FOM: A Framework for Metaheuristic Optimization

J.A. Parejo[1], J. Racero[1], F. Guerrero[1], T. Kwok[2], and K.A. Smith[2]

[1]Escuela Superior de Ingenieros, Camino de los Descubrimientos, s/n, 41092 Sevilla, Spain
[2]School of Business Systems, Monash University, 3800 Clayton, Vic Australia

Abstract. Most metaheuristic approaches for discrete optimization are usually implemented from scratch. In this paper, we introduce and discuss FOM, an object-oriented framework for metaheuristic optimization to be used as a general tool for the development and the implementation of metaheuristic algorithms. The basic idea behind the framework is to separate the problem side from the metaheuristic algorithms, allowing this to reuse different metaheuristic components in different problems. In addition to describing the design and functionality of the framework, we apply it to illustrative examples. Finally, we present our conclusions and discuss futures developments.

1 Introduction

In the past 15 years, a large amount of work has been carried out on development and research about heristics and metaheuristics. However, most of the efforts were directed to solve classical OR problems using this techniques, trying to improve results obtained using traditional techniques, and showing that this alternative methods are relevant and advantageous. However, that generated a lot of documentation, algorithms and researches about this techniques applied and adapted to this problems.

Until now, to apply this techniques we had to make a big effort to develop and adapt them to specific problems. This effort is added to the inherent complexity of some of this techniques, and make that in some cases these techniques were not applied. To avoid this, we have developed a tool that is aimed at making applicable these techniques in real production environments, with a minimum effort on development and adaptation..

The developed tool is called a Framework[1] in terms of computer science. This tool is flexible and extendable enough so as to be used in heterogeneous environments and to let the user to tune and retouch the needed items in order to adapt it to his needs.

In this sense our tool try to avoid ad-hoc implementations for specific problems, that reduce to the maximum the computational time to obtain a solution insignificantly better. However our tool try to make applicable these methods to a increasing number of problems in a easy, simple and fast way. And our tool is open to the numerous

[1] A framework is something more than a library. A frameworks is not only functional bat, also contains a design which can be used, adapted and completed so as to fit the needs of the user. A framework defines a entire structure for an application. The reader can find a more detailed definition in [5]

variants and chances that this methods bring us. It also let us make implementations of efficient methods that can be adapted to the special characteristics liable to have problems.
Although the object-oriented paradigm has been in use for many years, the production of mainstream, quality frameworks is a comparatively new phenomenon. The application of frameworks to combinatorial optimization is even more recent. Therefore we can say that this is a new area of research and development, that can contribute many advantages in order to approach optimization problems.
There are some precedents of frameworks of this type. Relevant developments in this way have been the Templar framework [1-2], and the HotFrame [3]. However we think that our framework, FOM, has enough innovations on both its design and implementation to consider it an advance in this area. Finally a relevant reference about the state of the art in these developments, and a compendium of the relevant references can be found in [4].

2 Framework Design

As a previous stage to design the framework, we have analysed the relationships between the optimization techniques, their components, the abstract problems and the specific instances of the problems that we want to solve. We have delimited and expressed in a formal way the interfaces between these elements, which are required to solve the optimization problems. Starting from this study, we have designed a structure that let us apply these techniques to the problems, and we have implemented them using the previous interfaces.
We show below a class diagram that shows the final structure of some of these interfaces. These diagrams express the relationships between the solutions to our problems and similar problems, as well as the characteristics that our solutions should have in order to apply our different metaheuristics. The most simple interface, called *Solution*, only defines the requirements that an object must comply with for being a solution to one of our problems: should be able to be evaluated and provide a new random solution to our problem. The design defines furthermore a class that implements this interface, called *AbstractSolution,* and a interface that represents the problem we want to solve, called *Problem*; likewise, it is established a relationship by means of which all *AbstractSolution* is associated to a *Problem.* Through this relationship the problem can evaluate the actual data for the solution, and determine if it is a valid solution. Finally the diagram defines some interfaces that bring the required functionalities to apply the different techniques. For example, we define an interface called *Neighbourhood* that defines the neighbourhood of a specific solution and give us the required operations to move through it.. There is another interface called *ExplorableSolution* that implements the interfaces *Solution* and *Neighbourhood.* In this way, using an object that implements this interface we, can explore the neighbourhood of this solution and "jump" from one solution to other, moving us around the solution space of the problem. This let us apply techniques based on local search. In the same way the different interfaces that are needed to apply the different techniques are defined. Also, the elements used in the different techniques and their functionality are defined.

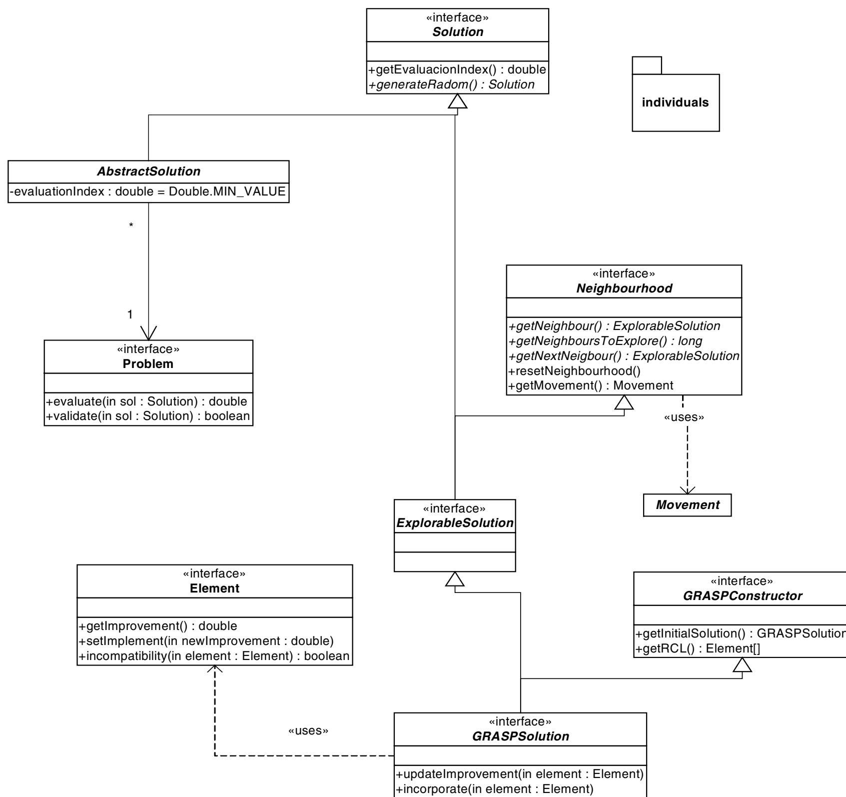


Fig. 1. Class Diagram of Problems and Solutions

In our framework we have implemented the metaheuristics listed below: Steepest Descent, Iterative Steepest Descent, Tabu Search [6], Simulated Annealing [7], GRASP, Variable Neighbourhood Search [8] and Genetic Algoritms [9-10]. One might think that the first two techniques are not important and could not be called metaheuristics. However, they have been implemented because they are useful when they are combined with the other techniques. Thus, the ability to combine several metaheuristics is included in the framework in techniques like GRASP and Genetic Algorithms.

Figure 2 shows both definitions and relationships between the metaheuristics.

A class called *Metaheuristic* is defined in the previous diagram. This class defines the interface that must have every elements that we want to use as an optimization technique in the framework. It only defines a public method, that will be the optimization algorithm, and that has a solution as parameter (the initial solution) and returns another solution (the optimal solution we found). All optimization methods that can be expressed and called in this way could be used in the framework in order to find an optimal solution to a problem.

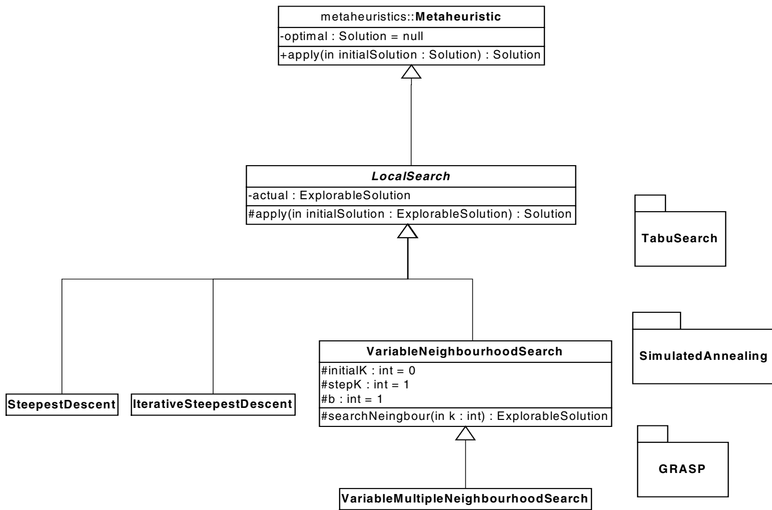


Fig. 2. Class Diagram of Metaheuristics and Local Search

In the previous diagram another class, called *LocalSearch* is defined; it only inherits the previous class, with the particularity that the initial solution must be a *ExplorableSolution* just as was defined in the Figure 1. In this way we can explore the neighbourhood of our solution and implement several techniques based on local search.

Figure 3 shows a diagram that defines the behaviour of Simulated Annealing. In this diagram the class *SimulatedAnnealing* is defined, that implements *LocalSearch.* This class uses a interface called *Cooler*, in order to decrease the value of their attribute *actualTemperature.* The interface *Cooler* defines a method called *toCool*, this method will give us the value of the temperature parameter in the next iteration. Therefore, that which determines the cooling schedule of this algorithm is the concrete *Cooler* object with what is connected the *SimulatedAnnealing* object.

As a result, we have techniques like Genetic Algorithms, Tabu Search or Simulated Annealing implemented in an abstract way. Also they are implemented in a independent way from the details of the problem we want to solve, based only on the specifications defined by our interfaces. Our framework defines a set of elements that could be useful in order to tune the concrete parameters and variants of this metaheuristic techniques that we should use in order to solve a specific problem.

It is widely accepted that a correct tuning of the parameters that control the behaviour of the metaheuristic techniques has big impact on the success of this techniques when solving problems. It is very useful in order to adapt and tune those parameters follow the value of some data along the execution of the metaheuristic technique. We could determine the best initial values that we should use for a metaheuristic, and even

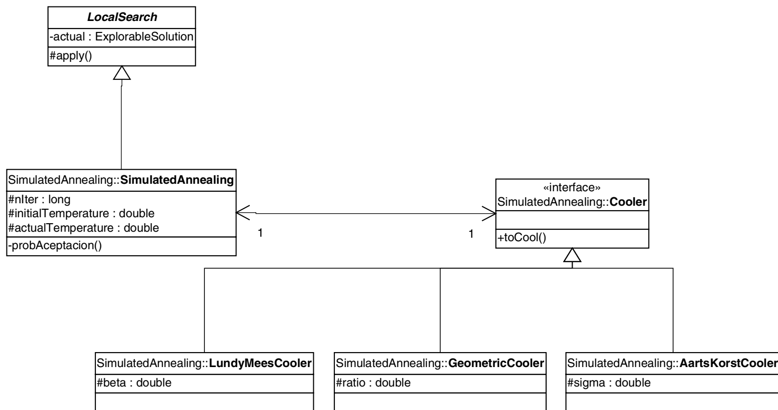


Fig. 3. Class Diagram of Simulated Annealing

determine what variant of a technique should be choosen when a specific problem is being solved This functionality is built into our framework by the use of some elements called "observers". Those elements will trace the value of the property they are following in every iteration of the algorithm. Those elements will let us to follow the evolution of any parameter of a metaheuristic. These parameters could be any of the following: the function value for the best solution we have found, the average function value of the population of a Genetic Algorithm in a iteration, the evolution of the temperature parameter in Simulated Annealing, etc. Moreover, our observers should be implemented only once, and could be used for any problem and metaheuristic technique. Also, it had been developed a set of elements, called "visualizators", that let us treat the information obtained by the observers in different ways. Using those visualizators we could by example, save in a file all the values obtained by the "observers", or show charts about their evolution. Furthermore those visualizators can be applied to any observer we have defined, even those defined for a specific problem by the user. As an example, all charts that we show later in the computational experiences had been obtained using a charting visualizator.
Finally, it is clear that using our framework we can experiment with different variants of this metaheuristic techniques without any codification effort. For example, as we have seen previously figure 3, the cooling schedule that uses Simulated Annealing is determined by the *Cooler* object that has associated. In this way we could change the *Cooler* object using one implementation or another, and subsequently using one cooling schedule or another. In this way we could experiment different cooling schedules for any problem defined for the framework, only changing the implementation of the *Cooler* object that has our Simulated Annealing object. In the same way we could change the type of memory used by Tabu Search, etc. Other relevant point of the framework is the reuse of functional elements across different metaheuristic techniques. Thus, abstraction and division of responsabilities of different functional elements can be reused. For example: in Genetic Algorithms is necessary to have criteria to choose some individuals: choose the individuals that could have a mutation or inversion, choose the individuals that could be crossed and

obtain from them other individuals, and choose the individuals that could survive and to be a member of the population in the next iteration.
In order to achieve this problem we have defined a set of elements called "selectors". We show a class diagram with their structure:

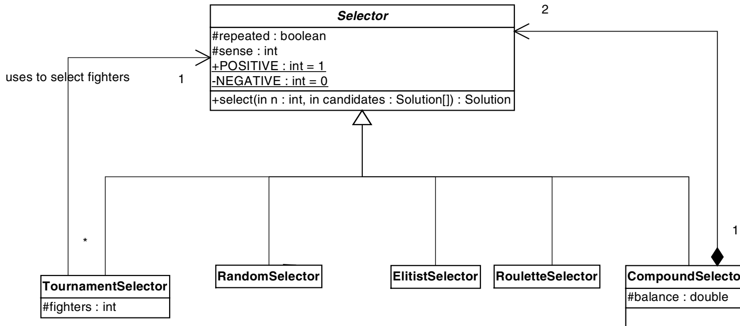


Fig. 4. Class Diagram of selector

Class *Selector* define the selection method, that taking a set of candidate solutions returns a subset that would be chosen. Moreover we define different classes that implements different criteria to choose elements, like tournaments of n elements, random, better, etc. Those elements could be used by the Ant System metaheuristic in order to choose ants that will update the feromona's trail.
There are present again the advantages of a correct abstraction and definition of the responsibilities of functional elements.

3 Computational Experiences

With the aim of proving the efficiency and applicability of the framework. We have approached two classical problems in the area of Operational Research, the SAT problem and the TSP problem. We used standard and public libraries of instances of these problems. We used the SATLib for the SAT problem, that we gathered from: http://www.satlib.org. We used the TSPLib for the TSP problem, that we gathered from: http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95.
We will show some results obtained applying the metaheuristic techniques to one instance of the selected problems, in order to show the capabilities and possibilities that our framework give us. We will show the obtained solutions, and finally we will present (if we consider it is relevant) a chart showing the evolution of interesting parameters. Because of space limitations we will show only the results for techniques that we consider interesting. Moreover, detailed values of parameters used for every technique will not be listed, but we have roughly used standard values from literature However, we have a detailed description of these values and a more exhaustive study of the results obtained applying all techniques, as well as applications of the techniques that form the framework to other instances of this problems.
The SAT problem. We have chosen the SAT's problem instance defined in the file JNH207.SAT of the SATLib to realize our tests. We will show the results obtained applying different techniques.

Tabu Search: Results: We firstly executed this algorithm using a little size for the tabu memory, and few iterations too. We could not find an optimal solution. However, we executed this algorithm again using a bigger memory size and more iterations and we found an optimal solution. Finally we executed this algorithm again in order to show the what is the critical factor, memory size or number of iterations, so we choose the same memory size than in the first application and a large number of iterations, and we could not found an optimal solution. However and according to the results that we can see in the next figures there were necessary a bigger number of iterations. Interesting facts: We show now two charts that show the evolution of the actual solution in every iteration, first with a little memory size and later with a bigger memory size. In the second chart we used a higher number of iterations.

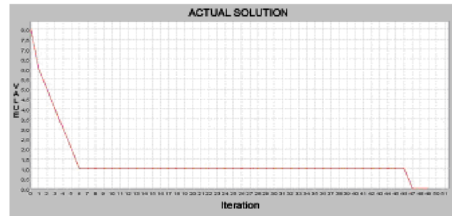
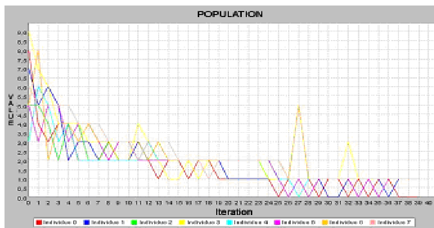


Fig. 5. Tabu Search. Non Optimal Soluts.

Fig. 6. Tabu Search. Optimal Soluts.

Genetic Algorithms: Results: We found an optimal solution every time we applied this technique. Interesting facts: We think that is interesting to have a look to the charts that show the evolution of the number of clauses not satisfied for every individual, and the figure that shows de evolution of the diversity of the population[2]. We can notice the evolution of our population to better solutions, and at the same time how their diversity decrease.



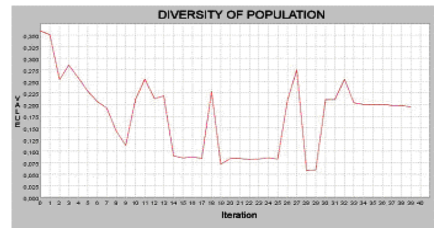


Fig. 7. Evaluation value for the individuals

Fig. 8. Diversity of the population

The TSP problem. In order to carry out our tests we have chosen the TSP's problem instance defined in the file att48.tsp from TSPLib. We will show results we obtained for some techniques:

Tabu Search: Results: We obtained a solution with objective function of 37711.35. It is the best solution obtained in the tests shown. However, It should not be forgotten

[2] Diversity of population is calculated comparing all pairs of individuals evaluating their equality with a value between 0 and 1, finally dividing the sum by the number of individuals.

that we used the highest iteration number of whole of the tests shown in this article. Interesting facts: We show the evolution of the evaluated solution in every iteration.

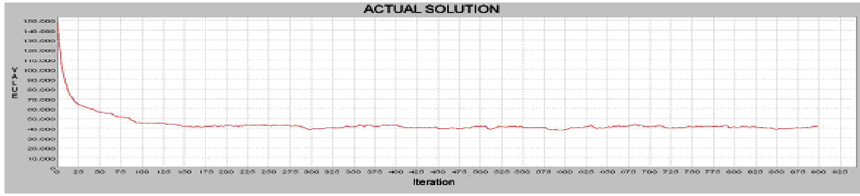


Fig. 9. Evaluated solution, Tabu Search

Simulated Annealing: Results: We obtained a solution with objective function of 54500.51 (the number of iterations was smaller than in tabu search). Interesting facts: We have chosen a cooling schedule that makes the temperature decrease geometrically. Figure 10 shows the curve of the descending temperature. Moreover, a chart that show the value of the actual solution allow us to discern the operation of this algorithm.

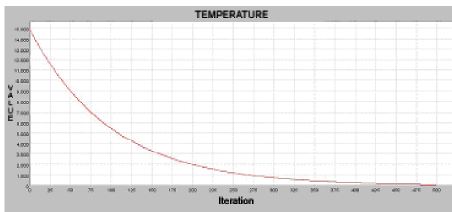


Fig. 10. Temperature Parameter

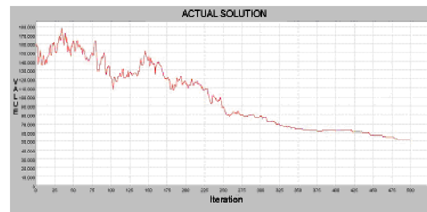


Fig. 11. Actual Solution

Genetic Algorithms: Results: We executed two variants of this technique in order to show their efficiency. In one hand we applied this technique without other mechanism for optimization than mutation, inversion and crossover of individuals, and we obtained a solution with objective function of 81012.90. On the other hand we used an advanced technique, i.e. a steepest descent algorithm applied to all individuals, we obtained better results, the obtained solution was 45817.78. Interesting facts: It is interesting to show the variation between Figures 12 and 13 which show the evolution of the individuals in the population. Evolution showed in Figure 13 is more “jagged”, and has a bigger initial improvement of the population. It was because of the effect of the local search.

4 Conclusions and Future Research

We have verified through the development and use of the framework to solve diverse problems that it can be a useful tool, and that it gives us interesting possibilities and innovative elements. We will enumerate the advantages and drawbacks of the use of our framework.

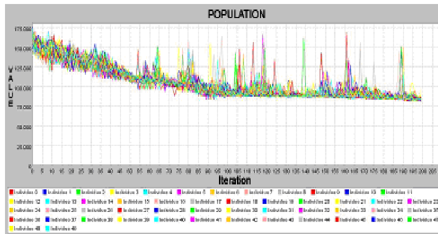


Fig. 12. Simple Genetic Algorithm

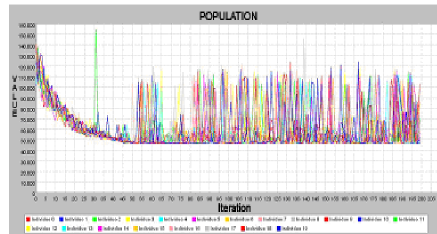


Fig. 13. Hybrid Genetic Algorithm

Advantages obtained by the use of our framework:

- We don't have to make a complete implementation of every technique for every problem we want to solve.
- It makes easy to reuse code. Moreover, it promotes that different techniques made use of the previous development of common elements.
- It induces a correct and robust structure in our implementation. It is produced thanks to the structures where our classes are embed on the framework, and to the study, separation and delimitation of the functional elements that have a role in the optimization process.
- It brings additional elements like "observators", "terminators", "selectors", etc.
- It is flexible and let us to use advanced techniques and variants. Moreover, It is open, so we can add new techniques that take advance of the common elements that had been implemented for other problems or techniques.

Drawbacks of the use of the framework:

- The use of the framework makes our implementation less efficient.
- It is necessary to learn the structure and key classes of the framework, now that it brings us all the possibilities of the framework.
- The use of the framework makes our implementation more complex, in spite of there are less code and elements to implement..

Finally we will show some future research that we think could be useful and interesting about the framework:

- We could add more metaheuristic techniques to the framework, like Ant Systems [11] or Scatter Search [12].
- We could make a distributed implementation of the metaheuristics, and as the user only have to implement the elements about solutions, it could be used without additional effort.
- We could implement a library of standard solutions. The most of the problems use the same data structures in order to represent solutions (arrays, lists, trees). We could implement general implementations of this structures as solutions (adding the necessary elements in order to can apply all techniques). In that way, the user should only use these basic structures in order to express their solutions to the problems.
- Finally we could implement a system that, based on a model of the problem expressed in a modelling language, could "compile" the required elements to can apply our techniques. We would have to use the library of basic standard solutions from the previous item and implement a compiler that compounds the solutions and problems in the correct way. It would bring us to a pseudo-declarative

paradigm of problem solving We only should describe the problem, and could obtain solutions using metaheuristics (probably that solution will not be so good as if we implement the problems and techniques “at hand”, but maybe could give us a solution that solve our problem to our needs).

References

1. M. Jones. An Object-Oriented Framework for the Implementation of Search Techniques. Doctoral thesis, University of East Anglia. 2000.
2. M. Jones, G. McKeown, V. Rayward-Smith. Templar: An object-oriented Framework for distributed combinatorial optimization. In Proccedings of the UNICOM Seminars on Modern Heuristics for Decision Support. UNICOM Ltd, Brunel University, UK
3. Adreas Fink, Stefan Voβ and David L. Woodruff. Building Reusable Software Components for Heuristic Search. 1998.
4. Stefan Voβ, David Woodruff. Optimization Software Class Libraries. Kluwer Academic Publishers, 2002.
5. Grady Booch. Object-Oriented Analysis and Design with Applications (2nd Edition). Addison-Wesley. 1994
6. Fred Glover, Manuel Laguna. Tabu Search. Kluwer Academic Publishers. 1997
7. S. Kirkpatrick, C. D. Gellat and M. P. Vecchi. Optimization by Simulated Annealing. Science, 220, 671–680. 1983.
8. N. Mladenovic, P. Hansen. Variable Neighbourhood Search. Computers & Operations Research, 24, 1097–1100. 1997.
9. David E. Goldberg. Genetic Algorithms in Search, Optimization & Machine Learning. Addison-Wesley. 1989.
10. Randy L. Haupt, Sue Ellen Haupt. Practical Genetic Algorithms. Wiley-Interscience. 1998.
11. M. Dorigo, V. Maniezzo, A. Colorni. Ant system: Optimization by a colony of cooperating agents. IEEE Transactions on Systems, Man, and Cybernetics, B-26: 29–41
12. M. Laguna. Scatter Search. Handbook of Applied Optimization (Pardalos et al editoral). Oxford University Press, 183–193. 2002