

# On the Design of Metaheuristic Algorithms using Fuzzy Rules

**David A. Pelta**

Depto. de CC & I.A  
Universidad de Granada  
18071 Granada, España  
dpelta@decsai.ugr.es

**Alejandro Sancho-Royo**

Depto de Matemáticas  
Escuela de Arte de Granada  
18002 Granada (España)  
alejandrosancho\_royo@hotmail.com

**Jose-L. Verdegay**

Depto. de CC & IA  
Universidad de Granada  
18071 Granada, España  
verdegay@decsai.ugr.es

## Abstract

We present here a co-operative multi-thread metaheuristic. Each thread employs a (possibly) different optimization strategy and they are controlled by a Coordinator process. A relevant point is the use of a fuzzy rule base embedded in the Coordinator to control and modify the behavior of the optimization threads. Preliminary results over the knapsack problem show the benefits of the proposal.

**Keywords:** metaheuristics, optimization, fuzzy sets and systems

## 1 Introduction

It's a basic fact in the optimization area that NP-complete problems cannot be approached with exact tools in reasonable time. So, approximation approaches are required. A very large list of problems that need this approach can be found in diverse areas as Graph Theory, Networks Design, Sets and Partitions, Scheduling, Mathematical Programming, etc, [1, 4].

Metaheuristics are methods that “orchestrate an interaction between local improvement procedures and higher level strategies to create a process capable of escaping from local optima and doing a robust search of the solution space” [6].

Heuristics and metaheuristics cannot guarantee the optimality of the solutions they find. However, in practice, they can provide reasonable good (excellent in some cases) solutions using a

moderate amount of resources. A recent survey of methods and applications appears in [6, 8].

Although heuristics methods can be considered as good tools to solve a lot of optimization problems, there are several difficulties in order to use this methods in optimization. Difficulties arises from different reasons:

1. Given a new problem, it is almost impossible to determine which is the best algorithm to solve it. The researcher must decide among algorithms that 'work well' in similar problems or, simply, he or she choose a known algorithm.
2. Even if we know a good algorithm for the kind of problem that we are dealing with, it can be possible that our problem instance not be a good or medium case for this algorithm, because 'goodness' of algorithm is a statistic behavior.
3. Sometimes, huge computation times to obtain excellent solutions are not affordable. It may be better to search for a 'reasonable good' solution in faster way.
4. Metaheuristics are based in several parameters that need to be adjusted to solve different problems. Usually they are tuned empirically and to the best of our knowledge, no systematic way exists to perform such a task in an analytical way.

In order to approach some of these drawbacks we present a cooperative, multi thread strategy to solve combinatorial problems. Each thread represents a particular optimization algorithm and

concurrent threads are controlled by a Coordinator process whose main tasks are: to collect the performance information of the threads; and to send them orders to alter their search behavior.

The use of fuzzy sets and systems to improve the behavior of heuristics is now widely recognized [10]. So, we decide to explore the use a fuzzy rule based manually generated in the Coordinator, to decide the orders to submit to the subordinated algorithms. We will show here how a very simple scheme allows us to obtain very good results over a test problem.

The paper is organized as follows: in Section 2 we describe other work in parallel metaheuristics, then in Section 3 we introduce our proposal. Implementation details, experiments and some preliminary results over the knapsack problem are shown in Section 6. Finally, Section 7 is devoted to the conclusions.

## 2 A Brief Description of Parallel Metaheuristics

In this section, we present some relevant concepts regarding parallel metaheuristics. Following the work of Crainic [3], we can recognize 3 strategies to parallelize metaheuristics.

The first one, or *Type 1 parallelism*, may be obtained by the concurrent execution of the operations or the concurrent evaluations of several moves making up an iteration of the search method. It's directly oriented to reduce the execution time of the method and it should be remarked that, when given the same number of iterations, the sequential and parallel implementations will lead to the same result.

The second strategy, or *Type 2 parallelism*, comes from the decomposition of the decision variables into disjoint subsets. The particular heuristic is applied to each variable in the subset and the other ones are considered fixed.

Finally, the third strategy or *Type 3 parallelism*, is made up of several concurrent searches in the solution space. Each concurrent thread may or may not execute the same method, they may start from the same or different initial solutions, etc. If the threads communicate during the search,

we call such strategy as *co-operative multi-thread* methods; if they communicate at the end of the run, we call them *independent search* methods.

We will focus on type 3 strategies which are often used to perform a greater exploration of the search space. Several studies have shown that multi-thread techniques lead to better solutions than the corresponding sequential counterparts, even when the running time available to each thread is lower than that of the sequential computation.

Studies have also shown that the combination of several threads, each one implementing a different strategy, increases the robustness of the global search relative to variations in the characteristics of problem instances.

It should be noted, that the use of classical performance measures to such parallel metaheuristics is somewhat problematic. For example, it is hard to eliminate or control the overlap between the search paths. Also, threads (usually) interact asynchronously, so computations can produce different outputs for the same input. So, the topic on how to compare this parallel strategies against their sequential counterparts, is an open and active field.

The interested reader is referred mainly to [3, 5] and the references therein for a recent review on parallel metaheuristics.

## 3 A Fuzzy Rule based Cooperative Multi Thread Strategy for Optimization

The main idea of our work can be summarized by imagining a committee in charge to solve some concrete problem. The committee is chaired by a *Director*, who knows all the general aspects of the concerned problem and the particular features of any worker of the team of experts.

Besides, the *Director* knows when an obtained solution can be considered acceptable, i.e., the *Director* knows when to stop.

Each worker is an expert problem solver using a specific approach, that may be shared by more than one worker. In this context each expert in the committee work alone, but all at the same

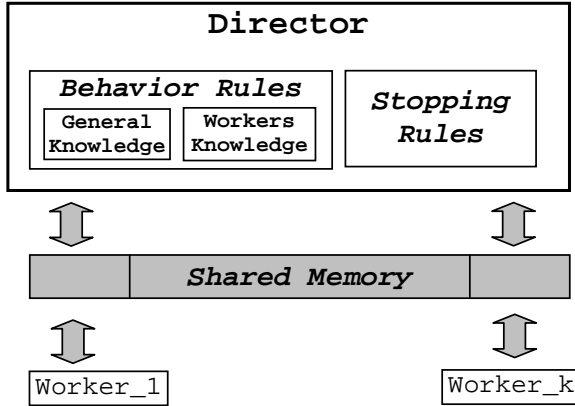


Figure 1: Scheme of the Proposal

time. The *Director* receive reports from the experts with the partial results being obtained.

Under this situation, we can model experts as optimization algorithms to obtain a co-operative multi-thread strategy, which is graphically represented in Fig. 1. Under the classification suggested by [3], our proposal is a type 3 strategy.

In this work, we model *Director's* role by means of a fuzzy rule based system, where the knowledge is represented by three groups of rules:

- *Global Rules*: representing the general knowledge of the *Director* about the problem (type, size, characteristics, ...), and possibly who are the best experts to solve it.
- *Worker's Rules*: representing the specific knowledge of the *Director* about the characteristics of the workers: knowledge about their behavior across the time, preferences, tuning, etc.
- *Stopping Rules*: representing the termination criteria that the *Director* uses. For example, they can be based in the cost of the current solution, in the elapsed time since the beginning, etc.

The fuzzy rule based is manually generated by the experts, trying to address essential principles:

1. If something runs well, why to change it?. But, if something doesn't work, change whatever it may be.

2. Get together with good people and you shall become like them.
3. Don't try to give grass to a tiger.
4. Good food doesn't change with a microgram more of salt.

The first principle govern a lot of algorithms and it has a very clear meaning. The second one govern the idea to share good solutions in order to concentrate the search around them. Third one advises about do not try to alter the original behavior of an algorithm supplying premature good solutions. Last principle is equivalent to the well-know fuzzy principle: "The very best solution is an enemy of good solutions". Sometimes, it is completely needless to jump from a good and feasible solution to the best one.

Now, having these ideas in mind, we can make a deeper description of the proposal with the help of the pseudocodes shown in Figures 2 and 3.

The first one is the pseudocode for *Director*. In a first step, the *Director* determines the initial behavior (set of parameters) for every algorithm or worker. Such behavior is passed to every algorithm and then, they are executed.

The communication between *Director* and workers is performed through a shared memory where: workers stores performance information and retrieves adaptation orders; and *Director* stores behaviors and retrieves performance measures used to adapt its internal fuzzy rule base.

As said before, workers execute asynchronously, reading and writing in the shared memory. The *Director* checks which worker provided new information and decides if its behavior needs to be adapted. If this is the case, it will use the fuzzy rule base to obtain a new behavior which will be stored in the shared memory.

When the stopping criteria are met, the *Director* stops the workers.

The operation of the workers is quite simple. They are running alone, alternating the read and write of the shared memory to obtain adaptation information from the *Director* or to inform their performance. Both operations are controlled by conditions

```

Procedure Director:
Begin
/* algi, a worker */
/* behaviori, set of parameters */
/* director, the director */
/* sharedMemi, shared memory for information
exchange between algi and director */
/* FRB, the fuzzy rule base of the director */
For i := 1 To k Do
  obtain behaviori for algi from FRB;
  setup algi with behaviori;
  algi starts to RUN;
Od
Repeat Until ( not stop criteria ) Do
  sharedMemi is updated by algi (asynchronously);
  For (every algi with new results) Do
    director decides if algi needs a change;
    IF (YES) Then
      obtain behaviori for algi from FRB;
      director stores behaviori in sharedMemi;
    Fi
  Od
Od
For i := 1 To k Do
  stop algi;
Od
End.

```

Figure 2: Scheme of *Director*

```

algi RUN:
Begin
Repeat Until ( stop = true ) Do
  check news from Director in sharedMemi;
  IF (new information available) Then
    update internal parameters;
  Fi
  ;
  /* here goes the particular scheme for algi */
  ;
  IF (communication conditions hold) Then
    store performance information in sharedMemi;
  Fi
End.

```

Figure 3: Scheme of algorithms

## 4 Implementation Details

When this paper is being written, a preliminary version of the scheme is fully developed. Each algorithm is a Fuzzy Adaptive Neighborhood Search (*FANS*) algorithm [2, 9], with different parameters. This implies that each algorithm shows a different behavior.

In this implementation, the parallelism is simulated by a random order of execution of the read and write instructions to the shared memory (which is implemented as a data structure)

As recognized in [5] regarding parallel strategies:

The most difficult aspect to be set up is the determination of the nature

of the information to be shared or exchanged to improve the search, without taking too much additional memory or time to be collected. .... This data can give a broader view of the search space and may be used to drive diversification and intensification procedures.

Because *FANS* is a local search based heuristic, and we are in a preliminary stage of development, we decided that the information sent by the *Director* to each algorithm is a new solution (a new point in the search space). When the algorithm receive it, it restart the search from that new point.

In the other way, each algorithm *alg<sub>i</sub>* send to the *Director*, their current solution, their cost and a time stamp. The *Director* records the information and calculates two values:

1. the improvement rate (with respect to the previous report of *alg<sub>i</sub>*)
2. the “badness” of the solution, in terms of a set of collected values from all the algorithms

The fuzzy rule base just contain one rule: *IF* the badness of the solution reported by *alg<sub>i</sub>* is *bad* and the improvement rate of *alg<sub>i</sub>* is *bad* THEN store in *sharedMem<sub>i</sub>* a new solution.

This new solution could be a randomly generated one, or could be the best one saw by the *Director*.

## 5 Experiments

In order to test the benefits of our proposal, we perform preliminary experiments over instances of the knapsack problem.

We defined 3 problem sizes,  $n = \{150, 300, 450\}$  and for each  $n$  we randomly generated 5 instances with a weak correlation between profits and weights. For each instance we calculated the Dantzig bound as the reference value to measure the effectiveness of the algorithm [7].

Then, for each one of the 15 instances, we run 30 times the scheme with and without the rule base activated. When the rule base is not activated, the information flow between *Director* and algorithms, is disabled.

Table 1: Average and Standard Deviation values for variables *Error* and *% Evals* over the 15 test instances.

FRB	<i>Error</i>		<i>% Evals</i>	
	<i>Avg</i>	<i>StdDev.</i>	<i>Avg</i>	<i>StdDev.</i>
<i>disabled</i>	4.66	0.37	57.19	27.24
<i>enabled</i>	4.01	0.59	79.95	18.53

Each run ended when  $100 \times n$  evaluations of the objective function were done. At that point, we calculate the error (with respect to the Dantzig bound) of the best solution found, (namely, *Error*) and the number of evaluations of the cost function used to achieve such error. This value is re-scaled to show the percentage of evaluations used over the evaluations available (*% Evals*).

## 6 Results

We analyzed the results from two points of view. First, considering the set of 15 instances in a global manner; and second, analyzing the results in terms of the size of the problem.

In Table 1 we present the results over the whole set of instances for the two variables of interest: *Error* and *% Evals*. In terms of the *Error*, it is clear that when the FRB is enabled, the strategy achieved significantly lower average values than when it is disabled. The standard deviation is a bit higher for the enabled case.

In terms of *% Evals*, the average is higher when FRB is enabled. This fact may be considered not good at a first sight. However, in our opinion two elements should be analyzed carefully: first, the number of evaluations is higher but the error is lower; so there is a trade off that the user needs to define. Second, the higher values also implies that almost the whole run was profitable. If we consider the case where *FRB* is disabled, the best values were achieved using around a 57% of the evaluations available, meaning in turn, that 43% of the evaluations were wasted.

The results in terms of the size of the problem are shown in Fig. 4. The upper graph shows the average error as a function of the size. It is easily

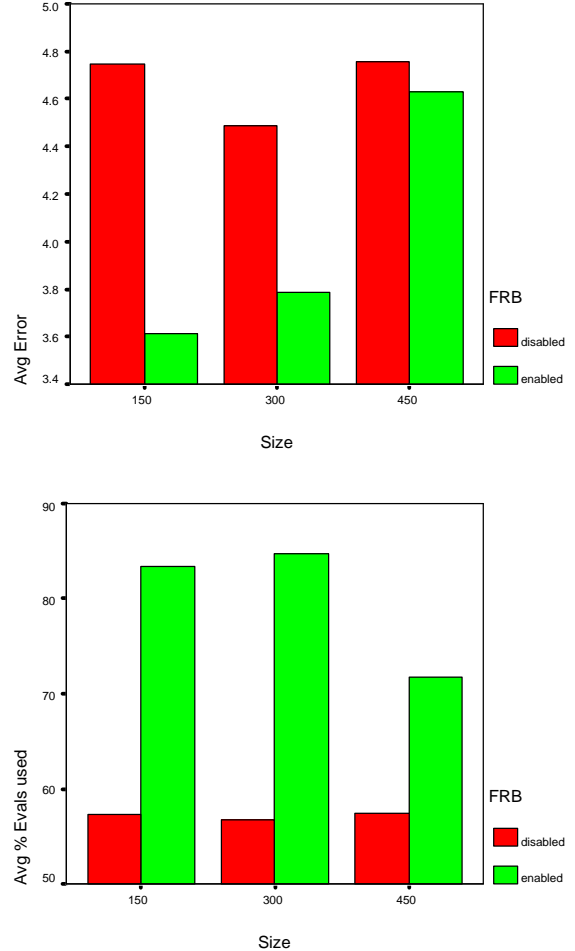


Figure 4: Bars graph showing: the average error as a function of the size for the FRB enabled /disabled (upper graph); and the average percentage of the evaluations used to achieve the best error.

seen that for each size, the use of the FRB allows to obtain lower error values. Also, as the size increase, the average error also increases. When the FRB is disabled, there is no clear tendency for the error.

The lower graph in Fig. 4, shows the average of *% Evals* for each size considered. When FRB is enabled, no clear behavior is detected; while, in the case of disabling the FRB, the values are almost the same for every size. Also, the enabled case presents higher values than the disabled one.

## 7 Conclusions and Further Work

We presented a co-operative multi-thread meta-heuristic, where each thread is modelled as an optimization algorithm, executed asynchronously, and controlled by a Coordinator process. A relevant point is the use of a fuzzy rule base embedded in the Coordinator to control and modify the behavior of the optimization threads.

Preliminary results over the knapsack problem show the benefits of the proposal and encourage us to perform experiments over a larger set of instances of several optimization problems.

### Acknowledgements

This work is partially supported by Project TIC-2002-04242-C03-02. Authors thank the reviewers for their very useful comments that helped to improve the quality of this paper.

## References

- [1] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. *Complexity and Approximation*. Springer Verlag, 1999.
- [2] A. Blanco, D. Pelta, and J. Verdegay. A fuzzy valuation-based local search framework for combinatorial problems. *Journal of Fuzzy Optimization and Decision Making*, 1(2):177–193, 2002.
- [3] T. G. Crainic and M. Toulouse. *Parallel Strategies for Metaheuristics*, volume Handbook of Metaheuristics, pages 475 – 513. Kluwer Academic Publisher, 2003.
- [4] P. Crescenzi and V. Kann. A compendium of np optimization problems. Technical report, <http://www.nada.kth.se/theory/problemlist.html>.
- [5] V.-D. Cung, S. L. Martins, C. Ribeiro, and C. Roucairol. *Strategies for the Parallel Implementation of Metaheuristics*, volume Essays and Surveys in Metaheuristics, pages 263–308. Kluwer Academic Publisher, 2001.
- [6] F. Glover and G. A. Kochenberger, editors. *Handbook of Metaheuristics*. Kluwer Academic, 2003.
- [7] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, 1990.
- [8] P. Pardalos and M. Resende, editors. *Handbook of Applied Optimization*. Oxford University Press, 2002.
- [9] D. Pelta, A. Blanco, and J. L. Verdegay. Applying a fuzzy sets-based heuristic for the protein structure prediction problem. *International Journal of Intelligent Systems*, 17(7):629–643, 2002.
- [10] J. L. Verdegay, editor. *Fuzzy Sets based Heuristics for Optimization*. Studies in Fuzziness and Soft Computing. Physica-Verlag, 2003.