Using Machine Learning in a Cooperative Hybrid Parallel Strategy of Metaheuristics

J.M. Cadenas [a,*] M.C. Garrido [a] E. Muñoz [a]

[a] *Dpto. Ingeniería de la Información y las Comunicaciones. Facultad de Informática. Universidad de Murcia. Spain.*

Abstract

This paper proposes the construction of a centralized hybrid metaheuristic cooperative strategy to solve optimization problems. Knowledge (intelligence) is bestowed on the coordinator to improve performance. This knowledge is incorporated through a set of rules and models obtained from a knowledge extraction process applied to the records of the individual metaheuristics. The effectiveness of the approach is tested in several computational experiments in which we compare the results obtained by the individual metaheuristics, several non cooperative and cooperative strategies and the strategy proposed here. The superiority of the proposed strategy is manifest in all cases.

Key words: Soft Computing, Optimization, Multi-agent Systems, Cooperative Metaheuristic Systems, Data Mining.

1 Introduction

In optimization problems we are, in general, seeking the best solution (or a sufficiently good one), from many alternatives, to a problem. Optimization problems crop up everywhere in our daily lives. We all constantly solve optimization problems like finding the shortest route from our house to our workplace given the traffic conditions, or organizing our agenda. And we efficiently find solutions to these daily problems. If we are able to do so, it is because the problems are still manageable, i.e. they are of a small enough dimension to be processed. However, these problems may grow to much larger scales. In order

* Corresponding author. Tel.: +34 968 364847; fax: +34 968 364151.
Email addresses: jcadenas@um.es (J.M. Cadenas), carmengarrido@um.es (M.C. Garrido), enriquemuba@dif.um.es (E. Muñoz).

to solve them, we have various algorithms, and among these one of the most important are metaheuristics. Metaheuristics can be defined as, [17]: *"Intelligent strategies to design or improve very general, high performance heuristic procedures"*.

Metaheuristics offer effective strategies to find approximate solutions to optimization problems. However, when we use them we can find the algorithm selection problem [22]. This problem tries to decide which algorithm has to be selected to solve a given instance trying to maximize a measure of performance. The most common approach to solve it is to measure the performance of a set of algorithms over a set of instances and use the best performer. This system, known as "winner-take-all" has spurred the development and refining of some metaheuristics, but has hindered the development of others, which, while not in overall terms competitive, may offer an excellent performance for particular instances of the problem. In any case, [13] states that "automatic algorithm selection problem is undecidable", but this does not deny the possibility of inducing an intelligent system that, using the theoretical and experimental evidences obtained by the resolution of different instances with different algorithms, suggests or directly applies an algorithm, or a set of algorithms.

This intelligent system has to be flexible, so that it can adapt better to changing characteristics of the problems. As Hybrid systems provide flexible mechanisms to solve complex problems, which are very difficult to solve using less tolerant approaches, they are an interesting way to tackle algorithm selection problem. Moreover we can expect that an intelligent combination of different metaheuristics will provide more efficient approaches and more flexibility, obtaining robustness by the use of different metaheuristics that cooperate to give high quality solutions for different classes of instance, [4,5,2]. But the intelligent combination of different metaheuristics requires sacrificing precision in order to gain more tolerance. This is what constitutes the basis of the methodology proposed by Soft Computing [28]. The use of a set of metaheuristics along with the methodologies provided by Soft Computing to build hybrid systems will give us the reasoning mechanisms and the search methods which will allow us to combine the knowledge with the experimental data to obtain new, flexible computational tools. With these tools we can solve complex problems that would be very difficult to solve with less tolerant approaches.

2 Related Work

Several studies have shown that metaheuristics are successful tools for providing reasonably good solutions (in some cases excellent) using a moderate number of resources. Recent literature, e.g. [9,12,17,19,20], reveals a wide vari-

ety of problems and methods that appear within the overall topic of heuristic optimization. The main trend in this field is to develop new strategies that obtain better solutions for given problems. However a very interesting line of development is to combine existing metaheuristics to obtain a hybrid strategy in which they cooperate parallely to solve a problem. This approach can be interesting since robust strategies can be obtained which offer high quality solutions in spite of the changes in the characteristics of the instances.

Metaheuristics are commonly classified, [7], as population based and trajectory based. Population based metaheuristics are those that produce an evolution over the search space of sets of solutions, usually called populations, and try to approximate the optimal solution with its elements. The main characteristic of these metaheuristics consists on the interaction between population members in contrast with searches which are guided by the information of individual solutions. On the other hand, trajectory based metaheuristics, are those that establish strategies for searching the solution space of the problem, performing iteratively changes on the initial solution. These two families have complementary properties: population based metaheuristics allow a better exploration of the search space, while trajectory based metaheuristics allow a better search intensification of the most promising areas. Metaheuristics hybridization tries to make the most of the different advantages of each kind of metaheuristic to get more robust algorithms which obtain better solutions.

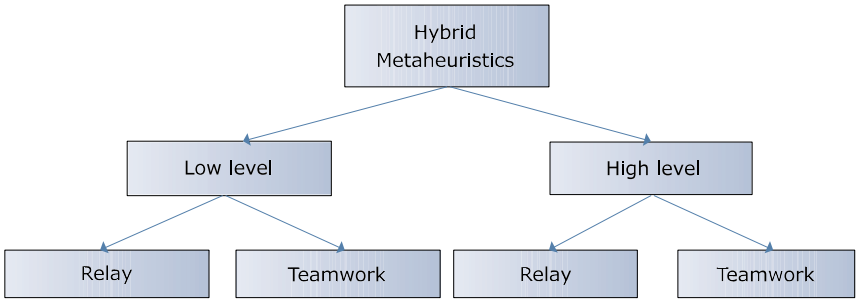

Fig. 1. Hybrid metaheuristics taxonomy.

In [25], E. G. Talbi describes a taxonomy of hybrid metaheuristics, and includes two levels and two modes of hybridization (fig. 1): Low and high levels, and Relay and Teamwork modes. Low level hybridization refers to functional composition of a unique optimization method. In the latter, one function of a metaheuristic is replaced by another metaheuristic. In contrast, high level hybridization techniques are self-containing and there is no direct relation with the internal working of a metaheuristic. On the other hand, in relay hybridization, a set of metaheuristics is applied in a pipeline way. The output of one metaheuristic is the input of the next one. Conversely, in teamwork hybridization, a set of cooperative agents carries out the search on a solution space. In this taxonomy we can find four classes:

- LRH (Low-level Relay Hybrid): This class represents those algorithms where

a given metaheuristic is embedded into a trajectory based metaheuristic.

- LTH (Low-level Teamwork): This class includes algorithms where a metaheuristic is embedded into a population based metaheuristic.
- HRH (High-level Relay Hybrid): In this class are included those hybrids where metaheuristics are executed sequentially.
- HTH (High-level Teamwork Hybrid) This class includes those strategies where different metaheuristics carry out a parallel cooperative search. These metaheuristics may be the same or different.

In this paper we present an HTH hybrid metaheuristic. This strategy can also be included on the field of parallel metaheuristics. This field appears as a way of improving acceleration factor in the search of solutions, although later it has been noted [8] that robust strategies can be obtained which can outperform the sequential algorithms that compose them.

In [10] a taxonomy of parallel metaheuristics is presented, with attention to the parallelism source used:

- Type 1 parallelism or low level. This source of parallelism is usually found within an iteration of the heuristic method. The limited functional or data parallelism of a move evaluation is exploited or moves are evaluated in parallel. This strategy, also called low-level parallelism, is quite straightforward and aims solely to speed up computations, without any attempt at achieving a better exploration (except when the same total wall-clock time required by the sequential method is allowed to the parallel process) or higher quality solutions.
- Type 2 parallelism or Domain Decomposition: This approach obtains parallelism by partitioning the set of decision variables. The partitioning reduces the size of the solution space, but it needs to be repeated to allow the exploration of the complete solution space. Obviously, the set of visited solutions using this parallel implementation is different from that of the sequential implementation of the same heuristic method.
- Type 3 parallelism or multi-search: Parallelism is obtained from multiple concurrent explorations of the solution space. Each parallel thread can execute the same metaheuristic or different ones, it can start from the same or different solutions etc. Search threads can communicate during the search (cooperative multi-search) or only at the end in order to identify the best solution (independent search).

The strategy proposed is included in type 3, specifically in cooperative strategies. This strategy is frequently used to perform a better exploration of a search space. Several studies [9,11,27] have shown that multi-thread techniques provide better solutions than their sequential components, even when the available execution time for each thread is smaller than the one for se-

quential strategies. These studies also have shown that the combination of different threads, each using a different strategy, increases robustness of the global search with respect to changes in the instances of the problem. They also indicate that while independent search strategies are easy to implement, and obtain good results, they can be improved by using a cooperative strategy. However, it has been shown [11] that cooperative methods with an unrestricted access to shared information can experiment premature convergence problems. This seems to be due to the stabilization of the shared information produced as a result of the intense exchange of the best solutions. It would, therefore, be interesting to find a way of controlling this information exchange. Thus, in [20] a cooperative strategy based on memory is proposed to control this effect. Here a coordinating agent, modeled by a set of fuzzy rules defined by the user, monitors a set of solver agents and sends orders to them about how they have to continue, with each agent implementing the same metaheuristic, FANS [1].

As we said before, it seems that a cooperative strategy based on a unique metaheuristic does not cover all the possibilities and it is recommended to combine different ones. Good example of this kind of strategy are [16] and the A-teams proposed by Souza and Talukdar [23] where a set of heuristics communicate with each other by exchanging solutions through a shared memory. A new field of research appears here with questions as: What will be the role of each metaheuristic? What cooperation mechanisms should be used? In this paper we try to answer this questions and propose a structure similar to that proposed in [20] where a coordinator modeled by a set of fuzzy rules gets information about the performance of the different metaheuristics and sends orders to them. The main differences are that it combines a set of different metaheuristics and that the rules are obtained as the result of an automated knowledge extraction process.

This last point, the combination of knowledge extraction with the use of heuristics and metaheuristics, has also become an interesting trend in heuristic optimization. In [14] a decision tree is obtained with the aim of selecting the algorithm to be applied to solve a given instance of a problem. On the other hand, within the framework of hyper-heuristics, in [3] a hyper-heuristic uses case-based reasoning to decide which heuristic has to be used at every moment, and in [26] associative classification is used instead of case-based reasoning. Our idea is similar, as we try to ascertain which metaheuristic is best for each kind of instance, but since we execute every metaheuristic parallely, we want to establish some kind of priority order.

In this paper we present a cooperative strategy of metaheuristics modeled using a knowledge extraction process. To do this, in section 3 we present the architecture of the strategy, which is based on a multi-agent strategy, and the design process to obtain it, which is based on a knowledge extraction process. After defining the strategy and its design process, we illustrate it in section

4by showing the modeling of a specific system to solve a concrete problem - knapsack problem. We include a detailed view of each phase performed. After that, in section 5 we carry out some tests in order to check the efficiency of the strategy obtained. Finally in section 6 we offer the conclusions we have drawn and the work we wish to develop.

3 Scheme of the strategy and its design process

The aim is to create a hybrid cooperative strategy/system of metaheuristics which intelligently combines different metaheuristics, which provides more efficient approaches and more flexibility when we face optimization problems and which obtains robustness through the use of different metaheuristics that cooperate to give high quality solutions for different types of instance, [4,5]. In the following sections we will discuss the system architecture and the proposed process for its design and construction.

3.1 The system architecture

A multi-agent system can be used to perform the cooperative execution of the hybrid system. In this system each agent implements one metaheuristic and tries to solve the problem while it coordinates with the other ones. There is also a coordinating agent in charge of controlling and modifying the behavior of the metaheuristics. Thus, we have a centralized model. The coordinating agent will therefore have two fundamental tasks: to collect the information on the performance of each of the solver agents and to send them orders which will affect their search behavior, [20]. This idea can be seen on fig. 2.

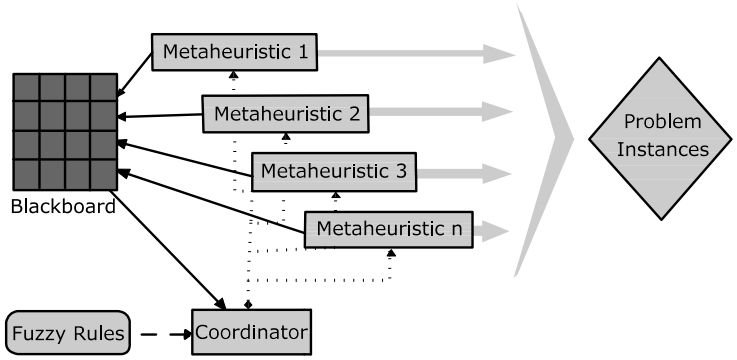

Fig. 2. Multi-agent metaheuristic system

The communication between the agents in this system will be made using a blackboard architecture, a structure of data usually employed as a general communication mechanism. In our proposal each agent has an assigned space

on the blackboard where it periodically leaves information about the solution it has found to a problem. The coordinator observes this information and directs the search of each agent.

Note that we use a blackboard model which is "adapted" to the objectives of the system, since it is a blackboard in which the solver agents, the metaheuristics, do not perform any process involving taking and receiving information. They are only capacitated to leave information on the blackboard. This means that the metaheuristics are unaware of the existence of other metaheuristics and cannot, therefore, take information left on the blackboard by the others. It is the coordinator who will take the information from the blackboard and then will give the orders directly to the metaheuristics, i.e. not through the blackboard.

3.2 Coordinator intelligence

Once we have decided on the communication model, the problem that arises immediately is how to describe the coordinator in such a way that it can modify the behavior of the metaheuristics efficiently and soften the problems shown before related to the behavior of the metaheuristics when taken individually.

We propose to give "intelligence" to the coordinator by means of a set of fuzzy rules since they allow data to be represented in a very similar way to human reasoning, and so make the system more comprehensible and at the same time will allow expert knowledge to be easily incorporated into the system as ad hoc rules. These rules will give intelligence to the coordinator and this will come about as a result of the methodology proposed in [4,5,2]. In this methodology, we first apply a supervised knowledge extraction process to fill in a template of fuzzy rules. Once this process has been applied, the rules are implemented and we obtain a completely operative system, without needing to apply the knowledge extraction process again. The knowledge extraction process used to fill in the template will be seen in subsection 3.3.

The template of fuzzy rules that models the system coordinator tries to control the following behaviors:

- When a metaheuristic has to be reinitiated with the solution which appears to be the best at a given moment. item When a metaheuristic should be reinitiated with a different set of parameters.

This template is composed of the following two rules:

- IF [MH IS $theWorst$] AND [($wm_1 * d_1$ OR ... OR $wm_n * d_n$) IS $enough$] THEN change the current solution of MH.

- IF $[(wm_1 * d_1$ OR ... OR $wm_n * d_n)$ IS $High$ AND ($time$ IS $THigh$ OR $TVeryHigh)]$ THEN $changeParameterValues$ of MH.

where:

- n is the number of metaheuristics.
- each one of the rules is evaluated once for each one of the metaheuristics that conform the system.
- MH is the metaheuristic being evaluated by the rule.
- $theWorst$ evaluates if the metaheuristic being studied now is having the worst performance according to any previously defined measure.
- $d_i = (perf_i - perf_{MH})/maximum(perf_i, perf_{MH})$, where $perf$ is a previously defined measure of performance.
- $wm_i \in [0, 1]$ where $\sum_{i=1}^{n} wm_i = 1$ and wm_i represents the weight of metaheuristic i (importance of metaheuristic i for solving the current instance).
- $Enough$, $High$, $THigh$ and $TVeryHigh$ are fuzzy sets with trapezoidal membership function with support contained in $[0, 1]$ defined by a cuadruplet (a,b,c,d). The mathematical representation is shown in equation (1):

$$\mu(x, a, b, c, d) = \begin{cases} 0 & x \leq a \ or \ x \geq d \\ (x - a)/(b - a) & x \in (a, b] \\ (d - x)/(d - c) & x \in [c, d) \\ 1 & x \in [b, c] \end{cases} \quad (1)$$

- $ChangeParameterValues$ is a function that changes the values of the parameters of a metaheuristic. To do this, a model is used which gives an order to the combinations of parameter values for the instance being solved. In this way the first set of parameter values used by the system is the first of this order and when a change of parameter values is needed the next set, in order, is used and so on.

The first of these rules tries to indicate how the position in the search space of the metaheuristic with the worst behavior can be changed for a position near that of another metaheuristic with a better behavior.

On the other hand, the second rule shows how the values of the parameters of the different metaheuristics have to be modified. Thus, if a metaheuristic is obtaining solutions with a worse performance than the rest, and it has been a long time since its parameters have been changed, then we can change its parameters for a new set using a new set of suitable values.

Finally, in order to obtain the wm_i, $\forall i$, that define the rules and the set of possible combinations of parameter values available for the function $changeParameterValues$, we use some decision trees. These trees receive a description or instance that has to be solved and output the wm_i and the order among the combination of

parameter values for each metaheuristic. Thus, we have a tree that assigns the weights to each metaheuristic and as many trees as metaheuristics to obtain the order among the combination of parameter values to be used, and those that replace them. An idea of how the trees give the configuration of the set of fuzzy rules can be seen on fig. 3.

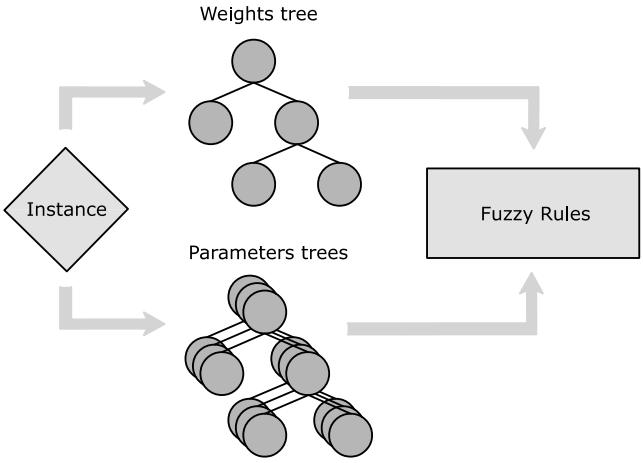

Fig. 3. Obtaining of weights and parameters values for metaheuristics.

To obtain these tree models we use the knowledge extraction process defined below. Note that this is a supervised process and previous to system operation.

3.2.1 Implementation issues

After defining the rules conceptually, some implementation issues need to be explained, for example the firing of rules or the interchange of solutions.

To fire the fuzzy rules we propose the use of $\alpha - cuts$, only firing those rules whose activation degree exceeds the $\alpha - cut$ defined. In the event that more than one rule is activated then we apply all of them.

In order to change the solution of the worst metaheuristic, we can take into account the following situations, noting that if the rule has been fired due to more than one of the antecedents, we consider the solutions of all the firers and define MH_{sender} as the metaheuristic that sends solutions and $MH_{receiver}$ as the metaheuristic that receives solutions:

- The $MH_{receiver}$ is based on trajectories. A solution "near" to the best solution obtained from among the metaheuristics that have fired the rule is sent to it. "Near" means at a number of random steps equal to $1/(2{*}wm_{MH_{sender}})$.
- $MH_{receiver}$ is based on populations. There are different options:
  - ◇ MH_{sender} is trajectory based. A proportion of the worst individuals of $MH_{receiver}$ equal to the $wm_{MH_{sender}}$ is substituted by a set of solutions consisting of solutions near MH_{sender} best solution.

- ◇ MH_{sender} is population based. A proportion of the worst individuals of $MH_{receiver}$ equal to the $wm_{MH_{sender}}$ is substituted by a set of solutions consisting of the best members of the population of MH_{sender}.
- ◇ Different metaheuristics have to send solutions. A proportion of the worst individuals of $MH_{receiver}$ equal to the sum of the wm_i of the senders is substituted by a set of solutions where each metaheuristic chooses its solutions as said before.

Other aspects that need to be defined are the fuzzy sets or the actual implementation of the rules. These issues, which are dependent on the problem or the platform choice, will be defined in the following sections.

3.3 Acquiring knowledge for the coordinator: The Knowledge Extraction Process

In order to acquire the knowledge needed by the coordinator of the cooperative system of metaheuristics we propose the following process: we compile information about the application of different metaheuristics to different instances of the problem that we want to treat. After that we apply a data mining technique to this information in order to find models that represent the behavior of the metaheuristics. This process can be seen in figure 4, and will be explained in the following sections.

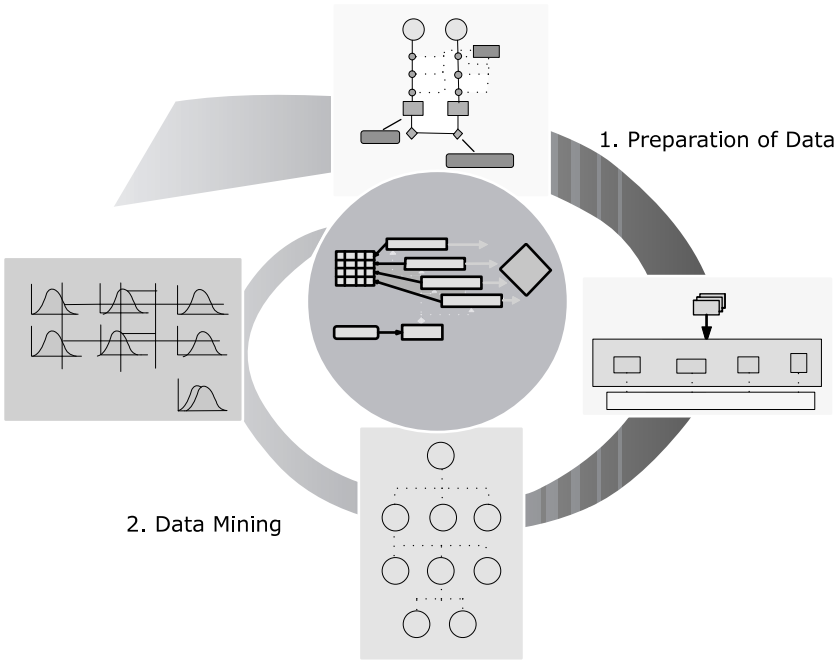

Fig. 4. Knowledge extraction process

3.3.1 Preparation of data

During the data preparation phase we are seeking a database which will hold useful information to which we can apply the necessary data mining techniques to obtain a model. The first step is to gather the information. Our information sources come from repeated applications (with different parameter values) of different metaheuristics to a set of instances of a problem (training instances). Certain data of interest are extracted from these executions, such as:

- A description of the specific instance of a problem.
- The values of the parameters used in the execution of each metaheuristic.
- Information obtained during the execution, e.g. intermediate solutions.
- The final solution and the error that is produced.

With this information we generate the structure of what we will call an example of the database we wish to build, which is made up of a vector which is defined according to a series of attributes.

In fig. 5 we can see a conceptual image of the information extracted from the executions of the different metaheuristics. In this database each line contains a description of the instance being solved and the aggregation of the data generated by an execution of each metaheuristic, so we have a combination of parameter values for each metaheuristic and information about the different solutions that it has obtained. The user can choose how to combine this information, for example including all the information about an instance in just one line or choosing the best combination of values of the parameters from each metaheuristic.

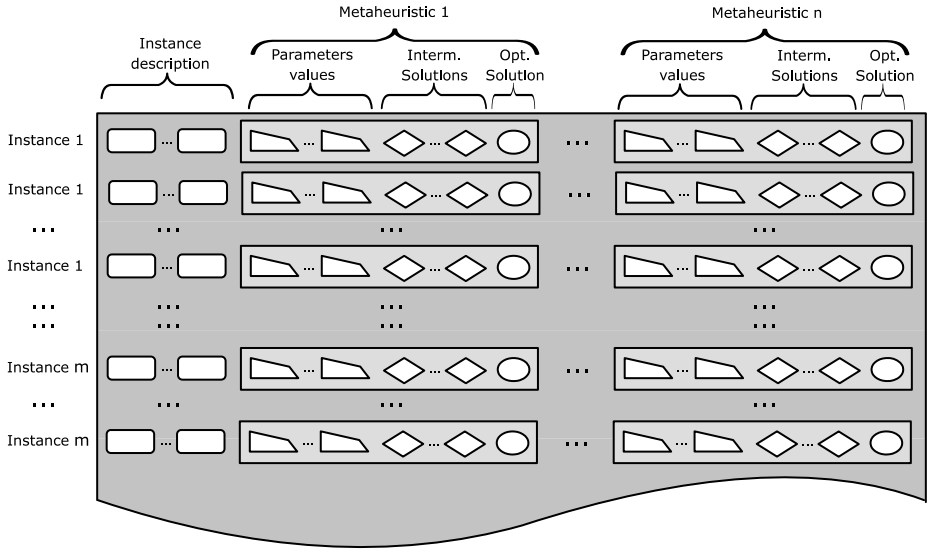

Fig. 5. Database with information about the metaheuristics

3.3.2 Data mining

In this phase we use data mining techniques to obtain the models that will fill the template of fuzzy rules. As we said before, we decided to use decision trees, and this was done for two main reasons. First, they can easily handle imperfect information, and second they are reasonably interpretable.

Decision trees are among the most commonly used methods of inductive learning. As a means of representing knowledge, decision trees stand out on account of their simplicity, their readability and their interpretability. Many learning algorithms exist for constructing decision trees and we have chosen a fuzzy decision tree (FDT) [15].

FDTs mix fuzzy representation and its approximate reasoning capacities with the generation of decision trees. Furthermore FDTs allow us to work with nominal and continuous attributes where the domains of the continuous attributes are partitioned in a set of linguistic labels.

FDTs have another advantage - their outputs are degrees of membership, so they are ideal for obtaining the weights of each metaheuristic and an order among the different combinations of parameters. It is also interesting how they can be modified to obtain different outputs by changing how they combine fuzzy inferences. Thus, we can obtain different behaviors.

An FDT therefore has to be used to obtain the models proposed in subsection 3.2, i.e. a model that assigns weights to the different metaheuristics according to the instance that has to be solved and one model for each metaheuristic that obtains the combinations of values of the parameters, again according to the instance that has to be solved.

After that we have to test the efficiency of the models utilized to fill the template of fuzzy rules. Once the template has been filled, and therefore the fuzzy rules are completely defined, we execute the cooperative system to solve a set of test instances (these test instances are different from the training instances) of the problem and we verify the efficiency of the coordinator with respect to the goodness of the solutions found.

4 Constructing an intelligent cooperative system (KEPS)

In order to test the feasibility of the system we build one, denoted KEPS, for a concrete problem. To undertake the construction of the system we shall decide the problem to treat and the set of metaheuristics that will compose the system, we also must apply the knowledge extraction process to give intelligence

to the coordinator.

In this section we show how the process previously described is applied to the knapsack problem and in this way we present the different decisions taken and the results obtained in the different phases of the knowledge extraction process.

4.1 The {0,1}-Knapsack Problem

The problem that we treat is the well known {0,1}-knapsack problem. This problem can be formally defined as follows: Given a set of items, each with a cost and a benefit, determine a subset such that the total cost is less than a given limit and the total benefit is as large as possible. As a mathematical formalization, it is:

$$Max \sum_{i=1}^{n} p_i \times x_i$$

$$s.t.$$

$$\sum_{i=1}^{n} w_i \times x_i \leq C$$

$$x_i \in \{0,1\}\, i = 1, ..., n$$

where

- n is the number of items.
- x_i indicates whether item i is included in the knapsack or not.
- p_i is the benefit associated to item i.
- $w_i \in [0, \ldots, r]$ is the weight of item i.
- C is the capacity of the knapsack.

It is assumed that $w_i < C,\ \forall i$ and $\sum_{i=1}^{n} w_i > C$.

In the {0,1}-knapsack problem we can construct instances of varying hardness by modifying both the size of the instances and the type of correlation between weights and profits. Although this problem is considered to be one of the "easiest" NP-Hard problems, "hard to solve" instances can be built which are considered a challenge for most of the exact algorithms. [21] studies where they can be found.

In the construction and evaluation of the system we shall deal with instances with different sizes, being the size the number of items that can be chosen, and with different correlation between weights and profits, where we can find instances which belong to the following five categories:

- Spanner: These instances are constructed in such a way that all their items are multiples of a small set of items called key. That key can be generated

using any distribution. In this case we consider three distributions:
  - ◇ Uncorrelated: $w_i = U(1, R), p_i = U(1, R)$,
  - ◇ Weakly correlated: $w_i = U(1, R), p_i = U(w_i - \frac{R}{10}, w_i + \frac{R}{10})$ with $p_i \geq 1$,
  - ◇ Strongly correlated: $w_i = U(1, R), p_i = w_i + \frac{R}{10}$.
- Profit ceiling: In these instances all the benefits are multiples of a given parameter d, in our case $d = 3$: $w_i = U(1, R), p_i = d\lceil w_i/d \rceil$.
- Circle: These instances are generated in such a way that the benefits are a function of the weights, with its graph having an elliptic representation ($d = \frac{2}{3}$): $w_i = U(1, R), p_i = d\sqrt{(4R^2 - (w - 2R)^2}$.

The weights w_i are uniformly distributed in a given interval with the data range R. H instances are randomly generated. The capacity of each instance h, with h = 1. . . ,H is calculated as $c = \frac{h}{1+H} \sum_{i=1}^{n} w_i$. The optimum of each instance is also provided.

4.2 Metaheuristics to cooperate

In order to shape the cooperative metaheuristic system we use both trajectory based and population based metaheuristics, since we wanted to study how the coordinator could manipulate each type. For that reason we selected a Tabu Search and a Simulated Annealing as trajectory based metaheuristics, and a Genetic algorithm as a population based one, [12]. These three metaheuristics followed a standard implementation without any special preparation for solving the {0,1}-knapsack problem. They were guided only by the value of the objective function.

4.3 Acquiring the intelligence

4.3.1 Data Preparation

To apply the Knowledge Extraction Process the first thing to be done is to gather information obtained from the independent application of the solver agents to solve the set of training instances of the problem. Therefore, we solved a big dataset of training instances with $R = 1000$, 4 problem sizes, $n = (500, 1000, 1500, 2000)$, and for each n and each type of instance series of $H = 100$ was randomly generated for a total of $4 \times 5 \times 100 = 2000$.

Each one of these instances was solved with each one of the metaheuristics that we wanted to incorporate to the system - the Genetic Algorithm, the Tabu Search and the Simulated Annealing. Moreover each metaheuristic was executed several times with the same instance using different configurations of parameters values. The Genetic Algorithm used eight combinations of param-

eter values, which controlled mutation and cross probability and number of individuals. The Tabu Search used nine combinations, which controlled tabu list length and long-term memory, and finally Simulated Annealing used eight combinations, which controlled the cooling schedule and the percentage of neighbors to be considered.

From these executions we extracted the information proposed in 3.3.1. In that way we obtained a first database that needed a preprocess before the data mining phase could be applied to obtain the models proposed in 3.2. We chose, from each instance, those executions of each metaheuristic which obtained the best result and these were fused in a vector (see Table 1). After that, we applied a preprocess, adding some relevant attributes and eliminating other superfluous ones. Thus, we eliminated all attributes concerning measures of performance and added a new one that summarizes their information, indicating which algorithm obtained the better performance for each instance. Similarly, we eliminated attributes concerning parameters and added a new one for each metaheuristic that indicates which combination of parameters obtained the best performance for each instance. Finally an example of the database was composed of the following attributes:

- A description of the instance of the problem, including: the number of objects available; the maximum, minimum and medium proportion of $w_i/capacity$ of the items available ($max(w_i/cap)$, $min(w_i/cap)$ and $med(w_i/cap)$); the maximum, minimum, and medium proportion of p_i/w_i of the items available ($max(p_i/w_i)$, $min(p_i/w_i)$ and $med(p_i/w_i)$), the standard deviation of p_i/w_i ($\sigma(p_i/w_i)$), the number of different values of p_i/w_i ($nSlopes$) and three estimations of instance type distinguishing between spanner, profit ceiling and circle, (*IsSpan*, *IsPCeil* and *IsCircle*).
- The configuration of the parameters of each metaheuristic which gave the best result (*BestParGen*, *BestParTab* and *BestParAnn*).
- The metaheuristic that obtained the best result, *BestMetah*.

All the domains of these attributes were partitioned into linguistic labels in order to facilitate the interpretation of the rules we obtain and to take advantage of the possibility provided by the FDT, [15], of working with this type of information. Fuzzy sets with a trapezoidal membership function were used for the partitions and were obtained using the tool given in the FDT.

4.3.2 Data Mining

In this phase we apply the FDT to the databases obtained in the previous phase in order to fill the template of the previously defined fuzzy rules. To do this, we need a tree that will provide the weights for the different metaheuristics, and another tree for each metaheuristic, which will provide the

Table 1
Part of the database

Instance description

Size	$max(w_i/cap)$	$min(w_i/cap)$	$med(w_i/cap)$
2000	2,19E-03	8,00E-05	1,56E-03
2000	2,24E-03	8,10E-05	1,58E-03
...	...	...	...

$max(p_i/w_i)$	$min(p_i/w_i)$	$med(p_i/w_i)$	$\sigma(p_i/w_i)$
... 8,67E-01	2,38E-02	5,46E-01	1,38E-01
8,67E-01	2,38E-02	5,43E-01	1,38E-01
...	...	...	...

nSlopes	*IsSpan*	*IsPCeil*	*IsCircle*
869	false	false	true
856	false	false	true
...	...	...	...

Metaheuristics information

BestParGen	*BestParTab*	*BestParAnn*	*BestMetah*
n-700-mp-0.001-cp-0.4	cd-1.5-tlt-100	pn-0.9-tf-E	genetic
n-300-mp-0.01-cp-0.4	cd-0.5-tlt-10	pn-1.0-tf-E	annealing
...	...	...	...

parameters to be used and those that will replace them.

In order to obtain the fuzzy tree of weights we applied the FDT to a subset of the database composed of the description of the instance and the algorithm which obtained the best performance. The tree obtained receives as an input the instance to be solved and gives as an output the weights that have to be applied to each metaheuristic. To do this, we make use of the fuzzy output of the tree, which returns the membership degree to each class, and assign weights according to the same.

On the other hand, to obtain the trees that will suggest which parameters values have to be used, we extracted a database for each metaheuristic composed of the description of the instance and a class describing the best combination of parameters for that instance. The output of these trees will give us a way of defining an order among the combinations of parameters - the first to be chosen being that which obtained a greater degree of membership. We decided to infer the combinations of parameters instead of a value for each parameter because we believe that the combinations of parameters are more important than parameters taken individually, and inferring them thus will improve the performance of the system. This belief was reinforced by some preliminary tests.

In fig. 6 we can see an example of the trees obtained where we present the tree that assigns weights to each metaheuristic. Here "ns" stands for nSlopes, "minW/C" for the minimum w_i/C, "medW/C" for the medium m_i/C, "sd" for the standard deviation of p_i/w_i and "minP/W" for the minimum p_i/w_i, each one of these attributes is divided in a series of linguistic values obtained automatically. Finally each of the leaves indicates the weight assigned by the tree to the Genetic Algorithm (ga), the Tabu Search (ts) and the Simulated Annealing (sa).

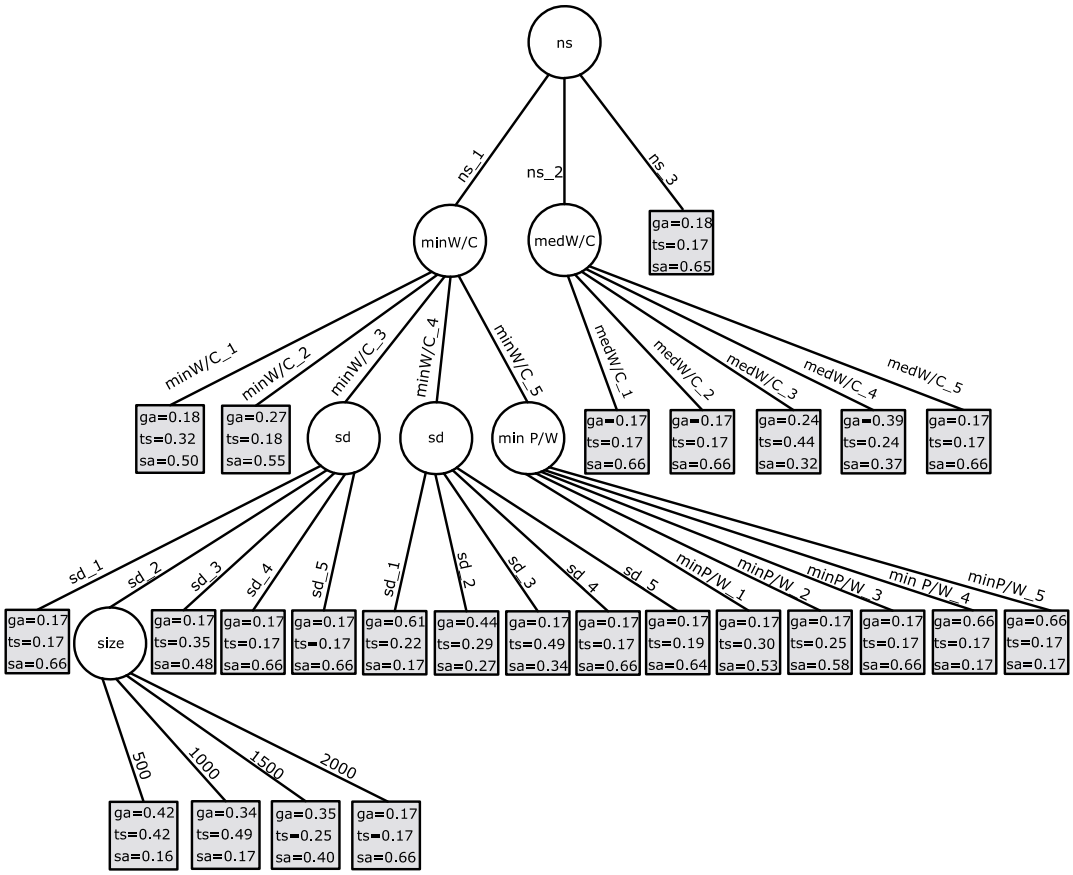

Fig. 6. An example of the trees obtained.

5 Computational experiments

In this section we perform some tests to assess the validity of the system obtained, denoted KEPS.

Initially, the performance of the individual metaheuristics is checked compared to a non cooperative parallel system in order to test the statement "which metaheuristic is best for each kind of instance". The non cooperative parallel system here consists of choosing the best solution obtained for each instance.

Subsequently, tests are made to check the efficacy/efficiency of the cooperative parallel systems and finally the usefulness of the incorporation of knowledge in these systems.

In the following subsections we will explain the configuration of the tests and the results obtained.

5.1 Configuration of the KEPS cooperative system

To carry out the tests we used a database of instances composed of 100 test instances (different from training instances) in which both size (number of objects) and the way they were generated were changed so that they represented all kinds of instances.

Every system and metaheuristic was programmed using Java. In order to obtain the fuzzy engine used to model the coordinator we used the XFuzzy 3.0 tool [18]. With this tool we modeled the different rules and obtained a fuzzy engine that was finally slightly modified to obtain an engine appropriate for our purpose.

Additionally we set the values of certain parameters of the rule base that were not set previously, as is the case of the different fuzzy sets:

- *Enough* is a fuzzy set with trapezoidal membership function where $(a, b, c, d) = (0.005, 0.01, 1, 1)$.
- $High$ is a fuzzy set with trapezoidal membership function where $(a, b, c, d) = (0.05, 0.1, 1, 1)$.
- $THigh$ is a fuzzy set with trapezoidal membership function where $(a, b, c, d) = (0.4, 0.5, 0.7, 0.8)$.
- $TVeryHigh$ is a fuzzy set with trapezoidal membership function where $(a, b, c, d) = (0.7, 0.8, 1, 1)$.

Finally, each instance was solved 10 times, and the results show the averages of error ratio, ($error = 100 \times \frac{(optimum - obtained\ value)}{optimum}$), and the average standard deviation. Every test was executed on an Intel core2 Quad 1.66Ghz with 2GB of Memory.

5.2 Performance of the individual metaheuristics

In this section we show the results obtained by the different metaheuristics that compose the system. We decided that each individual metaheuristic has to be executed during 90 seconds using the best parameters found during

the knowledge extraction process. In table 2 we show the average error ratio (average standard deviation) obtained by the individual metaheuristics, discriminated by instance sizes and types. We also show the average error ratio for two theoretical non cooperative systems where the different metaheuristics are executed in parallel and the best solution is chosen for each instance solved. The first non cooperative system (NCS-NI) did not use any information from the knowledge extraction process, and the second one (NCS-BP) used the best parameters obtained during the knowledge extraction process.

Table 2
Individual metaheuristics and non cooperative parallel systems

Type	Size	Genetic Alg.	Tabu Search	S. Annealing	NCS-NI	NCS-BP
Unc Span	500	5,07 (0,99)	1,21 (0,55)	0,21 (0,15)	0,70 (0,31)	0,17 (0,12)
	1000	18,06 (0,89)	0,90 (0,24)	0,21 (0,08)	0,71 (0,37)	0,14 (0,06)
	1500	25,14 (0,80)	1,17 (0,21)	0,20 (0,07)	0,98 (0,37)	0,16 (0,04)
	2000	28,66 (0,92)	1,05 (0,30)	0,13 (0,04)	1,01 (0,41)	0,11 (0,03)
Wea Span	500	1,87 (0,30)	3,10 (0,84)	2,02 (0,35)	1,51 (0,28)	1,67 (0,28)
	1000	6,17 (0,35)	1,81 (0,32)	2,18 (0,27)	1,59 (0,30)	1,44 (0,29)
	1500	9,48 (0,72)	2,35 (0,37)	2,31 (0,32)	1,95 (0,30)	1,71 (0,25)
	2000	12,03 (0,46)	2,62 (0,40)	2,60 (0,21)	2,06 (0,23)	1,95 (0,25)
Str Span	500	3,00 (0,29)	8,58 (1,97)	2,54 (0,24)	4,05 (0,82)	2,51 (0,25)
	1000	7,86 (0,34)	9,61 (1,29)	2,47 (0,16)	4,95 (1,63)	2,45 (0,16)
	1500	12,50 (0,97)	9,94 (0,58)	2,48 (0,15)	6,05 (2,06)	2,46 (0,15)
	2000	16,00 (0,89)	9,83 (1,08)	2,26 (0,08)	6,53 (2,27)	2,24 (0,08)
Pceil	500	1,15 (0,18)	0,47 (0,12)	0,30 (0,02)	0,41 (0,07)	0,29 (0,03)
	1000	4,96 (0,19)	0,58 (0,02)	0,36 (0,01)	0,43 (0,06)	0,36 (0,01)
	1500	9,88 (0,42)	0,64 (0,01)	0,39 (0,01)	0,45 (0,07)	0,39 (0,01)
	2000	10,75 (0,36)	0,63 (0,01)	0,38 (0,01)	0,46 (0,06)	0,38 (0,01)
Circle	500	7,12 (0,62)	19,59 (4,72)	7,00 (0,82)	14,34 (3,51)	7,00 (0,82)
	1000	19,86 (1,28)	24,94 (1,71)	7,82 (0,98)	15,33 (4,59)	7,82 (0,98)
	1500	27,92 (1,13)	25,52 (1,46)	8,22 (0,79)	17,41 (5,72)	8,22 (0,79)
	2000	32,84 (0,78)	25,39 (1,13)	7,97 (0,49)	14,92 (6,14)	7,97 (0,49)

In the table 2, we can observe how the worst metaheuristic in terms of performance is the Genetic Algorithm, whose error differences are, moreover, statistically greater than the other ones, according to Wilcoxon's signed-ranks non-parametric test (a confidence level of 95Concerning the parallel systems, NCS-NI, as expected, performs worse than the best individual metaheuristic (also statistically) as it has less information (it does not use the best parameters). In contrast, NCS-BP obtains a better performance than the best metaheuristic, and the differences in performance are significant according to the non-parametric test. Although, in general the best metaheuristic is the Simulated Annealing, NCS-BP obtains better results, since for certain instances it is another metaheuristic which returns the best result.. These results are collected in the NCS-BP non cooperative system.

5.3 Comparing various parallel systems

We now construct several cooperative systems from individual metaheuristics Each cooperative system was executed during 90 seconds, appearing the coordinator each 250 milliseconds to control the cooperation.

One of these cooperative systems, KEPS-F, consists of a system like the one defined in section 4 based on fuzzy rules where parameters, instead of being obtained from a set of trees, are fixed for all kind of instances, meaning that, as a result of the knowledge extraction process, we globally obtain the weights assigned to each metaheuristic, a set of initial values for the parameters of each metaheuristic and different sets of parameter values to substitute the initial ones if it were necessary. The other cooperative system, CS-WB, at each communication moment exchanges the solution of the metaheuristic returning the worst result with that returned by the one with the best.

Table 3 shows the results obtained with the two cooperative systems. We also compare the results with the best ones obtained with the previous systems (individual Simulated Annealing metaheuristic and the NCS-BP non cooperative system).

Table 3
Some parallel systems and some cooperative parallel systems

Type	Size	S. Annealing	NCS-BP	CS-WB	KEPS-F
Unc Span	500	0,21 (0,15)	0,17 (0,12)	0,01 (0,01)	0,04 (0,02)
	1000	0,21 (0,08)	0,14 (0,06)	0,16 (0,11)	0,22 (0,09)
	1500	0,20 (0,07)	0,16 (0,04)	0,39 (0,32)	0,32 (0,24)
	2000	0,13 (0,04)	0,11 (0,03)	0,49 (0,42)	0,33 (0,29)
Wea Span	500	2,02 (0,35)	1,67 (0,28)	0,66 (0,33)	0,56 (0,11)
	1000	2,18 (0,27)	1,44 (0,29)	0,85 (0,31)	0,84 (0,11)
	1500	2,31 (0,32)	1,71 (0,25)	1,05 (0,43)	1,11 (0,14)
	2000	2,60 (0,21)	1,95 (0,25)	1,13 (0,48)	1,26 (0,13)
Str Span	500	2,54 (0,24)	2,51 (0,25)	0,96 (0,43)	1,12 (0,14)
	1000	2,47 (0,16)	2,45 (0,16)	1,90 (1,61)	1,19 (0,11)
	1500	2,48 (0,15)	2,46 (0,15)	2,19 (2,01)	1,36 (0,17)
	2000	2,26 (0,08)	2,24 (0,08)	1,93 (0,91)	1,60 (0,11)
Pceil	500	0,30 (0,02)	0,29 (0,03)	0,21 (0,06)	0,19 (0,03)
	1000	0,36 (0,01)	0,36 (0,01)	0,22 (0,04)	0,24 (0,01)
	1500	0,39 (0,01)	0,39 (0,01)	0,26 (0,05)	0,27 (0,02)
	2000	0,38 (0,01)	0,38 (0,01)	0,29 (0,07)	0,27 (0,01)
Circle	500	7,00 (0,82)	7,00 (0,82)	5,77 (4,15)	3,48 (0,40)
	1000	7,82 (0,98)	7,82 (0,98)	7,90 (4,08)	4,99 (0,39)
	1500	8,22 (0,79)	8,22 (0,79)	7,37 (2,11)	5,96 (0,41)
	2000	7,97 (0,49)	7,97 (0,49)	9,38 (4,36)	6,72 (0,34)

We can observe, in table 3, how cooperative systems show a better behavior than the individual metaheuristics and non cooperative systems. Error differences are statistically significant according to Wilcoxon's signed-ranks non-parametric test.

We can also observe that between cooperative systems the one which shows the best behavior is that controlled by fuzzy rules obtained by knowledge extraction. Error differences between the two cooperative systems, CS-WB and KEPS-F, are statistically significant according to the same non-parametric test. The error obtained by the KEPS-F system is less than that obtained by the CS-WB. As can be seen, KEPS-F gives a very reasonable behavior in the harder instances. On the basis of these reasons we can state that the selected cooperative model is efficient and robust, and the information added in its base of rules is useful.

5.4 Performance of the KEPS cooperative system

In the previous section we focused on the comparison of KEPS-F with other parallel systems and confirmed its superior performance. In this section we study, the benefits of the adaptative behavior arising from the use of trees to select system parameters. We compared KEPS with the system where parameters, instead of being obtained from a set of trees, are fixed for all kind of instances, KEPS-F. In our opinion the inclusion of trees in KEPS helps to improve its adaptability and robustness, in contrast with the fixing of the parameters, which does not take into account the instance being solved. The results of this comparison are shown in table 4, where it can be appreciated how KEPS outperforms CS-WB and KEPS-F systems in almost every case. Besides, the difference between these parallel strategies was significant, according to the non-parametric test. Thus, the usefulness of trees to obtain system configuration is demonstrated.

As commented above, the systems were executed during 90 seconds, with the coordinator controlling the cooperation every 250 milliseconds.

One comment that can be deduced from the results of KEPS relates to the hardness of the instances of the knapsack problem. The hardness ratio established by KEPS is Unc Span<pceil<Wea Span<Str Span<circle.

Table 4
Incorporation of knowledge and models in cooperative parallel system: KEPS

Type	Size	CS-WB	KEPS-F	KEPS
Unc Span	500	0,01 (0,01)	0,04 (0,02)	0,01 (0,03)
	1000	0,16 (0,11)	0,22 (0,09)	0,14 (0,05)
	1500	0,39 (0,32)	0,32 (0,24)	0,23 (0,07)
	2000	0,49 (0,42)	0,33 (0,29)	0,13 (0,07)
Wea Span	500	0,66 (0,33)	0,56 (0,11)	0,52 (0,09)
	1000	0,85 (0,31)	0,84 (0,11)	0,75 (0,12)
	1500	1,05 (0,43)	1,11 (0,14)	1,00 (0,13)
	2000	1,13 (0,48)	1,26 (0,13)	1,15 (0,10)
Str Span	500	0,96 (0,43)	1,12 (0,14)	0,78 (0,14)
	1000	1,90 (1,61)	1,19 (0,11)	1,13 (0,09)
	1500	2,19 (2,01)	1,36 (0,17)	1,22 (0,12)
	2000	1,93 (0,91)	1,60 (0,11)	1,48 (0,07)
Pceil	500	0,21 (0,06)	0,19 (0,03)	0,16 (0,03)
	1000	0,22 (0,04)	0,24 (0,01)	0,22 (0,02)
	1500	0,26 (0,05)	0,27 (0,02)	0,25 (0,02)
	2000	0,29 (0,07)	0,27 (0,01)	0,26 (0,02)
Circle	500	5,77 (4,15)	3,48 (0,40)	3,09 (0,42)
	1000	7,90 (4,08)	4,99 (0,39)	4,52 (0,34)
	1500	7,37 (2,11)	5,96 (0,41)	5,55 (0,37)
	2000	9,38 (4,36)	6,72 (0,34)	6,31 (0,29)

6 Discussion and conclusions

In this paper, we have shown the construction of a hybrid, centralized, cooperative system of metaheuristics for solving optimization problems. The system was built as a multi-agent system, in which each metaheuristic is an agent that has to solve the problem while coordinating itself with the remaining metaheuristics. In order to accomplish this coordination we have used a coordinating agent which controls and modifies the behavior of the agents during their execution. We have given the coordinator knowledge ("intelligence") so that it has reasonable behavior when controlling the metaheuristics. This "intelligence" is the result of a prior knowledge extraction process which is reflected in a set of fuzzy rules and various models represented by decision trees.

It is important to mention that if we increase the number of metaheuristics cooperating, the knowledge extraction process that incorporates the knowledge will take more time, but the performance of the system will not decrease, because each metaheuristic is independent of the others (and they are executed parallely). Only the coordinator will be a bit slower, but its execution time is negligible compared with the execution time of the metaheuristics.

In order to test the idea, the "intelligent" system was used to deal with {0,1}-knapsack problem. From the comparisons we observe how in almost every case the hybrid system obtains better results than the individual metaheuristics. In the comparison with non-cooperative and cooperative parallel systems we could observe how the cooperative system KEPS performed better.

The results are very satisfactory and show that these are good systems for problems for which, on account of their characteristics, there is no algorithm or metaheuristic which outperforms the others in terms of finding better solutions for instances, and also in cases where some metaheuristics can help others to find better results.

Note that there is some cost of incorporating knowledge for the coordinator. To alleviate this we are working on incorporation in the "Active Learning" system [6] in order to reduce the instances we need to solve for this extraction process. We could also use "Learning by reinforcement", [24]. In the latter, we use an almost blind initial basic system, which will develop as the instances are solved and knowledge is acquired.

KEPS behaves well in general. As we have seen in the tables, the improvement in the easy type instances is small, But in hard type instances it is highly significant (a relative improvement of 31,57% in Str Span and 36,64% in Circle), which indicates that cooperation between metaheuristics is interesting for improving the solution of the instance to be solved. As was observed in subsection 5.4, the order of hardness of the problem {0,1}-knapsack problem is different to that proposed by Pisinger [21] and Pelta et al. [20]. As Pelta et al. Point out [20], this may be due to the type of metaheuristics that are cooperating, although in the case of Pelta et al. this is logical since there are clones from the same technique so that they can collaborate amongst themselves. In our case, KEPS is more heterogeneous since it contains metaheuristics of different natures and takes the goodness and improvements of each.

Acknowledgements

The authors thank the "Ministerio de Educación y Ciencia" of Spain and the "Fondo Europeo de Desarrollo Regional" FEDER) for their support given to this work under project TIN2005-008404-C04-02, and the "Fundación Séneca, Agencia de Ciencia y Tecnología de la Región de Murcia (Plan Regional de Ciencia y Tecnología 2007/2010)" which gives support under the "Programa de Ayudas a Grupos de Excelencia de la Región de Murcia".

References

[1] A. Blanco, D. Pelta, J. L. Verdegay, A fuzzy valuation-based local search framework for combinatorial problems, Fuzzy Optimization and Decision Making 1(2) (2002) 177–193.

[2] P. Bonissone, J.M. Cadenas, M.C. Garrido, J.J. Hernández Data Mining en el Modelado de un Sistema Metaheurístico Cooperativo: Una Comparativa, in: Proceedings of the National Congress of Metaheurísticas, Algoritmos Evolutivos y Bioinspirados (MAEB2007), Tenerife, Spain, 2007, pp. 215–222.

[3] E. K. Burke, B. MacCarthy, S. Petrovic, R. Qu, Knowledge Discovery in a Hyper-Heuristic Using Case-Based Reasoning on Course Timetabling, in: Proceedings of the 4th International Conference on the Practice and Theory of Automated Timetabling (PATAT 2002), Springer, Lecture Notes in Computer Science Vol. 2740, 2002, pp. 276-86.

[4] J.M. Cadenas, M.C. Garrido, L.D. Hernández, E. Muñoz, Towards the definition of a Data Mining process based in Fuzzy Sets for Cooperative Metaheuristic systems, in: Proceedings of the 11th Information Processing and Management of Uncertainty in Knowledge-Based systems Internacional Conference (IPMU2006), Paris, Francía, 2006, pp. 2828–2835.

[5] J.M. Cadenas, R.A. Díaz-Valladares, M.C. Garrido, L.D. Hernández, E. Serrano, Modelado del Coordinador de un sistema Meta-Heurístico Cooperativo mediante SoftComputing, in: Proceedings of the Campus Multidisciplinar en Percepción e Inteligencia (CMPI2006), Albacete, Spain, 2006, pp. 679–689.

[6] D. Cohn, L. Atlas, R. Landner, Improving Generalization with Active Learning, Machine Learning, 15 (1994) 201–221.

[7] D. Corne, M. Dorigo, F. Glover (Eds.), New Ideas in Optimization, McGraw-Hill, 1999.

[8] T.G. Crainic, M. Toulouse, Parallel Metaheuristics, in: T.G. Crainic, G. Laporte (Eds.), Fleet Management and Logistics, Kluwer Academic, Norwell MA, 1998, pp. 205–251.

[9] T. G. Crainic, M. Toulouse, Cooperative parallel tabu search for capacitated network design, Journal of Heuristics 8 (2002) 601–627.

[10] T. G. Crainic, M. Toulouse, Parallel Strategies for Metaheuristics, Handbook of Metaheuristics, Kluwer Academic Publisher, 2003, pp. 475–513.

[11] T. G. Crainic, M. Gendreau, P. Hansen, N. Mladenovic, Cooperative parallel variable neighborhood search for the p-median, Journal of Heuristics 10 (2004) 293–314.

[12] F. Glover, G.A. Kochenberger, Handbook of Metaheuristics, Kluwer Academic, Dordrecht, 2003.

[13] H. Guo, A Bayesian Approach for Automatic Algorithm Selection, in: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI03), Workshop on AI and Autonomic Computing, Acapulco, Mexico, 2003, pp. 1–5.

[14] H.Guo, H.W. Hsu, A machine learning approach to algorithm selection for NP-hard optimization problems: a case study on the MPE problem, Annals of Operation Research 156(1) (2007) 61–82.

[15] C.Z. Janikow, Fuzzy decision trees: issues and methods, IEEE Transaction System, Man, and Cybernetics, Part B. 28(1) (1998) 1–14.

[16] A. Le Bouthillier, T. G. Crainic, A cooperative parallel meta-heuristic for the vehicle routing problem with time windows, Computers and Operations Research 32(7) (2003) 1685–1708.

[17] B. Melián, J.A. Moreno, J.M. Moreno, Metaheurísticas: un visión global, Revista Iberoamericana de Inteligencia Artificial 19 (2003) 7–28.

[18] F. J. Moreno-Velo, I. Baturone, S. Sánchez-Solano, A. Barriga, XFUZZY 3.0: A Development Environment for Fuzzy Systems, in: Proceeding of the International Conference in Fuzzy Logic and Technology, Leicester, 2001, pp. 93–96.

[19] P. Pardalos, M. Resende (Eds.), Handbook of Applied Optimization, Oxford University Press, Oxford, 2002.

[20] D. Pelta, C. Cruz, A. Sancho-Royo, J. L. Verdegay, Using memory and fuzzy rules in a cooperative multi-thread strategy for optimization, Information Sciences 176(13) (2006) 1849-1868.

[21] D. Pisinger, Where are the hard knapsack problems?, Computer and Operations Research 32(9) (2005) 2271-2284.

[22] J. R. Rice, The algorithm selection problem, Advances in Computers 15 (1976) 65–118.

[23] P. Souza, S. Talukdar, Asynchronous organizations for multi algorithm problems, ACM Symposium on Applied Computing, Indianapolis, 1993.

[24] R.S. Sutton, A.G. Barto, Reinforcement Learning: An Introduction, MIT Press, Cambridge, MA, 1998.

[25] E. G. Talbi, A Taxonomy of Hybrid Metaheuristics, Journal of Heuristics 8 (2002) 541–564.

[26] F. Thabtah, P. Cowling, Mining the data from a hyperheuristic approach using associative classification, Expert Systems with Applications 34(2) (2008) 1093–1101.

[27] C. Tsai, C. Tsai, C. Tseng, A new hybrid heuristic approach for solving large traveling salesman problem, Information Sciences 166 (2004) 67–81.

[28] J.L. Verdegay, R.R. Yager, P.P. Bonissone, On heuristics as a fundamental constituent of soft computing, Fuzzy Sets and Systems 159(7) (2008) 846–855.