Proactive selection of metaheuristics based on knowledge of previous results

Alejandro Rosete-Suárez[1], **Mailyn Moreno-Espino**[1]

[1]Technical University of Havana, CUJAE, Havana, Cuba

{rosete,my}@ceis.cujae.edu.cu

Abstract

This paper presents two proactive algorithms that act as meta-metaheuristic agents: they decide which metaheuristic will be used to solve a new problem. These meta-metaheuristic agents operate in the environment of iterative work on optimizing problems, with the goal of selecting good metaheuristics to solve new problems. The information about previous results is converted into explicit knowledge that is used by the meta-metaheuristic agents to decide the most adequate metaheuristics. This proactive decision is based on a fuzzy vector that describes each problem. The proposal has been validated through experimentation with 28 functions on binary strings.

Keywords: metaheuristic, agents, proactive behavior

1. Introduction

Metaheuristics are popular optimization methods due to their ability to find good solutions (not necessarily optimal) to complex optimization problems in different domains [1].

Many metaheuristics have been proposed, e.g. Local Search, Great Deluge Algorithm, Genetic Algorithm, etc. According to the No Free Lunch (NFL) theorem, it is impossible to demonstrate that one metaheuristic outperforms all the others in all possible problems [2].

Several measures of problem difficulty (e.g. FDC [3]) have been proposed to learn which characteristics of a problem make it difficult for a given metaheuristic. These measures can also be used to characterize each problem and to select the best metaheuristic to solve it.

This paper is focused on developing meta-metaheuristic agents that behaves proactively, by choosing the metaheuristics that are supposed to be the best in new problems.

This proactive behavior is focused on the goal of the optimizer (the person who wants to optimize a new problem).

Section 2 explains the main concepts of agents and metaheuristics that are relevant to this paper. Section 3 presents the main components of the proposed meta-metaheuristic agents. Section 4 presents an experimental validation of the proposal in 28 functions on 100-bits strings. Section 5 presents the conclusions and discusses possible extensions.

2. Metaheuristics and Agents

2.1. Metaheuristics and measures

In recent years, the application of metaheuristics to solve complex optimization problems has become a very active research and development area. Talbi [1] defines two broad classes of metaheuristics: S-Metaheuristics and P-Metaheuristics. S-Metaheuristics are single-solution based metaheuristics, which use the current (single) solution as a reference, in order to generate new solutions by consecutive applications of the operators.

Local Search (LS) is the base of the genealogy of S-Metaheuristics. It generates new solutions by applying modifications to the best previous solutions. A new solution is only accepted as reference to generate new ones if it is better than the previous solution. This acceptance criterion may lead to converge to local (not global) optima, where the optimization is stagnated.

Many S-Metaheuristics have been designed to overcome this issue by relaxing the acceptance criterion and by considering some worse solutions as new references. For example, other S-Metaheuristics, such as Threshold Accepting (TA), Record-to-Record Travel RRT (RRT), and Great Deluge Algorithms (GDA) use a moderate acceptance criterion [1]. They accept some worse solutions by taking into account the quality of the new solution, and some other aspects and parameters.

Threshold Accepting (TA) accepts worse solutions that are not worse than the current solution by more than a given parameter called Threshold. Other two S-Metaheuristics take into account other aspects, instead of the current solution, e.g. Record-to-Record Travel (RRT) and Great Deluge Algorithm (GDA). RRT accepts worse solutions which are not worse than the best solution by more than a given parameter called Deviation. GDA accepts solutions which are better than a given parameter called Water Level, which is iteratively updated by adding a value given by other parameter called Rain.

On the other hand, P-Metaheuristics have a population of solutions that are used to generate new solutions. The best solutions of the population are selected to be transformed by operators (e.g. crossover, mutation, etc.) in order to produce new (possibly best) solutions. Some remarkable examples of P-Metaheuristics are Genetic Algorithms (GA) and Evolution Strategies [1].

According to the No Free Lunch (NFL) theorem, it is impossible to demonstrate that one metaheuristic outperforms all the others in all possible problems [2].

In spite of some general guidelines to adjust the parameters of each metaheuristics [1, 4], the best values depend on the problems, the operators, etc. This is also emphasized by the NFL theorem [2].

Many predictive measures have been proposed to understand and to predict the performance of metaheuristics, e.g. Fitness Distance Correlation (FDC) [3]. FDC computes the correlation between the fitness (evaluations of the solution using the objective function), and the distances (in terms of operators) between each solution and the global optimum. If FDC is near to -1, both characteristics vary in opposite directions, i.e., solutions near to the global optimum (small distance) are good too (high fitness). If FDC= -1 the problem may be easy, but if FDC= 1 it may be hard. As FDC needs the fitness values of all the solutions, it has been normally used to do post-mortem study of the performance of metaheuristics in certain controlled type of problems. Other measures of problem difficulty can be found in [3, 5-8].

In the case of continuous optimization some recent efforts have been developed to link landscape analysis measures and algorithm parameters to the performance of the algorithms using machine learning [9]. This model can be used to predict the algorithm performance when a new optimization problem is presented. The characterization of the problems is done in terms of some mathematical analysis of the functions on the continuous spaces, such as modality, smoothness and variable separability. The continuous space allows mathematical characterizations that are not available for discrete space. In [9] the information about the performance of the algorithm is not structured as knowledge to decide which metaheuristic is best for new problems.

This problem is very similar to the algorithm selection problem which has been extensively studied after its definition by Rice [8]. In the practice, an important aspect is to view this problem as "a form of time-saving to avoid future trial and error approaches to finding the best algorithm for a given instance" and it is a very important problem according to the discussion presented in [8].

2.2. Hyper-Metaheuristics and Portfolio Algorithm

In this section we briefly establish the relation between our proposal and some important concepts in modern optimization: hyper-heuristics and portfolio algorithms.

According to Burke [10], the term hyper-heuristic "has been defined to broadly describe the process of using (meta-) heuristics to choose (meta-) heuristics to solve the problem in hand". This author says "the key idea in hyper-heuristics is to use members of a set of known and reasonably understood heuristics to transform the state of a problem".

It is worth noting that many basic heuristic principles may be available to solve a problem and the idea is to use all of them in an intelligent way to produce a complex improved performance. The idea is to establish a high level of decision about the way of choosing and combining heuristics.

The same idea may be used to combine metaheuristics, which coincide with the interpretation of Talbi [1] who said that hyper-heuristics are related to "define adaptive cooperation mechanisms that allow to select dynamically the optimization methods according to convergence or other criteria such as diversity".

By the other hand, portfolio algorithms consist of a collection of algorithms that can be used in combination and collaboration in order to improve the final performance [11].

As in hyper-heuristics, the idea is to establish a high level of decision. In these cases, the objective is the adaptive change of the decision of the algorithm that will be used in each stage of the algorithm. This decision is based on the online performance during the optimization process.

Despite NFL state that it is not possible to get a metaheuristics with the best performance in all problems, this does not imply that it is impossible to identify the characteristics of a problem that are related to the performance of the metaheuristics. We can observe the repeated process of function optimization that an optimizer performs. That is, during the life of an optimizer many problems have been faced using different metaheuristics.

When a new problem is to be faced, a pessimistic approach is that no previous knowledge is useful, and that all available metaheuristic need to be executed in order to know which is the best. An optimistic approach is that if we characterize each problem in terms of some measures then we may use this knowledge of the previous problems to face new problems.

This knowledge may be used in an implicit way, as general principles or experiences in the head of the optimizer. Besides, it is possible to establish a framework where an agent acts in the place of the optimizer using the previous information to decide which metaheuristic must be used in the new problem.

The suffix meta means an upper level methodology, e.g. Glover introduced the term metaheuristic [12] as upper level general methodologies (templates) that can be used as guiding strategies in designing underlying heuristics to solve specific optimization problems.

Similarly, if we develop an upper level methodology that can be used as guiding strategies to use metaheuristics for solving specific optimization problems it may be called meta-metaheuristics.

A comprehensive review of many of the measures that can be used to characterize different optimization problem may be found in [5, 8].

It is worth noting that the name meta-metaheuristic was not previously used. More importantly, this kind of meta-metaheuristic algorithm (that try to characterize the problems in order to proactively decide which metaheuristics should be used) is a very important problem as it can be concluded from [8].

2.3. Agents and metaheuristics

The agent paradigm is a modern metaphor in software development. Two relevant characteristics of agents are autonomy and proactivity. Agents are able to act without direct intervention of humans, and are driven by goals [13]. This means that agents may be proactive, in the sense that they will take the initiative, i.e., decide by themselves what actions to perform in order to satisfy their goals.

This conducts to software requirements, not only in terms of what a system must do (late requirements) but also in terms of why it must be done (early requirements) [14]. This approach allows us to detect which goals of the users can be delegated to software, and which resources and capacities are necessary to this end.

Some agent based models of metaheuristics have been developed (e.g. [15-17]). Most of them rely on the social behavior characteristic of the agent paradigm, by defining agents that exchange messages to guide the search towards the most promising regions.

However, no proactive strategy to decide which metaheuristic should be used in a problem has been proposed.

This is a consequence of the absence of the explicit modeling of the goals around the use of metaheuristics.

Indeed, the history of the experiences of an optimizer have not been regarded as an environment to be observed and over which an agent may act proactively in order to reach the human goals, i.e. to get best results. It is important to note that this is the final goal of the optimizer.

3. Meta-Metaheuristics (MM)

We present two meta-metaheuristic (MM) agents to decide which metaheuristic should be used to solve new problems.

The optimizer gives a description of each problem in terms of some measures. The MM agents observe and record the performance of the available metaheuristics in each problem.

Then, to solve a new problem, each MM agent selects the old problem with the closest description. Finally, it uses the metaheuristic with best performance in the old problem to solve the new problem.

Our proposal can be seen as an instance of the algorithm selection problem [8], but in our case we do not try to predict the performance of each metaheuristic.

Besides, in the experiments of section 4 we are trying to face this problem without any previous information about the evaluation of the fitness function of the new problem. That implies that the measures that need to perform some evaluations of the new function will be avoided, i.e. the general problem-independent measures presented in [8] (such as FDC) will not be used.

3.1. Structure

Each meta-metaheuristic agent (MM) is a way of structuring the use of a collection of available metaheuristic. That is, each meta-metaheuristic agent selects the best metaheuristic for a new problem according to its performance in similar problems.

For several previous problems we have a collection of the metaheuristic with the best performance on each. That is, based on the previous history each MM has a set of pairs (D_i,B_i). Each pair (D_i, B_i) represents a previous problem. D_i is a vector that describes the i-th problem in terms of some measures. B_i is the set of identifiers of the best metaheuristics on the i-th problem.

Each MM agent uses this knowledge in the following way. When a new problem needs to be solved, the MM agent selects the old problem with the closest description to the new one. Then, one of the best performing metaheuristic in the old problem is selected to be used in the new one. Consequently, each MM agent has a portfolio of metaheuristics and it selects one of them to solve a new problem.

3.2. Problem description

Each vector Di (that describes a problem) is composed by fuzzy values that represent the degree to which several measures are satisfied by a problem.

Each measure may be a general measure such as FDC (that may be used to describe a general class of problems), or it may be a specific measure only available to certain classes of problems. Some examples of available measures can be found in [5, 8].

Each component of the vector D_i is a fuzzy value in order to simplify the comparison of new problems with old problems. The conversion of numerical values to fuzzy values may be implemented in different ways, according to the semantic of the measures. For example, a fuzzy label “high value of FDC” may be obtained via a linear normalization of FDC, i.e. assigning 0 to FDC=-1, 1 to FDC=1, etc. Other problem-specific measures, such as the number of cities in a routing problem may be converted (normalized) to fuzzy labels (e.g. “Many cities”) in a similar way.

In brief, each vector D_i represents a problem description, where each component of D_i is a fuzzy value that describes how much each measure is satisfied by the problem.

In section 4, we present an example of this idea in the context of block functions based on unitation [18]. The problem-specific measures presented in [8] are not suitable for block functions based on unitation. Then, we will introduce new problem-dependent measures for block functions.

3.3. Selection of the metaheuristic

The selection of the metaheuristic for a new problem is based on the comparison between the vector that describes the problem and the description of each old problem previously solved. To select the closest one, many strategies may be used, e.g. based in distance calculation as in case-based reasoning [19].

In our case we use two similarity indicators: dot (scalar) product and correlation. In both cases, these indicators are computed between the vector D_N describing the new problem and each vector D_i that describes each old problem.

First, we select the closest description D_C. Then, the new problem will be solved by one of the metaheuristics that are members of B_C (at random).

Consequently, each MM selects one of the metaheuristics with the best performance in the problem C with the closest description D_c. This implies the use of a metaheuristic that performs well on the closest old problem. The difference between the two MM agents that are proposed in this paper is based on the way they use to decide which of the old problems has the closest description to the new problem. In the rest of this paper we use the identifier MM-c for the MM agent based on correlation, and MM-d for the MM agent based on dot product.

4. Experiment and Discussion

4.1. Functions

We perform the experimentation with 28 100-bit binary coded functions. Each function consists of 25 copies of 4-bit building blocks with one optimum value of 100 when all bits are equal to 1. Each building block for the 28 functions is a unitation-based function. Four of them have been used before (e.g. [3, 18]): Deceptive, Onemax, RoyalRoad, and Plateau (Figure 1).

The unitation of a bit string is the number of bits that are equal to 1 in the bit string. The difference among the 28 functions is based on the inner building block functions, i.e. the value that each building block function assigns to each value of unitation.

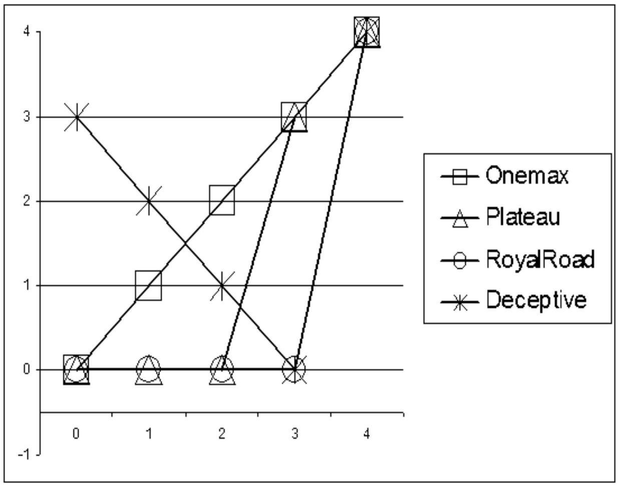

Figure 1. Some building blocks functions

For each possible value of unitation (i.e. 0, 1, 2, 3, and 4) of a block, each building block function returns a value of 0, 1, 2, 3, or 4. That is, a block function consists of a vector (u_0, u_1, u_2, u_3, u_4) of values for each unitation value. For example, the building block function Onemax may be expressed as the vector (0,1,2,3,4) because u_0=0, u_1=1, u_2=2, u_3=3, and u_4=4 (see Figure 1).

In [18] every building block function only returns 4 if the value of unitation is 4, i.e. u_4=4 for all functions. This implies that the only global optimum is the solution with 100 bits with 1 (every 4-bits block is 1111, with perfect unification value u_4=4). The different values that each building-block function returns for the rest of possible value of unification (i.e. u_0, u_1, u_2, and u_3) imply different building block functions. In the four building block functions of [18], only the values of 0, 1, 2, or 3 are used. As it can be seen, with the same structure of these four functions, it is possible to define 4^4=256 blocks functions.

Deceptive is supposed to be hard (FDC= 1), and Onemax is supposed to be easy (FDC= -1). We generate other 24 build block function with different values of FDC, in order to get a best sample of the 256 possible block functions. The 28 building block functions are shown in Table 1. In Table 1, each function fABCD represents the block function where u_0=A, u_1=B, u_2=C, and u_3=D. For example, functions f0123, f0003, f0000, are f3210 are Onemax, Plateau, RoyalRoad, and Deceptive, respectively.

It is important to note that the problem-specific measures provided in [8] are not oriented to block functions. This implies that we need to define measures to the block problem.

Building block functions			
f0000	**f0123**	f1120	f3000
f0001	f0131	f1221	f3001
f0003	f1001	f2000	f3002
f0022	f1002	f2001	f3010
f0111	f1012	f2012	f3012
f0120	f1111	f2200	f3102
f0122	f1112	f2220	**f3210**

Table 1. Functions used in the experiment

Table 2 summarizes the 28 building block functions by using average (Ave), minimum (Min), and maximum (Max) of some measures: BFDC, AveU, WAveU, and d(x,y).

BFDC computes FDC for each building block function using the possible unitation values. That is, the BFDC value of a block function is the correlation between two series: 0,1,2,3 and u_0, u_1, u_2, u_3. It is worth noting that BFDC is computed in the inner building block function (as the ones presented in Figure 1) with respect to the unitation values. This is different to the classical FDC [3, 8] which needs a complete evaluation of the search space composed by 2^{100} solutions.

AveU computes the average of the four values that the function returns for each value of unitation, i.e u_0, u_1, u_2, and u_3.

WAveU is weighted version of AveU. WAveU is the weighted average of u_0, u_1, u_2, and u_3, by taking into account how many possible block values are representative of each unitation value. For example, the weight of u_3 is 4 because the value of unitation 3 is represented by 4 block values (1110, 1101, 1011, and 0111). The weight of u_0 is 1 because the value of unitation of 0 is only represented by one block value (0000).

	Ave	Min	Max
BFDC	-0.03	-1	1
d(4,3)	2.64	1	4
d(4,2)	3.39	2	4
d(4,1)	3.11	1	4
d(4,0)	2.86	1	4
d(3,2)	0.75	-1	3
d(3,1)	0.46	-3	3
d(3,0)	0.21	-3	3
d(2,1)	-0.29	-2	2
d(2,0)	-0.54	-3	2
d(1,0)	-0.25	-3	2
AveU	1.00	0	1.5
WAveU	0.92	0	1.87

Table 2. Summary of the functions used

The measures d(x,y) computes the differences between the values that each function returns for the value of unitation u_x, and u_y.

For example, $d(3,1)=u_3-u_1$, where u_3 and u_1 are the values of the block function for the unitation value of 3 and 1, respectively. For instance, in f3210 (Deceptive) $u_3=0$ and $u_1=2$. Thus, in Deceptive the value $d(3,1)=u_3-u_1=0-2=-2$, because the difference between the value assigned for the unitation value of 3 (i.e. $u_3=0$) respect to the unitation value of 1 (i.e. $u_1=2$) is equal to -2 (i.e. -2=0-2).

These 28 functions have a good diversity in all these measures, as it can be observed in Table 2. It is worth noting that all the measures presented in Table 2 are problem-specific, i.e. they are particular measures for the block functions based on unitation. These measures have not been presented before.

4.2. Metaheuristics in comparison

Seven metaheuristics are used in the experiments. Four of them are S-Metaheuristic: TA, GDA, RRT, and LS. Three configurations of population-based metaheuristics (P-Metaheuristics) based on Evolution Strategies, and Genetic Algorithms are also compared.

The identifiers and parameters used for all these metaheuristics are the following.

- LS: Stochastic Local Search which accepts solutions with equal fitness, with 1-bit mutation.
- RRT: Record-to-Record-Travel, deviation D=5, 1-bit mutation.
- GDA: Great Deluge Algorithm, rain R=0.01, initial Water Level WL=0, 1-bit mutation.
- TA: Threshold Accepting, Threshold T=2, 1-bit mutation.
- GA: Genetic Algorithms, population size 100, steady-state replacement, truncation selection of the best 50 solutions, uniform crossover, probability of crossover: 1, 1-bit mutation, probability of mutation: 1.
- ES5: Evolution Strategies, population size 5, steady-state replacement, truncation selection of the best, 1-bit mutation, probability of mutation: 1.
- ES100: Evolution Strategies, population size 100, steady-state replacement, truncation selection of the best 20 solutions, 1-bit mutation, probability of mutation: 1.

These settings come from usual recommendations [1, 4] with manual adjustment in order to get the best results.

4.3. Problem descriptions

Each problem is described based on the 13 measures presented in Table 2. These measures are converted to fuzzy values by normalization: a fuzzy value of 1 is assigned to the maximum value of the measure, 0 to the minimum, etc.

If the value of BFDC is undefined (e.g. Royal Road) a central fuzzy value of 0.5 is assigned, because we believe that the absence of the value of correlation is closer to a null value of correlation than to an extreme value of -1 or 1. Finally, all vectors are normalized.

We use as history of old problems a selection of some functions from the 28 functions previously described. We conducted two experiments: Case A, and Case B, which differs in the selected functions. In future work, we believe that this aspect deserves a deep study because of its influence in the performance of MM.

Three examples of descriptions of three old problems (functions) of Case A, based on the 13 measures (vector Di) are presented in Table 3. The set of best-performing metaheuristics on each case (set Bi) is also included. For selecting the set of best metaheuristic in each function we execute 30 independent runs until 10000 fitness evaluations. The metaheuristic with the best average of fitness of the best solution in each run was selected as winner.

In Case A, ten functions are included as history: f0120, f1221, f3012, f0123, f0022, f0003, f0131, f3210, f3102, f2200. We do this selection manually trying to get representative values of problem descriptions Di. In these ten functions, the algorithm TA wins in 4 functions, LS and ES5 win in 3 functions, and GA and GDA win once. As can be inferred, ties are allowed, i.e. some Bi contains more than one member as in f0123 in Table 3.

In Case B, six functions are included as history: f0120, f1221, f3012, f1012, f0022, f3102. We do this selection manually trying to get representative values of best performing metaheuristics, i.e. Bi. In these six functions, only one algorithm wins in each case: TA, LS, ES5, ES100, GA and GDA. In this case there are not ties, i.e. each Bi contains one member.

Functions			
	f0120	f1221	f0123
Di: description			
BFDC	0.04	0.07	0.00
d(4,3)	0.22	0.13	0.43
d(4,2)	0.10	0.03	0.19
d(4,1)	0.02	0.03	0.05
d(4,0)	0.22	0.13	0.00
d(3,2)	0.02	0.03	0.05
d(3,1)	0.01	0.03	0.01
d(3,0)	0.06	0.07	0.00
d(2,1)	0.02	0.07	0.05
d(2,0)	0.10	0.13	0.01
d(1,0)	0.15	0.13	0.05
AveU	0.01	0.07	0.11
WAveU	0.02	0.09	0.06
Bi: best metaheuristics in each case			
Bi	{TA}	{GA}	{LS, ES5}

Table 3: Description of three old problems.

4.4. Results and discussion

In this section we compare nine algorithms: the two proposed versions of MM agents (i.e. MM-c, and MM-d), and the seven metaheuristics of section 4.2.

The comparison was conducted in the functions of Table 1 that were not selected in each case of section 4.3.

Case A	Ave Med	Min Max	SD	PC PB
MM-d	92.4 94.8	73.9 **100**	8.8	39 44
MM-c	**93.5** 94.8	77.7 **100**	7.2	39 44
LS	82.9 84.3	56.3 **100**	16.6	33 33
ES5	91.0 92.9	57.9 **100**	10.3	28 33
ES100	93.1 **96.6**	66.7 **100**	9.2	22 **50**
GA	91.2 91.2	**77.9** 98.3	**5.9**	0 0
GDA	80.7 85.9	53.9 98.4	15.6	0 6
TA	81.0 87.7	51.0 **100**	20	**44** 44
RRT	76.8 77.7	56.5 88.1	9.4	0 0

Table 4: Overall performance on the 18 test functions of Case A.

In Case A, the experiment was conducted in the set of test functions composed by the remaining 18 functions (i.e. 18=28-10).

In Case B, the experiment was conducted in the set of test functions composed by the remaining 22 functions (i.e. 22=28-6).

For each experiment of the 9 algorithms on the test functions, 30 independent runs were executed until 10000 fitness evaluations.

The best solution found on each run was registered. These 30 results of each metaheuristic in each test function were summarized using the average, and these results were used as primary data to the summary of overall performance given in Table 4, and Table 5.

This summary of the overall performance in the test function is shown in terms of seven performance indicators: average (Ave), median (Med), minimum (Min), maximum (Max), standard deviation (SD), the percentage of convergence (PC, i.e. how many times the algorithm get the perfect result of 100 in all runs), and percent of best performance (PB, i.e. how many times the algorithm get the best performance amongst the algorithms in comparison).

Case B	Ave Med	Min Max	SD	PC PB
MM-d	92.5 **100**	56.5 **100**	14	**59** **68**
MM-c	**95.2** 98.8	**77.9** **100**	**6.4**	41 50
LS	83.1 84.3	56.3 **100**	16.6	36 36
ES5	90.2 92.9	57.9 **100**	10.8	32 36
ES100	91.7 94.7	66.7 **100**	9.3	14 36
GA	90.4 91.1	76.9 99.8	6.7	0 0
GDA	82.7 85.9	53.9 99.7	15	0 5
TA	83.8 94	51 **100**	19.3	50 50
RRT	78.4 80.1	56.5 88.1	8.9	0 0

Table 5: Overall performance on the 22 test functions of Case B.

There is not an absolute winner algorithm in both cases, according to the results in Table 4 and 5.

In Case A, MM-c is the best in Ave, ES100 is the best in Med, GA is the best in Min and SD (the least is the best in SD), TA is the best in PC, and ES100 is the best in PB.

In Case B, MM-c is the best in Ave, Min, and SD. MM-d is the best in Med, PC, and PB.

It is worth noting that some algorithms get good results in PC and PB, but their results in average and median are very bad. A relevant exception of this rule is the case of the MM agents in Case B, where both algorithms are very good in almost all the seven performance indicators.

Table 6 and 7 give another view of the information in Table 4 and 5. Results in Table 4, and 5 may be sorted according to the results. Consequently, each algorithm gets a rank value (between 1 and 9) in each of the seven performance indicators depending on its relative position in each performance indicator, e.g. in Case A, the algorithm ES100 gets 1 in Med, GA get a 1 in SD (the smallest deviation is the best), MM-c get a 1 in Ave, etc.

Case A	1	2	3	Ave	Med
MM-d	1	3	3	2.3	**2.0**
MM-c	2	5	0	**1.7**	**2.0**
LS	1	0	0	5.6	6.0
ES5	1	0	0	4.4	5.0
ES100	3	1	0	2.7	**2.0**
GA	2	0	0	4.9	5.0
GDA	0	0	0	7.3	7.0
TA	2	1	0	5.0	6.0
RRT	0	0	0	7.6	8.0

Table 6: Ranking of performance in Case A.

Case B	1	2	3	Ave	Med
MM-d	4	1	0	2.4	**1.0**
MM-c	4	2	1	**1.6**	**1.0**
LS	1	0	0	5.6	7.0
ES5	1	0	0	4.1	5.0
ES100	1	0	3	3.4	3.0
GA	0	2	0	5.1	6.0
GDA	0	0	0	7.4	7.0
TA	1	2	0	4.7	4.0
RRT	0	0	1	7.1	8.0

Table 7: Ranking of performance in Case B.

Table 6 and 7 show how many times each algorithm gets a rank value of 1, 2, or 3 in the performance indicators on each case. Columns Ave and Med summarize the seven rank values of the algorithms in terms of average and median.

MM-c is the best algorithm according to the results in Tables 6 and 7. MM-c has a very robust performance, with rank values of 1 or 2 in almost all the seven indicators in both cases. MM-d is in the second place according to this analysis. This conclusion may also be supported by the values in terms of average and median of these rank values (columns Ave and Med of Table 6 and Table 7).

Finally, Table 8 summarizes the results in cases A and B in a similar way to Table 6 and 7, integrating the 14 values of performance indicators of both cases. These results confirm that MM-c is the algorithm with best overall performance.

In addition, the 14 values of the indicators for each algorithm were considered as primary data for doing a Friedman test [20]. We obtain a p-value of 0 that implies that the general performances of the algorithms are different. Column F of Table 8 shows the average rankings of the algorithms in the Friedman test.

These results shown in column F also confirm that MM-c is the algorithm with the best overall performance. Consequently, MM-c can be considered as the control algorithm to perform other statistical tests. We use the results of the Friedman test to perform Holm's and Hochberg's procedures [20] for comparing the control algorithm MM-c with the rest of the algorithms. According to both test, we obtain that MM-c is significantly better than GA, LS, TA, GDA, ES5, and RRT with respect to the overall comparison in terms of the 14 performance indicators.

Cases A and B	1	2	3	Ave	Med	F	W
MM-d	5	4	3	2.4	**2**	2.89	5
MM-c	6	7	1	**1.6**	**2**	**2.18**	**7**
LS	2	0	0	5.6	6.5	6.04	-3
ES5	2	0	0	4.3	5	4.75	0
ES100	4	1	3	3.1	3	3.50	2
GA	2	2	0	5	5.5	5.21	0
GDA	0	0	0	7.4	7	7.50	-4
TA	3	3	0	4.9	5	5.32	-2
RRT	0	0	1	7.4	8	7.61	-5

Table 8: Ranking of overall performance.

In addition, we use these 14 values of each algorithm as primary data for doing a Wilcoxon test [20] with significance of 0.05 between each pair of the 9 algorithms.

All the 36 pair-wise comparisons were conducted. When a metaheuristic is better than another according to the Wilcoxon test, the winner receives +1 and the looser receive -1. The sum of all these pair-wise comparisons using the Wilcoxon test is presented in column W of Table 8.

This column shows that MM-c is better than almost all the other algorithms, based on the Wilcoxon test. MM-c gets a score of 7 in column W because it outperforms 7 algorithms (i.e. LS, TA, GDA, ES100, ES5, GA, and RRT) according to the Wilcoxon test.

MM-d is the only algorithm that it is not outperformed by MM-c according to the Wilcoxon test. MM-d outperforms 5 algorithms (i.e. LS, TA, GDA, ES5, and RRT).

It is worth noting that despite the results are very similar in both cases (case A and Case B), the superior performance of MM-c is more evident in Case B. Case B uses only 6 functions as history but it seems that these 6 functions allow a best decision of the metaheuristic to be used for solving a new problem.

This result suggests that the selection of the history of old problems that is available for the MM agent is important to the performance of MM.

It is worth noting that in this paper we introduced several problem-dependent measures to characterize the problems to be solved. We had to do this because we did not find problem-dependent measures for the block functions based on unitation.

In the future, it is possible to extend our experiments to other problems (e.g. traveling salesman, knapsack, etc.) by using some of the problem-dependent measures that have proposed for them [8].

The results of the proposed meta-metaheuristic agents allow us to say that it is possible to proactively select the best metaheuristic based on the knowledge gathered from previous optimization history.

It also confirms that proactivity is a promising line of research in the field of metaheuristics.

5. Conclusion

This paper has introduced two proactive meta-metaheuristic agents which use the information of the environment (history of previously solved problems) in order to take actions (decision of the metaheuristic to be used for solving new problems).

These actions are oriented towards the goal of the human optimizer: to solve each new problem with the most suited metaheuristic amongst all the available metaheuristics in a portfolio.

The results in 28 100-bit binary coded functions show that the proposed meta-metaheuristic agents obtain good results in comparison to the available metaheuristics in the portfolio.

In order to characterize the problems that we used in the comparison, we introduced several problem-dependent measures. It is very easy to extend our proposal to other problems if it possible to define measures to characterize them.

6. References

[1] E. G. Talbi, “Metaheuristics: From Design to Implementation”, John Wiley & Sons, Inc., New Jersey, 2009.

[2] D. H. Wolpert, W. G. Macready, “No Free Lunch Theorems for Optimization”, IEEE Transactions on Evolutionary Computation, 1(1), pp. 67-82, 1996.

[3] T. Jones, S. Forrest, “Fitness distance correlation as a measure of problem difficulty for genetic algorithms”, Proc. of the 6th International Conference on Genetic Algorithms, pp. 184-192, 1995.

[4] M. Birattari, “Tuning Metaheuristics: A Machine Learning Perspective”, Springer, 2009.

[5] B. Naudts, L. Kallel, “A Comparison of Predictive Measures of Problem Difficulty in Evolutionary Algorithms”, IEEE Transactions on Evolutionary Computation, 4(1), 2000.

[6] G. R. Raidl, J. Gottlieb, “Empirical Analysis of Locality, Heritability and Heuristic Bias in Evolutionary Algo-rithms: A Case Study for the Multidimensional Knapsack Problem”, Evolutionary Computation, 13(4), pp. 441-475, 2005.

[7] M. Tomassini, L. Vanneschi, P. Col-lard, M. Clergue, “A Study of Fitness Distance Correlation as a Difficulty Measure in Genetic Programming”, Evolutionary Computation, 13(2), pp. 213-239, 2005.

[8] K. Smith-Miles, L. Lopes, “Measuring instance difficulty for combinatorial optimization problems”, Computers & Operations Research, 39, pp. 875–889, 2012.

[9] M. A. Muñoz-Acosta, M. Kirley, S. K. Halgamuge, “A Meta-Learning Pre-diction Model of Algorithm Performance for Continuous Optimization Problems”, Proc. of Parallel Problem Solving from Nature - PPSN XII, 2012.

[10] E. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, S. Schulenburg, “Hyper-Heuristics: An emerging direction in modern search technology”, Handbook of Metaheuristics. LNCS, F. Glover, G. A. K. (eds.), pp. 457-474, Kluwer Academics, 2003.

[11] B. S. Carla, P. Gomes, “Algorithms portfolio design: Theory vs. practice”, Interfaces in computer science and operations research, 2009.

[12] F. Glover, “Future Paths for Integer Programming and Links To Artificial Intelligence”, Computers & Operations Re-search, 13(5), pp. 533-549, 1986.

[13] M. Wooldridge, “An Introduction to MultiAgent Systems”, John Wiley & Sons, 2009.

[14] E.Yu, “Social Modeling and i*”, Conceptual Modeling: Foundations and Applications. LNCS, A. Borgida, V. Chaudhri, P. Giorgini, E. Yu (eds.), pp. 99-121. Springer-Verlag, 2009.

[15] M. E. Aydin, “Coordinating meta-heuristic agents with swarm intelligence”, Journal of Intelligent Manufacturing, 23(4), pp. 991-999, 2012.

[16] J. R. González, C. Cruz, I. G. del Amo, D. A. Pelta, “An adaptive multi-agent strategy for solving combinatorial dynamic optimization problems”, Proc. of Nature Inspired Cooperative Strategies for Optimization, pp. 41–55. Studies in Computational Intelligence, 2012.

[17] R. Malek, “Collaboration of Meta-heuristics Algorithms through a Multi-Agent System”, Proc. of HoloMAS, LNCS, vol. 5696, V. Marik, T. Strasser, A. Zoitl, (eds.), pp. 72-81. Springer, 2009.

[18] H. Wang, D. Wang, S. Yang, “A memetic algorithm with adaptive hill climbing strategy for dynamic optimization problems”, Soft Computing- A Fusion of Foundations, Methodologies and Applications, 13(8-9), pp. 763-780, 2009.

[19] S. K. Pal, C. Simon, K. Shiu, “Foundations of soft case-based reasoning”, John Wiley & Sons, Hoboken, New Jersey, 2004.

[20] S. García, D. Molina, M. Lozano, F. Herrera, “A study on the use of non-parametric tests for analyzing the evolutionary algorithms’ behaviour: a case study”, Journal of Heuristics, 15(6), pp. 617–644, 2009.