances).

Method	Dev.	#Best	Rank
OptQuest	14.18%	0	367
Evolver	9.03%	0	259
Solver	2.43%	1	179
Black-Box SS	0.17%	36	57
Specialized SS [4]	0.06%	87	7

The behavior of Black-Box SS is fairly consistent across both problem classes. The method delivers low relative percent deviations with respect to the best-known solutions and always ranks best among the black-box solvers, which are also based on evolutionary strategies. As expected, the solutions of all the black-box solvers are not as good as those obtained with the specialized, and

best-known, algorithms. However, they can be applied to a wide range of binary problems while the specialized SS algorithms considered can only be applied to one particular problem.

4 Conclusions

We have described the development of a scatter search application to optimization of problems whose solution representation is given by a binary vector. The results of our experiments are quite conclusive regarding the effectiveness of the method that we have developed. In the process of developing this method, we are able to show the advantages of using multiple combination methods and at the same time the disadvantage of reducing the size of the neighborhood in the improvement method. The first-improving strategy when applied to the complete neighborhood resulted in an effective use of the allotted searching time. We expect that our experience will help software developers to improve upon the commercial solvers that are based on evolutionary processes.

5 Acknowledgement

This research has been partially supported by the Ministerio de Ciencia e Innovación of Spain (Grant Ref. TIN2006-02696), by the Comunidad de Madrid - Universidad Rey Juan Carlos project (URJC-CM-2008-CET-3731) and by the Government of Castilla y León (BU008A06).

References

[1] V. Campos, M. Laguna, and R. Martí. Context-Independent Scatter and Tabu Search for Permutation Problems. *INFORMS Journal on Computing*, 17(1):111–122, 2005.

[2] A. Duarte, R. Martí. Tabu Search for the Maximum Diversity Problem. *European Journal of Operational Research*, 178:71–84, 2007.

[3] P. Festa, P.M. Pardalos, M.G.C. Resende, and C.C. Ribeiro. Randomized Heuristics for the Max-cut Problem. *Optimization Methods and Software*, 7:1033–1058.

[4] M. Gallego, A. Duarte, M. Laguna, and R. Martí. Heuristics Algorithm for the Maximum Diverstity Problem. *Computational Optimization and Application*, In Press.

[5] F. Glover. A Template for Scatter Search and Path Relinking. *Lecture Notes in Compuer Science*, 1363:13–54, 1998.

[6] M. Laguna, and R. Martí. GRASP and Path Relinking for 2-Layer Straight Line Crossing Minimization. *INFORMS Journal on Computing*, 11:44–52

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

[8] R. Martí, A. Duarte, and M. Laguna. Advanced Scatter Search for the Max-Cut Problem. *INFORMS Journal on Computing*, 21(1):26–38, 2009.

[9] 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.

[10] S. Sahni, and T. Gonzales. P-Complete Approximation Problem. *Journal of the Association for Computing Machinery*, 46:48–61, 1976.

[11] G.C. Silva, L.S. Ochi, and S.L. Martins. Experimental Comparison of Greedy Randomized Adaptive Search Procedures for the Maximum Diversity Problem. *Lecture Notes in Computer Science*, 3059:498–512.