Facing Dynamic Optimization Using a Cooperative Metaheuristic Configured via Fuzzy Logic and SVMs

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

[a]*Department of Information and Communications Engineering, Campus of Espinardo, Murcia University, 30100 Murcia, Spain*

Abstract

Many dynamic optimization problems appear in the real world, and to solve them we need to find strategies that can track the optimum as it moves in the search space. In this paper we propose the use of a cooperative metaheuristic to cope with such problems. In this strategy different metaheuristics cooperate under the supervision of a coordinator. This coordinator is able to control the cooperation using a collection of Support Vector Machine models and a fuzzy decision framework. The combination of these two techniques allows us to modify the behavior of the strategy depending on the instance being solved. In order to obtain the models we use a well defined knowledge extraction process, which is performed only once and before operating the strategy. To test the validity of this approach we have applied it to a combinatorial optimization problem and a continuous optimization problem, respectively, Dynamic Knapsack Problem and Moving Peaks Benchmark. The strategy has been compared with other individual and cooperative strategies with very interesting results.

Keywords: Dynamic Optimization, Cooperative Metaheuristic Systems, Hybrid Metaheuristics

1. Introduction

An optimization problem (OP) is the problem of finding the best solution from all feasible solutions. Such problems appear constantly in our daily lives e.g. organizing our agenda. We are able to solve them because they are still manageable, i.e., they are of a small enough dimension to be processed. However, most OPs are NP-hard and when they grow to larger scales they become difficult to solve. For a long time, these problems have awakened the interest of mathematicians and computer scientists and as a result a great variety of strategies to solve OPs have appeared. Metaheuristics stand out on account of their success and their reduction of computational resources use.

Metaheuristics [36] are intelligent strategies used to design and improve very general heuristic procedures with a high performance. Metaheuristic research has traditionally focused on static optimization problems - those problems where the search space does not change during search time. However, real-world OPs are rarely static, in fact, they are usually dynamic i.e. their search space may change during a run. Some examples of such problems are: the balance of load in telecommunications networks [46], where the traffic conditions vary over time; the control of a greenhouse [52, 53] where the environmental conditions may change; the problem of meeting fall-due dates in a flowshop production environment [49] with jobs with different weights and uncertain

*Principal corresponding author

Email addresses: jcadenas@um.es (J.M. Cadenas), carmengarrido@um.es (M.C. Garrido), enriquemuba@um.es (E. Muñoz)

processing times and where new jobs may arrive with time; or the dynamic optimization of human walking [1], where the problem is to calculate the muscle stimulation records, muscle forces, and limb motions subject to minimum metabolic energy expenditure per distance unit traveled. These kinds of problems are known as Dynamic Optimization Problems (DOPs) [10].

In this paper we present an adaptive cooperative metaheuristic to cope with DOPs. Here different metaheuristics are executed in a parallel way while they cooperate to solve an instance of a DOP, with the cooperation being controlled by a coordinator.

In order to direct the parallel cooperation the coordinator is able to adapt its behavior during the resolution of the DOP, adjusting its parameters to the different scenarios that appear after a change in the search space. Adaptation is performed by exploiting a set of Support Vector Machines (SVM) models and a fuzzy decision making framework which modify the behavior of the coordinator depending on the instance being solved. SVM models contain interesting knowledge about the problem in hand, and help the coordinator to choose the best combination of parameters for each metaheuristic. In addition, a set of fuzzy rules configured using SVM models permits the coordinator to decide when a metaheuristic is obtaining poor performance and needs information from the rest of metaheuristics.

The paper is structured as follows. In Section 2 we present some related work, and in Section 3 we introduce cooperative metaheuristics. In Section 4 the cooperative strategy and its construction process are presented. Next, in Section 5 some experiments are carried out to test the validity of the approach proposed. Finally Section 6 offers the conclusions drawn.

2. Related Work

Several algorithms and strategies have been proposed to tackle DOPs, with the prevailing strategies being those which maintain a collection of solutions that evolve during execution i.e. population based metaheuristics. The acceptance of these approaches is based on a simple idea: if a population of solutions is maintained then it should be easier to locate new optimums as the search space changes. Among these strategies the most used are Evolutionary Algorithms (EAs) [10].

The main problem with EAs is the tendency of the population to converge to the same point of the search space. For that reason many efforts have focused on maintaining the diversity of the populations with different techniques - inserting randomly generated individuals [26], niching [15], adaptively modifying parameters [4, 17], using explicit memory structures [8], etc.

Apart from EAs other population based techniques have been used to cope with DOPs and interesting results, like Ant Colony Optimization (ACO) [2, 31] or Particle Swarm Optimization (PSO) [7] have been forthcoming. Another interesting research line, which has been shown to give excellent results, is the use of multipopulation strategies, i.e. those in which a set of populations evolves in parallel to find the optimum. Some examples are [20, 5].

However, the prevalence of population based metaheuristics applied to DOPs has hindered the development of trajectory based strategies - those that search the solution space performing iterative changes on the initial solutions. Some trajectory based strategies applied to DOPs have been proposed, though [21, 34, 25, 13]. These strategies have demonstrated that trajectory based metaheuristics are valid approaches able to obtain results similar to those obtained by population based strategies. For this reason we believe that cooperative strategies which involve both population and trajectory based metaheuristics can be used to deal with DOPs and give interesting results.

3. Cooperative metaheuristics

In the following lines we will give a brief introduction to cooperative metaheuristic strategies. The development of such strategies consists of the combination of existing metaheuristics in order to obtain hybrid strategies, where the metaheuristics cooperate in a parallel way. This approach can be interesting since robust strategies can be obtained which offer high quality solutions in spite of the changes in the instances characteristics.

Due to their ability to cooperate parallelly to solve an instance of an OP, cooperative metaheuristics are closely related to parallel metaheuristics, i.e. strategies which parallelly explore the search space. In fact cooperative metaheuristics can be considered a part of parallel metaheuristics, as the latter includes other strategies which cannot be considered cooperative. Several studies [19, 39, 12] have shown that parallel metaheuristics provide better solutions than their sequential components, even when the available execution time is smaller. These studies have also shown that the combination of different threads, each using a different metaheuristic, increases robustness of the global search with respect to changes in the instances of the problem. Moreover, 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. A good example of this kind of strategy is that proposed in [33], or the A-teams, proposed in [48].

The combination of different types of metaheuristics leads us to the field of hybrid metaheuristics, which seeks to make use of the complementary properties that characterize population and trajectory based metaheuristics. Population based metaheuristics allow a better exploration of the search space, while trajectory based metaheuristics allow a greater search intensification in the most promising areas. Combinations of algorithms such as descent local search, simulated annealing, tabu search, and evolutionary algorithms have provided very powerful search algorithms able to obtain the best results found for many practical or academic optimization problems [50].

Regarding DOPs, different strategies have been proposed that, although not using a parallel execution, do perform cooperation between different populations. The self organizing scouts [9] is a clear example of these approaches, where a multipopulation EA is proposed. But not only EA multipopulation approaches have been proposed, PSO multipopulations have also become an interesting research line of which [6, 20] are clear examples. On exploring other directions we find [25] in which different trajectory based metaheuristics cooperate or [40] in which different agents parallelly cooperate to solve a DOP, with each agent implementing a simple optimization strategy.

However the use of hybrid cooperative strategies to solve DOPs is not extended. Nevertheless we believe that their characteristics will help in the resolution of this kind of problems. In our opinion the use of different metaheuristics will allow the system to track the optimal solution better as the search space changes for two main reasons: first, the use of different metaheuristics increments the diversity of the solutions, second, the use of different metaheuristics with different search strategies favors at least one of them being abe to track the optimum and guide the others.

Another promising approach in the field of metaheuristic hybridization [30] is the use of data mining techniques to help hybridization and improve metaheuristics' performance. Data mining is the process of automatically exploring large volumes of data to discover patterns. In order to achieve this goal, data mining uses computational techniques from statistics, machine learning, pattern recognition or combinatorial optimization. The combination of data mining with the use of heuristics and metaheuristics has become a popular research field which has shown promising results. An interesting review of the field can be found in [30]. Of these approaches we can outline the strategy proposed in [12] which was applied to static optimization problems and obtained good results. That strategy proposed a synchronous cooperative metaheuristic configured using fuzzy decision trees. However in recent literature it has been argued that asynchronous strategies generally obtain better results than synchronous strategies. We believe that an asynchronous cooperative

metaheuristic system, controlled by the knowledge obtained from an automatic knowledge extraction process, will be an interesting tool to solve DOPs.

4. A cooperative metaheuristic strategy to solve dynamic optimization problems

In this section we describe the cooperative strategy on which we focus our attention. It will be known as KEPS+D. It is a hybrid approach where different metaheuristics cooperate in an asynchronous way. Additionally it uses data mining techniques (in particular SVM) and a cooperation scheme based on a set of fuzzy rules to adapt its behavior to the changing space of a DOP, and therefore improve its performance.

Three main issues shall be discussed, the architecture of the system, the cooperation scheme which permits a synergy to be created among the different strategies and the knowledge extraction process that gives the system adaptability.

4.1. Architecture of the cooperative metaheuristic strategy KEPS+D

The computation of the proposed strategy is performed by executing different metaheuristics in a parallel way in order to explore the search space. The strategy is able to intelligently choose the metaheuristic's configuration parameters and to control the cooperation between metaheuristics, which is carried out by exchanging solutions in an adaptive way.

The strategy is modeled using a multiagent system, Fig. 1, which contains two types of agents - *optimization agents* and a *coordinator agent*.

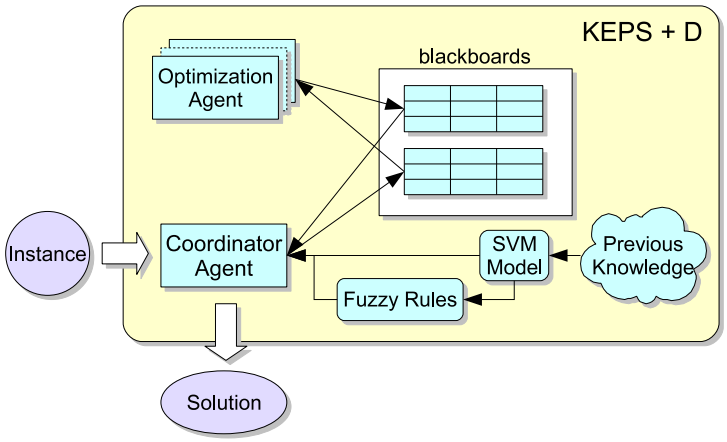

Figure 1: A multiagent system of metaheuristics

Each optimization agent executes a metaheuristic and the coordinator agent has 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. In particular, to measure the performance of an optimization agent and decide if it is obtaining a poor behavior we will use the value of the objective function. But other measures can be used, such as the increment in this value since the last solution share or the presence in the solution of characteristics identified as promising.

The coordinator agent is responsible for initiating optimization process by parallel activating optimization agents, and it coordinates the cooperation among optimization agents by choosing their configuration parameters and controlling their exchange of solutions in an adaptive way. The adaptability task of coordinator agent is performed by considering knowledge coming from machine learning coded by a collection of SVM models. In particular, SVM models the support coordinator

agent in 1) choosing the best metaheuristic parameters and 2) ranking the metaheuristics suitability to solve a given instance through numerical weights in the range $[0, 1]$. Via weights analysis, the coordinator may realize that a metaheuristic is poorly performing and, consequently, may update its solutions by adding a set of better solutions computed by other more suitable metaheuristics. This process allows poor metaheuristics to restart their optimization task in a more interesting point of the search space.

After the initialization stage performed by the coordinator agent, optimization agents execute asynchronously their optimization strategies while sending and receiving information. The information exchange is performed by means of a blackboard scheme. Two blackboards are used. The first is used by the optimization agents to publish information about their behavior, the second by the coordinator to post the instructions for each optimization agent. After a specified period of time every metaheuristic stops and posts its current solution/population (depending on its type - trajectory or population based). The coordinator agent then analyses the behavior of each metaheuristic, taking into account previous knowledge, and decides if any strategy needs to improve its performance. In this case, the coordinator writes the new solution/population that the metaheuristics have to use on the second blackboard. Algorithms 1 and 2 respectively illustrate the behavior of optimization and coordinator agents by means of pseudocode.

Algorithm 1: Optimization Agent's Pseudocode

```
Input: q: Instance of the optimization problem, m_i: Metaheuristic to use, n: execution period
begin
    while end condition is not satisfied do
        read instructions from blackboard;
        if instructions are received then
            replace current solution/population with solution/population contained in instructions
        end
        solution=compute metaheuristic m_i during n;
        write solution on blackboard
    end
end
```

Algorithm 2: Coordinator Agent's Pseudocode

```
Input: SVM: set of SVM models to configure the coordinator, n: execution period for each
       optimization agent, q: Instance of the problem, M: set of metaheuristics
Output: solution: the best solution found
begin
    analyze SVM in order to select the best parameter values for each metaheuristic in M
    according to q;
    while end condition is not satisfied do
        compute in a parallel way each agent using a metaheuristic in M;
        exploit fuzzy rules, SVM and blackboard data to select POOR, a set of agents performing
        poorly;
        foreach agent a in POOR do
            create instructions_a containing a solution/population of more suitable solutions;
            write instructions_a in the blackboard;
        end
    end
    return best solution computed by optimization agents;
end
```

As a way of further adapting the system to DOPs, the coordinator agent will be monitoring the instance being solved. As soon as it notices that the instance has changed, its configuration is revised. To do that, it uses the previously mentioned SVM models, which are independent of the instance being solved, and obtains new weights and parameter values for each metaheuristic. Like this it can adapt better to the changing environment.

Two approaches have been proposed in the literature to face the problem of detecting a change, [7]: assuming that the algorithm knows when the environment change occurs, or that it can detect the change. Following [51], which states that the detection is yet another challenge alongside the one that concerns efficient adaptation to the change, we have decided not to solve two problems at the same time. Therefore, we assume that the coordinator agent "knows" when the environment has changed. We expect that once the detection mechanism has been conveniently studied and adjusted it will be able to detect the changes soon after their appearance and with a low cost in computational resources.

After introducing the architecture of the strategy, we tackle a crucial question in the design of the cooperative strategy, the cooperation scheme, which will be presented in the following subsection.

4.2. Cooperation scheme

The cooperation scheme of this strategy is mainly defined by the control rules used by the coordinator and the SVM models that guide its adaptation mechanism. These components have to help the coordinator agent to:

- choose the best metaheuristics' parameters values;
- individuate when poor metaheuristics have to receive new solutions from other strategies that are showing a better behavior.

First, behavior is carried out by means of a set of SVM [42, 47] models obtained from the application of a supervised learning process, which is performed only once and before a system's operation. Each of these models is related to a different parameter of a different metaheuristic. We have realized that the inference of the whole configuration generates a high number of classes, which increases error rate. Therefore, the use of a different model for each parameter reduces error rate. It could be argued that the separate inference of the parameters is inaccurate as long as their values are dependent. However, the learning process, which learns from previous executions, avoids this problem because of the correlations that appear in the dataset i.e. for each instance we store the best configuration of parameters, which we learn separately for each parameter.

The coordinator agent carries out the remaining behavior by considering a different set of SVM models coming from machine learning. In detail, these models analyze the instance at hand and rank the different cooperating metaheuristics, by returning a weights collection $\Omega = \{\omega_i, i = 1, \ldots, nm\}$ where nm is the number of metaheuristics, $\omega_i \in [0, 1]$ and $\sum_{i=1}^{nm} \omega_i = 1$.

In short, each weight ω_i is associated with a metaheuristic m_i and represents its suitability to solve the instance at hand. More precisely, let m_i, m_j be two optimization strategies then $\omega_i > \omega_j$ implies that, according to previous executions, i^{th} metaheuristic obtains an overall better performance than that obtained by j^{th} metaheuristic.

In order to control solution exchange, the coordinator uses the following decision framework, consisting of a set of fuzzy rules, which exploits Ω, this collection is replicated once for each metaheuristics:

if $(\omega_1 \cdot \delta_1)$ is *enough* **then** send solutions from m_1 to m_h
if $(\omega_2 \cdot \delta_2)$ is *enough* **then** send solutions from m_2 to m_h
...

if $(\omega_{h-1} \cdot \delta_{h-1})$ is *enough* **then** send solutions from m_{h-1} to m_h
if $(\omega_{h+1} \cdot \delta_{h+1})$ is *enough* **then** send solutions from m_{h+1} to m_h
...
if $(\omega_{nm} \cdot \delta_{nm})$ is *enough* **then** send solutions from m_{nm} to m_h

where:

- nm is the number of metaheuristics.
- m_h is the metaheuristic being evaluated by the rule.
- $\delta_i = (\xi_i - \xi_{m_h})/maximum(\xi_i, \xi_{m_h})$, where ξ is a measure of the performance shown by metaheuristics, defined by the user.
- $\omega_i \in [0, 1]$ where $\sum_{i=1}^{nm} \omega_i = 1$ and ω_i represents the weight of meta-heuristic i (importance of metaheuristic i for solving the current instance).
- *enough* is a fuzzy set with trapezoidal membership function with support contained in $[0, 1]$ defined by a quadruplet (a,b,c,d). The mathematical representation is shown in the following equation:

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

  with (a,b,c,d)=(0,0.1,1,1).

This set of rules indicates to the coordinator when it has to send new solutions to a given metaheuristic. It changes the position in the search space of a metaheuristic which is showing a bad performance for a position near the position of another metaheuristic with a better behavior. The firing of the rule is controlled by an $\alpha - cut$ defined by the user.

The use of $\alpha - cuts$ is illustrated in Fig. 2 with a graphical example. In the upper part of the figure we can observe a case in which the activation of the rule is higher than the $\alpha - cut$ $((\omega_i \cdot \delta_i) \geq 0.1\alpha)$ and thus, the rule is fired. On the other hand, the lower part of the figure represents a case in which activation is under the $\alpha - cut$ $((\omega_j \cdot \delta_j) < 0.1\alpha)$ and consequently the rule is not fired.

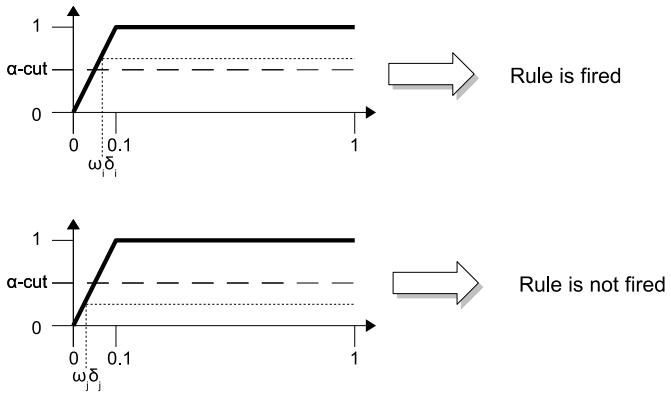

Figure 2: $\alpha - cut$ example.

Note that the definition of "enough" fuzzy set could have used any type of membership function, but this does not change the design of the strategy. From a global perspective, another type of membership function (along with the time interval for cooperation) would produce the same behavior of the strategy as we conclude in the experiments section. Therefore, we do not lose generality if we use the trapezoidal membership function.

In the event that more than one rule is activated the coordinator applies all of them. In this way, it is capable of simultaneously adapting the behavior of several poor metaheuristics.

In order to change the solution of the poor metaheuristics, we take into account the following situations:

- The $m_{receiver}$ is based on trajectories. A solution "near" to the best solution obtained from the metaheuristics that have fired a rule is sent to it. "Near" means that a mutation operator is used a number of times equals to $1/(2 * \omega_{m_{sender}})$.
- $m_{receiver}$ is based on populations. There are different options:
  - m_{sender} is trajectory based. A proportion of the worst individuals of $m_{receiver}$ equal to the $\omega_{m_{sender}}$ is substituted by a set of solutions consisting of solutions near m_{sender} best solution.
  - m_{sender} is population based. A proportion of the worst individuals of $m_{receiver}$ equal to the $\omega_{m_{sender}}$ is substituted by a set of solutions consisting of the best members of the population of m_{sender}.
  - Different metaheuristics have to send solutions. A proportion of the worst individuals of $m_{receiver}$ equal to the sum of the ω_i of the senders is substituted by a set of solutions where each m_{sender} chooses its solutions as described above.

To obtain parameters' and weights' models we use a knowledge extraction process like the one shown in Fig. 3. This process is composed of two different phases - data preparation and data mining - which will be explained in the following section. Note that this process is a supervised process and previous to system operation.

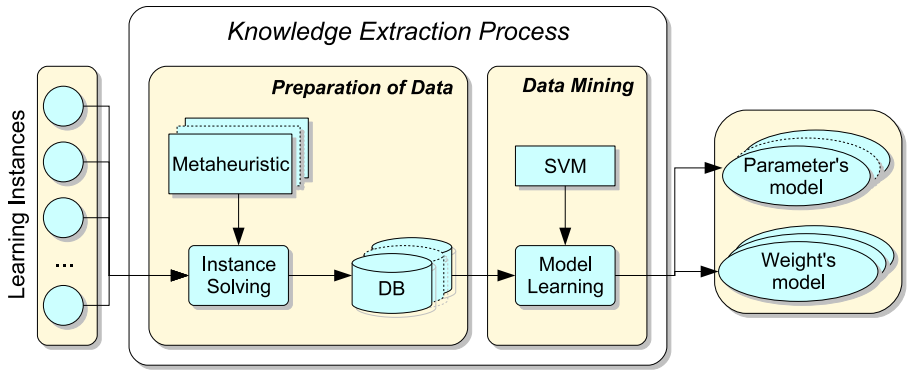

Figure 3: Knowledge extraction process

4.3. Knowledge Extraction Process

4.3.1. Data preparation phase

In this phase the different metaheuristics are executed several times using different parameter values in order to obtain a set of datasets which describes the behavior of each metaheuristic.

These datasets will be exploited using SVM to obtain the models that control parameters and weights. The obtaining of the datasets is performed in two steps:

a) In the first step a set of datasets is obtained which contains rough information (called *rough datasets*). In particular, for each metaheuristic we have to choose a set of values for each parameter. Some experiments have demonstrated that for each parameter at least 10 values covering its domain should be chosen. This number can be reduced using expert knowledge or some previous experiments that could guide the choice. Once the parameters values have been chosen each training instance has to be solved using all possible combinations, repeating each execution a number of times, k, which guarantees stable results. k should take values in the range $[10, 20]$. The information obtained from these executions is stored in the rough datasets. A different rough dataset will be kept for each metaheuristic, and for each execution the following information is stored: a description of the instance being solved, the values for each parameter and the objective function value obtained.

Example 1. *Let us suppose a maximization problem P, a set of metaheuristics composed of Tabu Search, Simulated Annealing and Genetic Algorithm, a set of instances composed of two instances q_1 and q_2, each described by two values q^a and q^b, and for each metaheuristic the sets of parameters values shown in Table 1. Finally, each execution is repeated 2 times.*

Table 1: Parameters values

Tabu Search	**Simulated Annealing**	**Genetic Algorithm**	
Tabu list size (tls)	Cooling schedule (cs)	mutation prob. (mp)	cross prob. (cp)
5	Exponential	0.01	0.4
10	Lineal	0.001	0.7

After the first step the rough datasets obtained will be those shown in Table 2.

□

b) In the second step this information is processed to obtain the final datasets.

Before rough datasets are prepared for data mining it is necessary to refine them to obtain the *processed datasets*. These datasets contain refined information that is useful to obtain the previously defined SVM models for parameters and weights. Two different kinds of processed datasets can be distinguished - *parameters processed datasets* and *weights processed datasets*. The first type is related to the choice of parameters and there is one dataset for each parameter of each metaheuristic. The second type is related to the importance of each metaheuristic for a given instance, and there is one dataset for each metaheuristic.

The process to obtain all the parameters processed datasets associated to a metaheuristic m_i is the following:

1. for each instance in the training dataset and for each parameters combination the average objective function value is calculated.
2. for each instance in the training dataset the parameter combination computing the best average fitness value is obtained;
3. for each instance in the training dataset and for each parameter associated to m_i the parameters processed dataset is updated with a new entry containing the description of the instance and the parameter value included in the best parameter combination.

The process to obtain all the weights processed datasets is the following:

1. for each metaheuristic m_i, for each instance and for each parameters combination the average fitness value is computed.
2. for each m_i and for each instance, let us select the best average fitness value and compute the *metaheuristic weight* as:

$$\omega_i = \frac{fit_{best}^{q,i}}{\sum_{l=1}^{nm} fit_{best}^{q,l}}$$

   where $fit_{best}^{q,i}$ is the best objective function value associated to metaheuristic i and instance q.
3. for each metaheuristic m_i and for each instance m_i's weights processed dataset is updated with a new entry containing the description of the instance and the metaheuristic weight.

Example 2. *Following the previous example, we will now illustrate the process to construct the processed datasets. Table 2 shows the different steps applied.*

□

4.3.2. Data mining phase

Once performance information has been collected through processed datasets, the data mining phase extracts the collection of knowledge models. A main issue is the choice of the data mining technique. Our data mining approach exploits a SVM technique to extract knowledge from processed datasets and build the aforementioned collection of models that guide the coordinator.

We have to obtain two different groups of models, *parameter models* and *weights models*. The first kind of model is related to coordinator's choice of parameters values and requires the use of a classification learning technique - there is one data mining model for each parameter of each metaheuristic. On the other hand, the second type of model is related to obtaining the set Ω of weights and requires the use of a regression learning technique - there is one model for each metaheuristic.

SVM - Support Vector Machines

SVM are simple techniques widely used in machine learning. We will briefly explain the process to construct them, both for classification [42] and regression [47].

Classification with SVM - parameter models.

Let us suppose the processed dataset, $S = \{(\vec{x}_1, parameter_K), \ldots, (\vec{x}_M, parameter_K)\}$ of M examples that act as training dataset, where $\vec{x}_i = \{x_{i1}, \ldots, x_{in}\}$ is the description of an instance of the problem and $parameter_K = \{value_r, value_s\}$ (denoted by $\{-1, 1\}$) represents the value of the parameter $parameter_K$ of a given metaheuristic. This last attribute acts as class attribute for the classification process.

Despite the fact that SVM is a binary classifier, to learn parameters models we need multiclass models. To solve this problem we use the philosophy of pairwise classification.

If we are using a linear separator, SVM builds a model based on a n-dimensional hyperplane defined by $f(\vec{x}_i) = \vec{w} \cdot \vec{x}_i + b$, where $w \in \mathbb{R}^n$ y $b \in \mathbb{R}$, which separates the examples into two classes. But we will normally use non-linear classification, in which we need a function $\phi(\cdot)$ which transforms the set of examples S to a different space. In this way we can perform a linear classification on transformed examples. In this situation, examples appear as $\phi(x_i)$ and w will appear in the new

Table 2: Example processed datasets.

	Tabu Search				**Simulated Annealing**				**Genetic Algorithm**				
	Inst.	desc.	tls	solution	Inst.	desc.	cs	solution	Inst.	desc.	mp	cp	solution
Rough Datasets	q_1^a	q_1^b	5	1245	q_1^a	q_1^b	Exp.	1126	q_1^a	q_1^b	0.01	0.4	1353
	q_1^a	q_1^b	5	1209	q_1^a	q_1^b	Exp.	1047	q_1^a	q_1^b	0.01	0.4	1438
	q_1^a	q_1^b	10	1757	q_1^a	q_1^b	Lin.	1325	q_1^a	q_1^b	0.01	0.7	1252
	q_1^a	q_1^b	10	1892	q_1^a	q_1^b	Lin.	1549	q_1^a	q_1^b	0.01	0.7	1188
									q_1^a	q_1^b	0.001	0.4	1724
									q_1^a	q_1^b	0.001	0.4	1654
									q_1^a	q_1^b	0.001	0.7	1032
									q_1^a	q_1^b	0.001	0.7	1164
	q_2^a	q_2^b	5	2345	q_2^a	q_2^b	Exp.	2526	q_2^a	q_2^b	0.01	0.4	2253
	q_2^a	q_2^b	5	2596	q_2^a	q_2^b	Exp.	2848	q_2^a	q_2^b	0.01	0.4	2138
	q_2^a	q_2^b	10	2157	q_2^a	q_2^b	Lin.	2555	q_2^a	q_2^b	0.01	0.7	2762
	q_2^a	q_2^b	10	2126	q_2^a	q_2^b	Lin.	2456	q_2^a	q_2^b	0.01	0.7	2334
									q_2^a	q_2^b	0.001	0.4	2454
									q_2^a	q_2^b	0.001	0.4	2632
									q_2^a	q_2^b	0.001	0.7	2132
									q_2^a	q_2^b	0.001	0.7	2321
Datasets Construction	↓ ↓ Average Computation ↓ ↓												
	q_1^a	q_1^b	5	1227	q_1^a	q_1^b	Exp.	1086.5	q_1^a	q_1^b	0.01	0.4	1395.5
	q_1^a	q_1^b	10	1824.5	q_1^a	q_1^b	Lin.	1437	q_1^a	q_1^b	0.01	0.7	1220
									q_1^a	q_1^b	0.001	0.4	1689
									q_1^a	q_1^b	0.001	0.7	1098
	q_2^a	q_2^b	5	2470.5	q_2^a	q_2^b	Exp.	2687	q_2^a	q_2^b	0.01	0.4	2195.5
	q_2^a	q_2^b	10	2141.5	q_2^a	q_2^b	Lin.	2505.5	q_2^a	q_2^b	0.01	0.7	2548
									q_2^a	q_2^b	0.001	0.4	2543
									q_2^a	q_2^b	0.001	0.7	2226.5
	↓ ↓ Parameters datasets entry generation ↓ ↓												
	q_1^a	q_1^b	10		q_1^a	q_1^b	Lin.		q_1^a	q_1^b	0.001		
	q_2^a	q_2^b	5		q_2^a	q_2^b	Exp.		q_2^a	q_2^b	0.01		
									q_1^a	q_1^b		0.4	
									q_2^a	q_2^b		0.7	
	↓ ↓ Weights datasets entry generation ↓ ↓												
	q_1^a	q_1^b	0.37		q_1^a	q_1^b	0.29		q_1^a	q_1^b	0.34		
	q_2^a	q_2^b	0.32		q_2^a	q_2^b	0.35		q_2^a	q_2^b	0.33		

space of higher dimension. SVM formulation becomes:

$$\begin{aligned} \text{Min} \quad & \tfrac{1}{2}||w||^2 + D\sum_{i=1}^{l}\xi_i \\ \text{s.t.} \quad & \\ & parameter_K(\vec{w}\cdot\phi(\vec{x}_i)+b) \geq 1-\xi_i \\ & \xi_i \geq 0, \quad i = 1,\ldots,l \end{aligned}$$

where

- D is a constant that indicates the importance of one criterion against another.
- ξ_i is a margin of error, i.e., ξ_i will be zero for those examples correctly classified and $\xi_i > 0$ for incorrectly classified examples.

- $\sum_{i=1}^{l} \xi_i$ represents a classification error measure.

This problem can be solved by building a Lagrangian problem and transforming it into a dual problem.

Regression with SVM - weights models. Let us suppose the processed dataset, $S = \{(\vec{x}_1, y_{H,1}), \ldots, (\vec{x}_M, y_{H,M})\}$ of M examples that act as training dataset, where $\vec{x}_i = \{x_{i1}, \ldots, x_{in}\}$ is the description of an instance for the problem and $y_{H,i}$ the associated weight for a metaheuristic H to solve that instance i. This last attribute acts as an attribute for the regression process.

In SVM regression, the goal is to find a function $f(\vec{x})$ that has at most ϵ deviation from the actually obtained targets $y_{H,i}$, for all the training examples.

Once again, SVM builds a model based on a n-dimensional hyperplane $f(\vec{x}_i) = \vec{w} \cdot \vec{x}_i + b$, and we can write the problem as a convex optimization problem:

$$\begin{array}{ll} \text{Min} & \frac{1}{2}||w||^2 + D\sum_{i=1}^{M}(\xi_{r,i} + \xi'_{r,i}) \\ \text{s.t.} & \\ & y_{H,i} - (\vec{w} \cdot \phi(\vec{x}_i) - b) \leq \xi_{r,i} + \epsilon \\ & (\vec{w} \cdot \phi(\vec{x}_i) + b) - y_{H,i} \leq \xi'_{r,i} + \epsilon \end{array}$$

Thus, if the examples are at a distance of less than ϵ from the hyperplane then these examples are not penalized (as in the classification with examples that were well classified). Unlike what happened in classification, in regression there are two different types of margins, $\xi_{r,i}$ and $\xi'_{r,i}$. As in classification, to solve this problem the use of a Lagrangian problem is necessary.

Implementation used. After theoretically explaining SVM, we have to indicate the actual implementation used in the learning process. In particular, we have used the Sequential Minimization Optimization proposed in [42] for classification, and improved in [47] to cope with regression.

Once we have applied the techniques to processed databases, this phase is finished and the collection of SVM models that guides coordinator agent is obtained. Thus, the coordinator is ready to control the execution of the optimization agents.

4.4. Classifying KEPS+D

The proposed cooperative system of metaheuristics can be included in diverse taxonomies depending on the point of view. If we consider the taxonomy of hybrid metaheuristics proposed in [50], the strategy proposed shall be included in type HTH, High level, teamwork hybrids, which includes those hybrids metaheuristics where different metaheuristics carry out a cooperative search. However, if we classify the strategy as a parallel metaheuristic, we shall include it in Type 3, parallel metaheuristics or multi-search parallelism, from the taxonomy proposed in [19]. This class includes those parallel metaheuristics that perform a multiple concurrent exploration of the solution space, specifically the strategy is included in cooperative strategies.

An important feature of the cooperative system of metaheuristics is the use of a knowledge extraction process to configure the control of the cooperation. Thus, we can consider the strategy as a hybridation of metaheuristics and knowledge extraction. Taking into account the taxonomy proposed in [30], we can define the strategy as a priori if we attend to the kind of knowledge used, because its knowledge is acquired before the execution of the strategy, its aim of cooperation is to improve the quality of the solutions, and the part of the metaheuristic to which knowledge is applied are the parameters.

5. Experimental analysis

In this section we will perform tests to assess the validity of the cooperative strategy proposed. In order to test the effectiveness of the proposal we will use two problems - a combinatorial optimization problem and a continuous optimization problem - which are, respectively, the Dynamic 0-1 Knapsack problem (DKP) and the Moving Peaks Benchmark (MPB). Both are typical DOPs that have been thoroughly studied in the literature. Furthermore, the cooperative system proposed is compared with the individual metaheuristics that compose it and with two other cooperative strategies.

Experiments will be computed in two steps. In the first step we will compare the results obtained by the system with those obtained by the individual metaheuristics that comprise it and with a simple cooperative strategy (CS-WB). This strategy is obtained using the same system of metaheuristics, but substituting the coordinator for a simpler one that only reads the best solutions obtained by each metaheuristic and sends the best solution in terms of fitness value to the metaheuristic which has the solution with the worst fitness value. Once the effectiveness of the strategy is demonstrated, we can perform second step, where we compare our strategy with a recently published state-of-the-art cooperative strategy [40].

5.1. Problems

5.1.1. Dynamic 0-1 Knapsack Problem

DKP is an NP-complete problem and one of the classical problems of dynamic optimization, which is an excellent benchmark to test novel algorithms in DOPs [45]. It 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:

$$\begin{aligned} Max \quad & \sum_{i=1}^{nob} p_i x_i \\ s.t. \quad & \\ & \sum_{i=1}^{nob} w_i x_i \ \leq \ C \\ & x_i \in \{0,1\} \, i = 1, \ldots, nob \end{aligned}$$

where

- nob 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 $\sum_{i=1}^{nob} w_i > C$.

DKP was first introduced by Goldberg and Smith in [24]. In that paper a very simple instance of DKP was defined in which C changed over time between two values. We consider this instance too simple to perform our tests, fortunately, so more challenging scenarios have been proposed:

- *Mori*: The first kind of instances that are going to be used are those defined in [37]. In these instances we have three different sizes 30, 60 and 90 items, with a w_i and a p_i chosen randomly from the interval [1,500].

  C changes over time according to the following pattern: C is reduced from 90% of c_{sum} to 10% of c_{sum} by 10% of c_{sum} where $c_{sum} = \sum_{i=1}^{nob} c_i$.

  Note that instances defined in [37] have only size 30, but we have decided to generate bigger instances to add diversity and increase their difficulty. Similarly their change pattern only allows us to perform 8 changes, in order to allow more instance changes we have decided that once C is equal to 10% of c_{sum} it changes to 90%, using a cyclic pattern.

- *Karaman*: The second kind of instances is defined in [32]. In these instances we have four different sizes: 100, 200, 300 and 400 items. Each item has a w_i and a p_i obtained randomly from the interval [1000,5000] and C changes randomly over time.

  Note that instances defined in [32] have only size 100, but we have decided to generate bigger instances to add diversity and increase their difficulty.

- *Str span*: The last kind of instances was recently proposed in [45]. These instances are inspired by the static instances proposed in [41] which have been demonstrated to be difficult to solve by state-of-the-art algorithms.

  In [45] only spanner instances are considered. These instances are constructed from a small set of parameters known as the spanner set. Each such set is characterized by the pair (v, m) where v is the number of items in the spanner set and m is a multiplier limit. A set of v items is generated with weights in $[1, R]$ (i.e., R denotes the maximum possible weight) and profits according to some distribution. In [45] strongly correlated instances are chosen, such that for each $w_i \in [1, R]$, the corresponding profit is $p_i = w_i + R/10$. Finally, all items in the spanner set are normalised. The *nob* items of the actual problem instance are then constructed by randomly choosing an item from the spanner set and a randomly chosen multiplier $a \in [1, m]$ such that each item is specified by $(a \cdot w_i, a \cdot p_i)$. It follows that the *nob* items are split into v disjunctive sets, each with a different profit-weight ratio.

  In order to make the problem dynamic, the weight of a spanner item i is denoted w_i, as a fraction of the maximum possible weight, $\alpha_j R$, where $\alpha_j \in [0, 1]$. The weights and hence the profit-weight ratio of group j may then be altered by changing the value of α_j. The dynamics are thus restricted to these coefficients and any type of function whose (scaled) output is in the range [0, 1] may be used to describe the trajectory of each α_j. In this case the following function is going to be used: $\alpha_j(t+1) = \mu\alpha_j(t)(1 - \alpha_j(t)$ where $\mu = 4$.

  For this scenario we have generated instances with four different sizes: 500, 1000, 1500 and 2000 elements. The rest of the parameters have been set to the following values, $v = 2$, $m = 10$, $R = 10000$. Note that in [45] the instances defined have only size 20, but we have decided to generate bigger instances to add diversity and increase their difficulty.

Finally, we should mention that in order to obtain the optimum of each instance generated we used the exact algorithm proposed in [35].

5.1.2. Moving Peaks Benchmark

MPB is one of the most used DOPs to test the effectiveness of the proposed strategies [40]. The MPB defines an n-dimensional landscape in which a pre-set number of peaks are defined with specific locations (X), heights (H), and widths (W). The peaks are distributed randomly in a restricted area. Each of the peaks may vary its height, width and location over time.

With the aim of bridging the gap between very complex, hard to understand real-world problems and too simple toy problems, in [8, 38] a problem with a multidimensional landscape consisting of several peaks was suggested, where the height, the width and the position of each peak is altered slightly every time a change in the environment occurs.

Note that a small change in the landscape may have two consequences: sometimes it may be sufficient to adapt the current solution to reach the new optimum, and at others it may be necessary to switch to another, previously slightly inferior but now better solution. The former happens e.g. when the optimum shifts slightly, while the latter happens when the height of the peaks changes such that a different peak becomes the maximum peak.

A test function with n dimensions and m peaks can be formulated as:

$$F(\vec{x}, t) = \max(B(\vec{x}), \max_{i=1\ldots m} P(\vec{x}, h_i(t), w_i(t), \vec{p_i}(t)))$$

where $B(\vec{x})$ is a time-invariant "basis" landscape, and P is the function defining a peak shape, where each of the m peaks has its own time-varying parameters height (h), width (w), and location ($\vec{p}$).

The coordinates, the height and the width of each peak are randomly initialized. Every Δe evaluations the height and width of every peak are changed by adding a random gaussian variable. The location of every peak i is moved by a vector $\vec{v_i}$ of fixed length s in a random direction (for $\lambda = 0$) or a direction depending on the previous direction (for $\lambda > 0$).

Overall, the parameter s allows to control the severity of a change, Δe will determine the frequency of change and λ allows to control whether the changes exhibit a trend.

More formally, a change of a single peak can be described as

$$\begin{aligned} \sigma &\in N(0,1) \\ h_i(t) &= h_i(t-1) + height\_severity \cdot \sigma \\ w_i(t) &= w_i(t-1) + width\_severity \cdot \sigma \\ \vec{p_i}(t) &= \vec{p_i}(t-1) + \vec{v_i}(t) \end{aligned}$$

The shift vector $\vec{v_i}$ is a linear combination of a random vector $\vec{r}$ and the previous shift vector $\vec{v_i}(t-1)$, and normalized to length s, i.e.

$$\vec{v_i}(t) = \frac{s}{|\vec{r} + \vec{v_i}(t-1)|}((1-\lambda)\vec{r} + \lambda\vec{v_i}(t-1))$$

The random vector $\vec{r}$ is created by drawing random numbers for each dimension and normalizing its length to s.

The function's complexity may easily be scaled by increasing the number of dimensions and/or the number of peaks, or by using complex peak- and base-functions.

Different environments have been generated with MPB according to standard settings given on its web page[1]:

- Scenario 1: fitness landscape is defined for the 5-dimensional search space with boundaries for each of dimensions set to $[0, 100]$. For such a domain there are five moving peaks which vary their height randomly within the interval $[30, 70]$, width within $[0.0001, 0.2]$ and position by a distance of 1.

[1] http://www.aifb.uni-karlsruhe.de/~jbr/MovPeaks/

- Scenario 2: fitness landscape is defined for a 5-dimensional search space with boundaries for each of dimensions set to $[0, 100]$. However in scenario 2 we have 5 configurations, which have 10, 20, 30, 40 and 50 moving peaks which vary their height randomly within the interval $[30, 70]$, width within $[1, 12]$ and position also by a distance of 1.

5.2. Configuration of KEPS+D

The description of the system given previously is general, and thus we need to be more specific about certain aspects. The first and most important is the metaheuristics that will configure the system. With both benchmarks we will use a system composed by three different metaheuristics - a Genetic Algorithm (GA), a Tabu Search (TS) and a Simulated Annealing (SA). All these algorithms were programmed by the authors. In particular, for the DKP we used the general schemes proposed in [43] (GA), [23] (TS) and [27] (SA). For the genetic algorithm we used a single point crossover, and a one bit mutation (adding or removing one item to/from the knapsack), and two possibilities for the selection - roulette wheel or tournament. Regarding trajectory based metaheuristics we consider as neighbors those solutions that can be obtained by adding or removing one item. Focusing on TS, we used a static tabu list (i.e. with an invariable size) and a diversification mechanism which restarts the search in an unexplored area. With regard to SA, we have used four different cooling schedules: square root, hyperbolic tangent, hyperbolic cosine and exponential. Finally to generate the solutions we used a random strategy. On the other hand, for the MPB we used implementations adapted to continuous problems, in particular: [28] (GA), [16] (TS) and [44] (SA).

The next aspect to be discussed is the knowledge extraction process followed by the design of the systems that will cope with each problem. As stated above, this process is performed only once and before system operation. Note that, although test problems are dynamic, the system has to learn which can be characterized from static problems. This is not a problem, because dynamic problems can be seen as a series of static problems, and we have previously expressed how the system can handle dynamic problems by reconfiguring itself after each change.

An important issue is how many instances should be used to perform the knowledge extraction process, since an insufficient number will worsen the system's behavior and too many instances will unnecessarily increase the training cost. After some trials, we have determined that 25 instances for each type and size is a good value. Following these considerations we generated 275 instances for DKP and 150 for MPB. Each of these instances was solved over 4,000 objective function evaluations using different parameter values combinations by each of the metaheuristics. With the generated information we filled the rough datasets and by means of their refinement obtained the processed datasets. Processed datasets were the input for data mining phase, whose outputs are the SVM models that control parameters and weights.

5.3. Configuration of the experiments

The available scenarios were configured to change after 5000 evaluations of the fitness function, and 100 problem changes occurred per run. Results show the average of 100 runs. To measure the performance of each strategy we use offline error [10], a widely accepted measure of performance for DOPs, which is the running average of the difference between the optimum values and the best solution encountered at any time:

$$offline\ error = \frac{1}{T}\sum_{t=1}^{T}(optimum - bestSolution)$$

For the cooperative strategies one can resort to parallel schemes if time is important, or one can simulate the parallelism in a one-processor computer. This is the strategy adopted here and the procedure is extremely simple. We construct an array of solvers and we run them using a round-robin scheme. This implementation uses an asynchronous communication mode that is simulated

as follows: solvers are executed during a certain number of evaluations of the objective function. This amount of evaluations is a random number in a given interval and after this period of time, information exchanges are performed. These steps are repeated until the stop condition, given in terms of objective function evaluations, is fulfilled. The use of the number of objective function evaluations to control the execution of the strategies is broadly extended as a way to perform fairer comparisons with existing techniques, because like that we can compare the results independently of the programming language used, the computer architecture or the ability of the programmer to optimize his code. However, we consider that the time can be considered as an important measure of performance, and in that sense we have to point out that the overhead derived from the execution of the coordinator is negligible and that the execution time of the cooperative strategies is always smaller than that of the slower individual metaheuristic for the same number of evaluations of the objective function.

All algorithms were programmed by the authors using Java and the tests were performed on AMD Opteron Dual Core with 8 GB RAM from UGRGrid cluster[2].

In each subsection, after showing the experimental results, we make an analysis of methods using statistical techniques, following the methodology of [22], which uses non-parametric tests. We use the Wilcoxon signed-rank test to compare two methods. This test is a non-parametric statistical procedure for performing pairwise comparisons between two algorithms. This is analogous to the paired t-test in non-parametric statistical procedures; therefore, it is a pairwise test that aims to detect significant differences between two sample means, that is, the behavior of two algorithms. When we compare multiple methods, we use the Friedman test and the Benjamini-Hochberg method [3] as post-hoc test. This last method is more powerful than the Bonferroni-Dunn test, Holm test and Hochberg method. The Friedman test is a non-parametric test equivalent to the repeated-measures ANOVA. Under the null-hypothesis, it states that the algorithms are equivalent, so a rejection of this hypothesis implies the existence of differences in the performance of all the algorithms studied. After this, a post-hoc test (we use the Benjamini-Hochberg method) could be used in order to find whether the control or proposed algorithm presents statistical differences with regard to the remaining methods in the comparison. Specifically, we will consider that there are significant differences among the observed results if we obtain a significance level below 0.02. In order to apply these statistical tests we used the tool R[3].

5.4. Results

5.4.1. Testing the usefulness of cooperation

The first tests performed were executed to ascertain how the cooperation among the different metaheuristics influences the performance obtained by the system, i.e. how many solution exchanges are necessary to obtain good results. The parameters of the system that control the number of communications between metaheuristics, and hence the cooperation, are the number of evaluations of the fitness function before cooperation starts and the $\alpha - cut$ which controls the firing of the rules. In these tests we have chosen different values for these two parameters in order to test how the cooperation influences in the performance of the system.

Tables 3 and 4 and Figures 4 and 5 show the results obtained for DKP and MPB respectively. Tables and figures show the average offline error for all previously mentioned instances, which is shown as the first value in each cell, its standard deviation, second value, and the average number of times the rule was fired during the execution. Note that as these are preliminary tests, averages are calculated from 10 runs.

[2]http://ugrgrid.ugr.es/
[3]http://www.r-project.org/

Table 3: Cooperation tests for DKP

		evaluations			
		100-150	500-600	800-900	1300-1500
$\alpha - cut$	0.01	3.00% (1.47) 2731	2.25% (1.27) 673	1.53% (1.03) 349	1.64% (1.01) 328
	0.1	3.04% (1.54) 2474	2.25% (1.30) 592	1.54% (1.06) 314	1.64% (1.03) 297
	0.5	3.08% (1.58) 1925	2.26% (1.27) 440	1.54% (1.03) 246	1.64% (1.03) 231
	0.65	3.09% (1.55) 1787	2.28% (1.33) 410	1.52% (1.04) 233	1.63% (1.00) 220
	0.9	3.09% (1.55) 1586	2.25% (1.30) 372	1.54% (1.06) 214	1.65% (1.05) 205

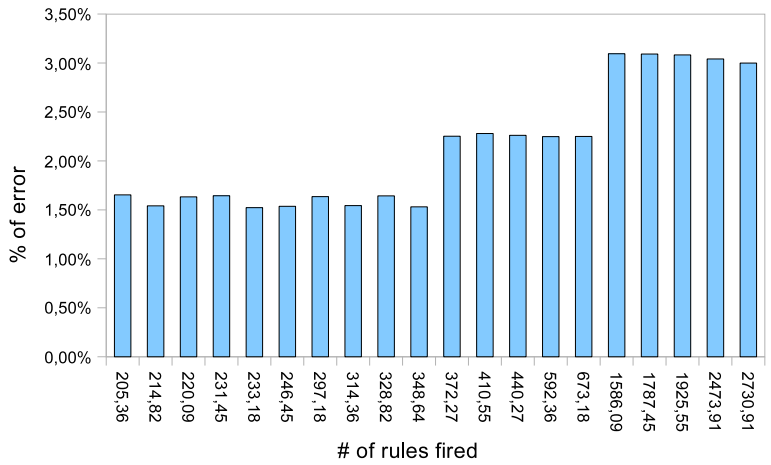

Figure 4: Cooperation tests for DKP

In both tables we can see a clear tendency: the cooperation rate should be neither too large nor too small. If we move from the left to the right of the table, decreasing the communication ratio, we see how offline error initially decreases until we reach a minimum for the interval $[800, 900]$ of objective function evaluations before cooperation starts, after which offline error increases. From these results we can infer that too much cooperation makes the system unstable, however, too little cooperation decreases performance as there is not enough information for the metaheuristics.

On the other hand the influence of the $\alpha - cut$ is not so clear as its impact in communication ratio is less, but it should be noted that for each interval of fitness evaluations the $\alpha - cut$ which obtains the best performance is different, which indicates that the $\alpha - cut$ is an important parameter to be taken into account. In the design of the strategy, the $\alpha - cut$ is defined by the user. In future work we will introduce a data mining process to obtain a set of appropriate models for the $\alpha - cut$. The strategy will use these new models to intelligently choose an $\alpha - cut$ value for each instance being solved.

Using the information of these experiments, we have set two important parameters of the system for the following tests. Thus, we have decided that every metaheuristic should stop for cooperation after a number of evaluations in the interval $[800, 900]$ for the two problems, an $\alpha - cut = 0.65$ for DKP and an $\alpha - cut = 0.5$ for MPB.

5.4.2. Results for DKP

In this section we will test the performance obtained by the system for the DKP. We compare its results with those obtained by the aforementioned individual metaheuristics and cooperative strategies. The results are shown in Table 5. In particular, in each cell of the table the average

Table 4: Cooperation tests for MPB

		evaluations			
		100-150	500-600	800-900	1300-1500
$\alpha - cut$	0.01	7.82% (1.83) 2638	5.27% (1.08) 677	4.02% (0.83) 388	4.21% (0.94) 358
	0.1	7.71% (1.96) 1964	5.46% (1.14) 587	3.81% (0.80) 325	4.20% (0.96) 289
	0.5	8.01% (1.73) 1455	5.36% (1.14) 482	3.77% (0.69) 262	3.92% (0.87) 233
	0.65	8.09% (1.48) 1415	5.27% (0.94) 461	3.94% (0.79) 245	4.12% (0.86) 221
	0.9	8.22% (1.56) 1370	4.25% (0.79) 331	4.02% (0.86) 228	4.07% (0.89) 205

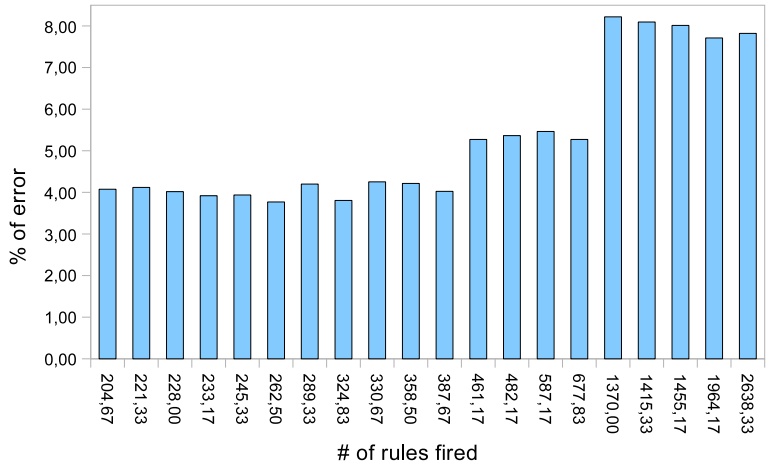

Figure 5: Cooperation tests for MPB

error value obtained by a strategy is presented, and the standard deviation is shown as a subscript. For example, in the first column and row we indicate that the Genetic Algorithm obtained an average error of 1.61%, with a standard deviation of 0.10 for an instance of the type Mori with a size of 30 items. Additionally, in Figure 6, we compare the results obtained by the best individual metaheuristic and the two cooperative strategies. In particular, in this graph the height of a bubble represents the average error and its width indicates the standard deviation.

Several conclusions can be drawn from the results. First, following the methodology of statistical tests indicated in subsection 5.3, we used the Friedman test to check the statistical differences among all strategies, obtaining significant differences with 99% confidence. After that, we apply a post-hoc test (Benjamini-Hochberg together with Wilcoxon test) to study which the differences among the strategies are. The confidence levels shown are those obtained after the whole process.

If we compare the performance of the individual metaheuristics we can observe how the one which obtained the best results is TS, statistically confirmed with a confidence level of 98%, followed by the GA, statistically better than SA with a confidence level of 99%, and the one which showed the worst behavior is the SA. These results are very interesting because the best results are obtained by a trajectory based metaheuristic which, as was previously mentioned, are traditionally discarded when the problem is dynamic. However, we can observe that TS does not obtain the best results for all instances. In that sense the use of a heterogeneous cooperative metaheuristic is justified, because each individual metaheuristic is able to contribute in the resolution of different instances.

Another interesting conclusion is that all cooperative strategies outperform the individual metaheuristics in almost every instance. Moreover their results are better statistically, with a confi-

Table 5: Results obtained solving DKP

type	size	GA	SA	TS	CS-WB	KEPS+D
Mori	30	1.61% (0.10)	2.87% (0.35)	3.33% (0.15)	0.75% (0.09)	0.42% (0.06)
	60	2.78% (0.09)	2.60% (0.16)	4.15% (0.20)	2.19% (0.14)	1.21% (0.07)
	90	3.96% (0.13)	5.77% (0.34)	3.52% (0.13)	2.79% (0.15)	1.81% (0.12)
Karaman	100	3.97% (0.11)	6.72% (0.22)	4.34% (0.18)	3.10% (0.20)	1.76% (0.11)
	200	7.42% (0.17)	11.01% (0.22)	4.69% (0.19)	4.11% (0.25)	2.72% (0.16)
	300	9.75% (0.16)	13.91% (0.20)	4.73% (0.19)	4.87% (0.31)	3.20% (0.18)
	400	11.71% (0.19)	18.11% (0.25)	4.78% (0.19)	5.00% (0.29)	3.18% (0.19)
Str Span	500	3.16% (0.07)	17.23% (0.68)	1.16% (0.71)	1.10% (0.14)	0.88% (0.10)
	1000	3.33% (0.05)	19.03% (0.46)	1.01% (0.29)	0.66% (0.10)	0.48% (0.09)
	1500	5.38% (0.06)	26.88% (0.69)	0.80% (0.35)	1.03% (0.10)	0.75% (0.08)
	2000	5.12% (0.06)	28.61% (0.41)	1.88% (0.55)	0.69% (0.08)	0.50% (0.05)

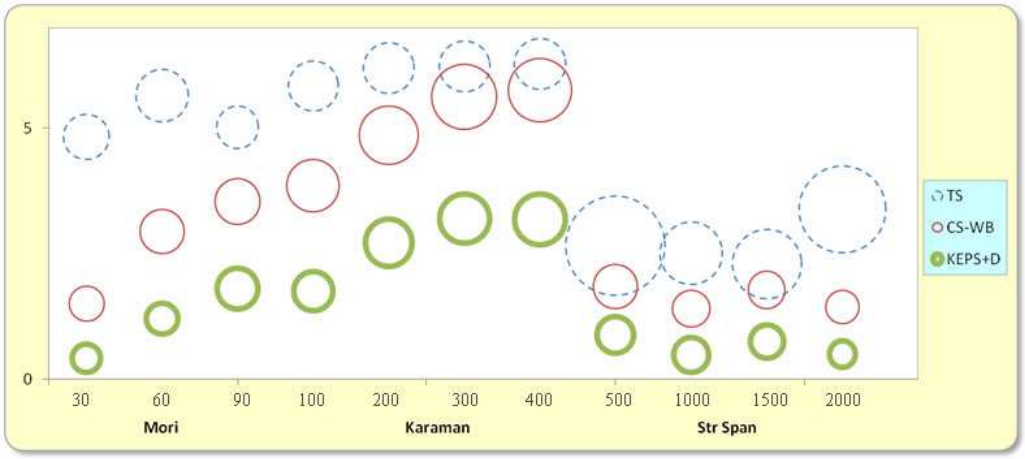

Figure 6: Bubble graph for DKP

dence level of 98%. This indicates that the cooperation introduced is useful for the resolution of this problem. Finally if we focus on the performance of the cooperative strategies, we can state with a confidence level of 98% that KEPS+D outperforms CS-WB, indicating that the intelligence introduced through knowledge models in KEPS+D is very useful. In fact, KEPS+D can outperform all other strategies on every instance.

5.4.3. Results for MPB

After testing our ideas with the DKP, we continue the experiments by testing the different approaches with MPB. The results are shown in Table 6. Once again, in each cell of the table the average error value obtained by a strategy is presented, and the standard deviation is shown as a subscript. For example, in the first column and row we indicate that the Genetic Algorithm obtained an average error of the 12.45% with a standard deviation of 1.74 for an instance of scenario 1 with 5 moving peaks. Once again, in Figure 7, we use a bubble graph to compare the average error and dispersion obtained by the best individual metaheuristic and the cooperative metaheuristics.

Continuing with the structure followed in the previous subsection, we used the Friedman test to check the statistical differences among all strategies, obtaining significant differences with 99% confidence. After that, we apply the post-hoc test to study what the differences among the strategies

Table 6: Results obtained solving MPB

scenario	# of peaks	GA	SA	TS	CS-WB	KEPS+D
scenario 1	5	12.45% (1.74)	14.85% (1.28)	7.03% (0.63)	6.00% (0.77)	4.01% (1.11)
scenario 2	10	10.89% (1.98)	13.07% (1.11)	7.66% (0.32)	5.17% (0.30)	4.56% (0.31)
	20	10.08% (1.86)	8.04% (0.73)	5.42% (0.29)	3.57% (0.26)	2.96% (0.23)
	30	13.51% (2.02)	9.85% (0.84)	7.80% (0.33)	5.37% (0.32)	4.53% (0.36)
	40	12.61% (2.15)	7.05% (0.63)	5.71% (0.38)	3.86% (0.21)	3.40% (0.26)
	50	11.31% (1.54)	7.29% (0.68)	6.12% (0.24)	4.24% (0.20)	3.92% (0.26)

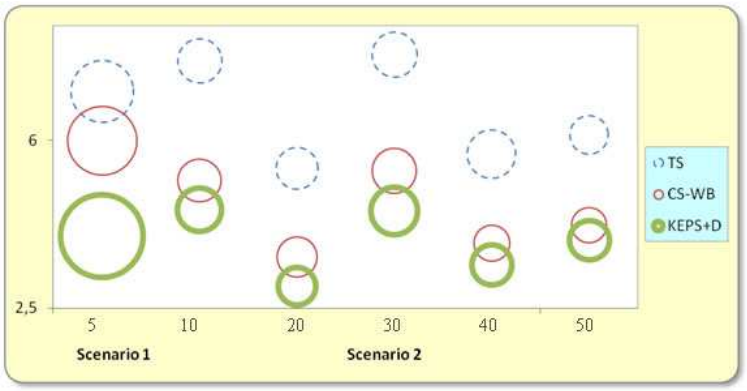

Figure 7: Bubble chart for MPB

are. The confidence levels shown are those obtained after the whole process.

As in the previous section we will start analyzing the results of the individual metaheuristics. In this case once again the one which showed the best performance is TS, corroborated by the statistical tests with 98% confidence level. The other two metaheuristics obtained results statistically equivalent, but SA seems to obtain better results. It is important to note that the best metaheuristic is TS, which is not a population based approach.

If we compare the performance of the individual metaheuristics with that obtained by the cooperative strategies, the cooperative strategies once more obtain a better performance statistically, corroborated with 98% confidence level. So we find again that the collaboration introduced effectively helps in the resolution of dynamic problems. The strategy which showed the best performance is KEPS+D, for all instances, with a confidence level of 98%. The use of models to control the choice of parameter values depending on instance state combined with the rule of solution interchange controlled by the historical performance of the individual metaheuristics is shown to help in these environments, demonstrating its capacity to adapt to the changing environments.

5.5. Comparing KEPS+D with Agents*[40]*

In this section we want to assess the validity of the proposed strategy by comparing it with a state-of-the-art strategy, as is Agents, proposed in [40]. This strategy is based on the joint use of two populations. The first is composed of a set of solutions arranged in a grid M. Each cell $M(i, j)$ contains: (1) a solution to the problem at hand; and (2) the corresponding cost of the solution. Adjacent solutions in the grid do not share any relation; they may represent completely different points in the search space.

The second population comprises a set of "agents" $A = \{a_1, a_2, \ldots, a_n\}$, where the number of agents is much lower than the size of M. Agents are able to move over the grid and change

the solutions stored in it. Using different cooperation and diversification methods, they obtain significant results.

In order to perform the comparison we have to carry out some new experiments, since the results presented in [40] focused on instances of MPB with 10 and 100 peaks and with a dimensional space of 5 or 10 dimensions.

Table 7: Comparison with *Agents*.

# of dimensions	# of peaks	Agents	KEPS+D
5	10	5.84% (8.3)	4.56% (0.31)
5	100	5.66% (5.31)	4.17% (0.25)
10	10	11.25% (12.44)	7.39% (0.53)
10	100	9.79% (9.09)	7.60% (0.69)

Table 7 shows the results obtained by both metaheuristics. As can be seen, the results obtained by KEPS+D are lower than those obtained by Agents, in fact statistically the results are better, with a confidence level of 98% and their typical deviation is much less. These results demonstrate the validity of KEPS+D for solving DOPs.

6. Conclusions and future work

In this paper we have presented a cooperative metaheuristic strategy, called KEPS+D, able to solve dynamic optimization problems. In KEPS+D different metaheuristics (a Genetic Algorithm, a Simulated Annealing and a Tabu Search) cooperate in trying to solve an instance of a problem under the supervision of a coordinator. The coordinator is able to adapt the behavior on the strategy depending of the state of the instance being solved. To do so, it uses previous knowledge and a fuzzy decision framework. In particular, the knowledge is obtained using a supervised learning process and coded using SVMs. This process is able to learn from historical data of executions of the individual metaheuristics. The coordinator uses the knowledge to choose the best parameters values for each metaheuristic, and also to configure a set of fuzzy rules which analyses the execution of the metaheuristics and decides if they need any information from the others.

To test the approach proposed, we have chosen two dynamic optimization problems, which are two of the most used in the literature - Dynamic Knapsack Problem and Moving Peaks Benchmark. The results obtained by the cooperative strategy are compared with those obtained by the individual metaheuristics that conform it and with two other cooperative strategies.

Before starting the comparisons, a study of the impact of cooperation in the performance of the cooperative strategy was carried out. This study indicated that an excess of cooperation can make the behavior of the strategy unstable as it can experience premature convergence problems while too little cooperation also decreases the performance of the system because the information available for each metaheuristic is limited. For that reason we finally chose an intermediate configuration which showed the best performance.

Next, we compared different strategies in order to test the proposed one. From these tests several conclusions can be drawn. First if we focus on the individual metaheuristics performance we have obtained some interesting results which suggest that the use of trajectory based metaheuristics could be an interesting research path which has not been thoroughly studied. This is supported by the fact that for both problems the metaheuristic which showed the best performance was tabu search. Another interesting result is that for both problems cooperative strategies obtained results significantly better than those obtained by the individual metaheuristics. This is important as it outlines the benefits that can be obtained from the collaboration of different metaheuristics, so adding robustness to the system. Finally the cooperative strategy which showed the best

performance was KEPS+D which is the one initially proposed and which uses the knowledge obtained from the historical performance of individual metaheuristics to adapt its parameters as the instance of problem changes. In this way it can change its behavior to obtain better results.

Acknowledgements

The authors thank the MICINN of Spain and the "Fondo Europeo de Desarrollo Regional" (FEDER) for their support given to this work under the project TIN2008-06872-C04-03. Thanks also to the Funding Program for Research Groups of Excellence with code 04552/GERM/06 by the "Fundación Séneca". E. Muñoz is supported by the scholarship program FPI from the "Fundación Séneca". And thanks to the University of Granada (Spain), for the computational resources provided by the use of the UGRGrid.

References

[1] F.C. Anderson, M.G. Pandy, Dynamic Optimization of Human Walking, *Journal of Biomechanical Engineering* 123, 381–390 (2001).

[2] D. Angus, T. Hendtlass, Dynamic Ant Colony Optimisation, *Applied Intelligence* 23, 33–38 (2005).

[3] Y. Benjamini, Y. Hochberg, Controlling the false discovery rate: a practical and powerful approach to multiple testing, *Journal of the Royal Statistical Society Series B* 57, 289–300 (1995).

[4] Z. Bingul, Adaptive genetic algorithms applied to dynamic multiobjective problems, *Applied Soft Computing* 7 (3), 791–799 (2007).

[5] T. Blackwell, P.J. Bentley, Dynamic search with charged swarms, *Genetic and Evolutionary Computation Conference*, Morgan Kaufmann, 19–26 (2002).

[6] T. Blackwell, J. Branke, Multiswarms, exclusion, and anti-convergence in dynamic environments, *IEEE Transactions on Evolutionary Computation* 10(4), 459–472 (2006).

[7] T. Blackwell, Particle Swarm Optimization in Dynamic Environments, *In S. Yang, Y.S.Ong, Y. Jin (Eds), Evolutionary Computation in Dynamic and Uncertain Environments*, Springer, 29–49 (2007).

[8] J. Branke, Memory enhanced evolutionary algorithm for changing optimization problems, *Proc. the Congress on Evolutionary Computation* 3, 1875–1882 (1999).

[9] J. Branke, T. Kauler, C. Schmidt, H. Schmeck, A multi-population approach to dynamic optimization problems, *Adaptive Computing in Design and Manufacturing: Selected Papers from ACDM00*, 299–308 (2000).

[10] J. Branke, *Evolutionary Optimization in Dynamic Environments*, Kluwer Academic Publishers, (2002).

[11] J.M. Cadenas, M.C. Garrido, L.D. Hernández, E. Muñoz, Towards a definition of a Data Mining process based on Fuzzy Sets for Cooperative Metaheuristic systems, *Proc. Conference Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU'06)*, 2828–2835 (2006).

[12] J.M. Cadenas, M.C. Garrido, E. Muñoz, Using machine learning in a cooperative hybrid parallel strategy of metaheuristics, *Information Sciences* 179(19), 3255–3267 (2009).

[13] J.M. Cadenas, M.C. Garrido, E. Muñoz,, B. Romera, Un sistema híbrido de metaheurísticas en entornos dinámicos. *Proc. Congreso Español sobre Metaheuristicas, Algoritmos Evolutivos y Bioinspirados (MAEB'09)*, 387–394 (2009).

[14] A. Carlisle, G. Dozier, Adapting Particle Swarm Optimization to Dynamic Environments. *Proc. International Conference on Artificial Intelligence*, 429–434 (2000).

[15] W. Cedeno, V.R. Vemuri, On the use of niching for dynamic landscapes, *Proc. the International Conference on Evolutionary Computation*, IEEE Press, 361-366 (1997).

[16] R. Chelouah, P. Siarry, Tabu Search Applied to Global Optimization, *European Journal of Operational Research* 123, 256–270 (2000).

[17] H.G. Cobb, J.J. Grefensttee, Genetic algorithms for tracking changing environments, *Proc. the Fifth International Conference on Genetic Algorithms*, Morgan Kaufmann, 523–530 (1993).

[18] D. Cohn, L. Atlas, R. Ladner, Improving Generalization with Active Learning *Maching Learning*, 15 (2), 201–221 (1994).

[19] T. G. Crainic, M. Toulouse, Parallel Strategies for Metaheuristics, *In F. Glover and G.A. Kochenberger (Eds.), Handbook of Metaheuristics*, Kluwer Academic Publisher, 475–513 (2003).

[20] W. Du, B. Li, Multi-strategy ensemble particle swarm optimization for dynamic optimization, *Information Sciences* 178, 3096-3109 (2008).

[21] R. Fabera, T. Jockenhvelb, G. Tsatsaronisc, Dynamic optimization with simulated annealing, *Computers & Chemical Engineering* 29(2), 273–290 (2005).

[22] S. García, A. Fernández, J. Luengo, F. Herrera, A study statistical of techniques and performance measures for genetics-based machine learning: accuracy and interpretability, *Soft Computing* 13 (10), 959–977 (2009).

[23] M. Gendreau, An Introduction to Tabu Search, *In F. Glover and G.A. Kochenberger (Eds.), Handbook of Metaheuristics*, Kluwer Academic, 2003.

[24] D.E. Goldberg, R.E. Smith, Non-stationary function optimization using genetic algorithms with dominance and diploidy. *Proc. the Second International Conference on Genetic Algorithms*, 59–68 (1987).

[25] J.R. González, A.D. Masegosa, I.J. García, A cooperative strategy for solving dynamic optimization problems *Memetic Computing*, DOI: 10.1007/s12293-010-0031-x.

[26] J. Grefenstette, Genetic algorithms for changing environments, *Parallel Problem Solving from Nature*, vol. 2, Elsevier, 465–501. 1993.

[27] D. Henderson, S.H. Jacobson, and A.W. Johnson, The Theory and Practice of Simulated Annealing, *In F. Glover and G.A. Kochenberger (Eds.), Handbook of Metaheuristics*, Kluwer Academic, 2003.

[28] F. Herrera, M. Lozano, E. Pérez, A.M. Sánchez, P. Villar, Mltiple Crossover per Couple with Selection of the Two Best Offspring: An Experimental Study with BLX-α Crossover Operator for Real-Coded Genetic Algorithms. *In F.J. Garijo, J.C. Riquelme, M. Toro (Eds.), Advances in Artificial Intelligence IBERAMIA 2002*, LNCS 2527, Springer Verlag, 392–401 (2002).

[29] S. Janson, M. Middendorf, A hierarchical particle swarm optimizer for dynamic optimization problems. *Applications of Evolutionary Computing*, LNCS, 513-524 (2004.

[30] L. Jourdan, C. Dhaenens, E.G. Talbi. Using datamining techniques to help metaheuristics: a short survey. *Hybrid Metaheuristics*, LNCS 4030, Springer Verlag, 57–69 (2006).

[31] A. Kamiya, K. Abiko, S. Kobayashi, Discussions of worker ants' rule-based CHC dealing with changing environments, *Applied Soft Computing*, 10(1), 245–250 (2010).

[32] A. Karaman, S. Uyar, G. Eryigit, The Memory Indexing Evolutionary Algorithm for Dynamic Environments *EvoWorkshops 2005*, LNCS 3449, Springer Verlag, 563–573 (2005).

[33] 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), 1685–1708 (2005).

[34] A.R. McKendall, J. Shanga, S. Kuppusamyb, Simulated annealing heuristics for the dynamic facility layout problem *Computers & Operations Research* 33(8), 2431–2444 (2006).

[35] S. Martello, D. Pisinger, P. Tohh, Dynamic programming and strong bounds for the 0-1 knapsack problem, *Management Science* 45, 414–424 (1999).

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

[37] N. Mori, H. Kita,Y. Nishikawa, Adaptation to a Changing Environment by Means of the Feedback Thermodynamical Genetic Algorithm, *Proc. Parallel Problem Solving from Nature*, 149–158 (1998).

[38] R. W. Morrison and K. A. DeJong, A test problem generator for non-stationary environments, *Proc. Congress on Evolutionary Computation* 3, 2047–2053 (1999).

[39] 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), 1849–1868 (2006).

[40] D. Pelta, C. Cruz, A study on diversity and cooperation in a multi-agent strategy for dynamic optimization problems, *International Journal of Intelligent Systems* 24(7), 844–861 (2009).

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

[42] J. Platt, Fast training of Support Vector Machines using Sequential Minimal Optimization. *In B. Schoelkopf, C. Burges, A. Smola (eds), Advances in Kernel Methods - Support Vector Learning*, Mit Press, 185–208 (1998).

[43] C. Reeves, *Genetic Algorithms*, *In F. Glover and G.A. Kochenberger (Eds.), Handbook of Metaheuristics*, Kluwer Academic (2003).

[44] B. Reusch, L. Nolle, A. Goodyear, A. Hopgood, P. Picton, N. Braithwaite, On Step Width Adaptation in Simulated Annealing for Continuous Parameter Optimisation, *Computational Intelligence. Theory and Applications*, LNCS 2206, Springer Verlag, 589–598, 2001.

[45] P. Rohlfshagen, X. Yao, The Dynamic Knapsack Problem Revisited: A New Benchmark Problem for Dynamic Combinatorial Optimisation *EvoWorkshops 2009*, LNCS 5484, Springer Verlag, 745–754, (2009).

[46] R. Schoonderwoerd, J.L. Bruten, O.E. Holland, L.J.M. Rothkrantz Ant-based load balancing in telecommunications networks, *Adaptive Behavior* 5(2), 169–207 (1996).

[47] S.K. Shevade, S.S. Keerthi, C. Bhattacharyya, K.R.K. Murthy, Improvements to the SMO Algorithm for SVM Regression. *IEEE Transactions on Neural Networks* 11(5), 1188–1193 (2000).

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

[49] R. Swaminathana, M.E. Pfund, J.W. Fowler, S.J. Mason, A. Kehab, Impact of permutation enforcement when minimizing total weighted tardiness in dynamic flowshops with uncertain processing times, *Computers & Operations Research* 34, 3055-3068 (2007).

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

[51] K. Trojanowski, S. T. Wierzchoń, Immune-based algorithms for dynamic optimization, *Information Sciences* 179(10), 1495–1515, (2009).

[52] R.K. Ursem, B. Filipic, T. Krink, Exploring the performance of an evolutionary algorithm for greenhouse control. *Journal of computing and information technology* 10(3), 195–201 (2002).

[53] Z. Zhang, Multiobjective optimization immune algorithm in dynamic environments and its application to greenhouse control, *Applied Soft Computing* 8, 959–971 (2008).