Applying Copeland Voting to Design an Agent-Based Hyper-Heuristic

Vinicius Renan de Carvalho
Intelligent Techniques Laboratory
Computer Engineering Department
University of São Paulo (USP)
vrcarvalho@usp.br

Jaime Simão Sichman
Intelligent Techniques Laboratory
Computer Engineering Department
University of São Paulo (USP)
jaime.sichman@poli.usp.br

ABSTRACT

Meta-heuristics are algorithms which are applied to solve problems when conventional algorithms can not find good solutions in reasonable time; evolutionary algorithms are perhaps the most well-known examples of meta-heuristics. As there are many possible meta-heuristics, finding the most suitable meta-heuristic for a given problem is not a trivial task. In order to make this choice, one can design hyper-heuristics. In the literature, one can find some agent-based research whose focus is to propose a framework where meta-heuristics are considered as agents, that solve a given problem in a collaborative or competitive way. Most of these works focus on mono-objective meta-heuristics. Other works focus on how to select multi-objective meta-heuristics, but not using an agent-based approach. We present in this work an agent-based hyper-heuristic for choosing the most suitable evolutionary meta-heuristic for a given problem. Our approach performs a cooperative Copeland voting procedure, considering five different metrics, to define which one of three competitive evolutionary meta-heuristics should execute during a certain processing time. We use the Walking Fish Problem (WFG) suite with two and three objectives to analyse the proposed approach performance. The obtained results showed that in all cases our strategy found the most indicated evolutionary algorithm and gets competitive results against the state of art.

CCS Concepts

•Computing methodologies → Multi-agent systems;

Keywords

Meta-heuristics, Hyper-heuristics, Evolutionary algorithms, Self-organization, Complex systems, Analysis of agent-based simulations, Agent cooperation.

1. INTRODUCTION

Heuristics are basic approximate algorithms that search the solution space to find a good solution [5]. Meta-heuristics are algorithms used to solve problems when there isn't any problem-specific algorithm that gives a satisfactory solution.

Appears in: *Proc. of the 16th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2017), S. Das, E. Durfee, K. Larson, M. Winikoff (eds.), May 8–12, 2017, São Paulo, Brazil.*

Thus, usually, heuristics are specialized in solving problems for one particular domain, while meta-heuristics are more generic and adaptive in several domains.

Because of meta-heuristics generality, this kind of algorithm is widely used to solve complex problems in industry and services, in areas ranging from finance to production management and engineering [6].

Evolutionary algorithms are meta-heuristics which employ Darwin's theory of the survival of the fittest as their inspiration. This kind of algorithms generates solutions using the crossover and mutation heuristic operators and applies a fitness function to choose which solution will compose the next population.

There are many different evolutionary algorithms available in the literature, and they can be divided into two classes: single-objective evolutionary algorithms, such as Genetic Algorithm (GA) [18], and multi-objective evolutionary algorithms (MOEA), like Non-Dominated Sorting Genetic Algorithm II (NSGA-II) [14], Strength Pareto Evolutionary Algorithm 2 (SPEA2) [38] and Indicator-Based Evolutionary Algorithm (IBEA) [37].

However, different evolutionary algorithms produce different results when applied to different problems. Therefore, techniques able to choose the most suitable evolutionary algorithm for a given problem can solve this difficult task from the user, thus diminishing part of his tuning effort.

Hyper-heuristics are meta-algorithms that can be used to reduce the difficulty of selecting the most suitable meta-heuristic for a given problem. According to Burke et al. [8], hyper-heuristics are considered both as (i) meta-heuristic selection methodologies (they help to choose meta-heuristics) and (ii) meta-heuristic generation methodologies (they can generate new meta-heuristics from a given set of components). Some works in the domain focus on selecting the most suitable (meta-)heuristic. However, the majority of research in this area has been limited to treat single-objective optimization problems [24].

This present work proposes the **MOABHH** (Multi-Objective Agent-Based Hyper-Heuristic) framework, a multi-agent-based hyper-heuristic focused on finding the best evolutionary algorithm among a set of algorithms. In our model, there are three kinds of agents: *Multi-objective Evolutionary Algorithm (MOEA)* agents, responsible for finding solutions for the optimization problem; (*Indicator voter*) agents, who perform a Copeland election and are responsible for evaluating the performance of each MOEA agent, according to their own particular quality indicator; and a *hyper-heuristic* agent, which decides the share of the population that should be treated by each MOEA agent. In our approach, no additional population or archive is employed. Thus, there is only one population shared among all evolutionary algorithms. This choice makes it easier to compare our approach with pure evolutionary algorithms.

The remainder of this text is organized as follows: Section 2 presents an overview of evolutionary algorithms, multi-objective evolutionary algorithms (MOEAs) and indicators. In Section 3, we give a brief explanation about Copeland voting method. Our approach is detailed in Section 4. In Section 5, we explain our experiments, and their results are shown in Section 6, when we compare our approach with single evolutionary algorithms. Related work is discussed in Section 7. Finally, in Section 8 we present our conclusions and further work.

2. EVOLUTIONARY ALGORITHMS

Evolutionary algorithms can be classified, according to the number of fitness values, as mono-objective evolutionary algorithms or multi-objective evolutionary algorithms.

Mono-objective evolutionary algorithms use one value fitness functions to represent a solution quality. Applications of this kind can be seen in finding the minimal (or maximal) value for a mathematical function with one output. Instead, Multi-objective Evolutionary Algorithms (MOEAs) employ a set of objective values. An example of application is to provide solutions for choosing a car to buy, considering simultaneously max speed, fuel consumption, and average market price.

Multi-objective Evolutionary Algorithms usually employ Pareto Dominance [27] to find solutions for multi-objective optimization problems. Differently from mono-objective approaches, where there is just one final solution, in a multi-objective approach there are a set of solutions, which do not dominate one another. Thus, the user can choose from the set the solution that he finds most suitable.

In Pareto dominance, a certain solution A in the decision space of a multi-objective problem is superior to another solution B if and only if fitness(A) is at least as good as fitness(B) in terms of all the objectives and strictly better than fitness(B) in terms of at least a single objective. Solution A is also said to strictly dominate solution B [3].

In the experiment described in Section 5, we use the well known Walking Fish Group (WFG) [20] benchmark, aimed to test multi-objective optimization algorithms, for problems with two and three objectives.

We selected as MOEA agents the algorithms NSGAII, SPEA2, and IBEA. As quality indicators for the *Indicator voters*, we choose the Hypervolume [39], IGD [40], GD [31], Spread [31] and RNI indicators [33]. These evolutionary algorithms and quality indicators are described in the sequence.

2.1 Multi-objective Evolutionary Algorithms

We present next three different MOEAs. All of them employ Pareto dominance, crossover, and mutation, but they differ in the way they choose the solutions for the new population:

NSGA-II The Non-Dominated Sorting Genetic Algorithm II [14] builds a population of competing solutions, ranks and sorts each solution according to non-domination level, applies evolutionary operations to create a new pool of offspring, and then combines the parents and offspring before partitioning the new combined pool into fronts [9]. The new combined pool is employed to generate a new main population. This is possible by calculating the *Crowding Distance*. The *Crowding Distance* value of solution x is calculated by the Euclidean distance between x and its neighbors. This metric privileges more spread solutions for more space exploration.

SPEA2 Strength Pareto Evolutionary Algorithm 2 [38] is an algorithm which uses the *Strength* value for selecting new solutions for the new population. This algorithm also employs the use of an additional population called external archive, to store non-dominated solutions found along the execution. This algorithm incorporates a fine-grained fitness assignment strategy, which considers for each solution the number of solutions that it dominates and that it is dominated by. It uses a nearest neighbor density estimation technique in order to increase the efficiency of the search [24]. This fitness function is named as solution *Strength*.

IBEA The Indicator-Based Evolutionary Algorithm [37] is an algorithm which focuses on maximizing the population quality according to a previously determined indicator. This indicator is used to calculate a contribution of a solution; this means how much the quality increases if this solution is kept in the population. Thus, the algorithm can compare different contributions to add to the new population the solutions which contribute more. One of most known indicators used together IBEA is the Hypervolume [39], as explained next.

2.2 Quality Indicators

MOEAs usually returns a set of solutions (S) when the execution is finished. Different indicators may be used to evaluate the quality of the outcome population; some of them are presented next:

RNI The ratio of non-dominated solutions [33] evaluates the percent of non-dominated solutions in the population, as shown in Equation 1. Higher RNI values are better than lower ones.

$$RNI(S) = \frac{|NonDominated(S)|}{|S|} \quad (1)$$

Hypervolume The hypervolume [39] of a non-dominated solution set S is the size of the part of objective space that is dominated collectively by the solutions in S [35]. Thus, the hypervolume indicator computes the area (or volume when more than two objectives are employed) in the search space [39]. Equation 2 presents how to calculate this indicator, where v_i is the volume. Higher hypervolumes are preferred to lower ones.

$$HV(S) = volume(\cup_{i=1}^{|S|} v_i) \quad (2)$$

GD The Generational Distance [31] corresponds to the sum of the Euclidean distances between the outcome population of solutions S and the solutions in Pareto Front

P. In general, it is impossible to find all solutions on a continuous Pareto Front [21]; in this case, the Pareto Front is previously known. Equation 3 presents how to calculate the GD indicator, where $q = 2$ and d is the Euclidean distance from $i \in S$ to $q \in P$. Lower GD values are better than higher ones.

$$GD(S,P) = \frac{(\sum_{i=1}^{|S|} d_i^q)^{\frac{1}{q}}}{|S|} \quad (3)$$

IGD The Inverted Generational Distance [40] is very similar to the GD. Instead calculating the Euclidean distance from S to P, as GD does, this indicator calculates the Euclidean distance from P to S. Equation 4 presents how to calculate the IGD indicator, where we have also $q = 2$. Similarly to the GD indicator, lower values are preferred to higher ones.

$$IGD(S,P) = \frac{(\sum_{i=1}^{|P|} d_i^q)^{\frac{1}{q}}}{|P|} \quad (4)$$

Spread The Spread [31], or Δ metric, evaluates the distribution of non-dominated solutions over the Pareto Front [24]. Equation 5 presents how to calculate the Spread indicator, where d_i is the Euclidean distance between solutions in sequence, $\overline{d}$ is the distance mean of d_f, d_l are the minimum Euclidean distances from solutions in S to the extreme (bounding) solutions of P [21]. Higher spreads are preferred to lower ones.

$$\Delta(S,P) = \frac{d_f + d_l + \sum_{i=1}^{|S|-1} |d_i - \overline{d}|}{d_f + d_l + (|S|-1)\overline{d}} \quad (5)$$

3. COPELAND VOTING METHOD

The Copeland Voting method [10] is a Condorcet method. Condorcet's principle says that should a candidate defeat every other candidate in pairwise comparisons (a Condorcet winner), it must be elected [26]. In order to follow this principle, Copeland's method ranks the alternatives according to their score in the sum of rows in the antisymmetric matrix of the Condorcet relation [28].

To perform a Copeland voting, first we create a pairwise competition between candidates. Considering two candidates c_i and c_j, the pairwise is calculated according to Equation 6.

$$S(i,j) = \left\{ \begin{array}{l} 1 \text{ if } c_i \text{ is better than } c_j \\ -1 \text{ se } c_i \text{ is worse than } c_j \\ 0 \text{ otherwise} \end{array} \right\} \quad (6)$$

After calculating all pairwise comparisons, for each c_i we find its Copeland score according to Equation 7.

$$CS(i) = \sum_{i \neq k} S(i,k) \quad (7)$$

Posteriorly, all Copeland scores can be sorted from higher to lower in order to generate a ranking. The selected candidate is the top ranked candidate.

4. MOABHH FRAMEWORK

MOABHH (Multi-Objective Agent-Based Hyper-Heuristic) is an agent-based hyper-heuristic framework focused on automatically selecting the most suitable evolutionary algorithm for a given multi-objective optimization problem. In this approach, there are three kinds of agents, who share four kinds of *artifacts* [30].

Figure 1 shows the interaction between MOABHH agents and artifacts, where solid arrows mean writing permission and dotted lines mean reading permission.

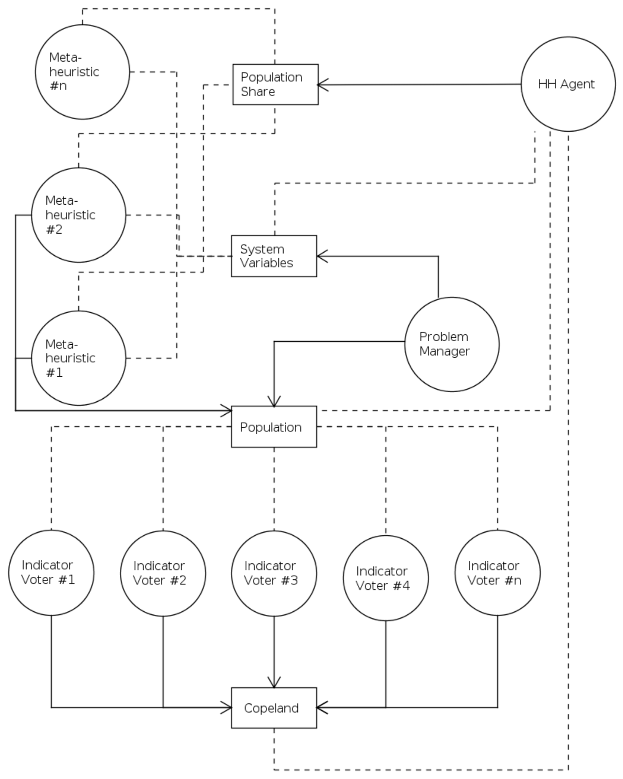


Figure 1: MOABHH modules interaction. Problem manager agent can write on System variables and Population artifact. MOEA agents can read Population and System Variables artifacts and write on Population Artifact. Indicator Voters can read population artifact and write on Copeland Artifact. HH Agent can read all artifacts and write on Population Share.

4.1 Artifacts

Artifacts are non-autonomous, function-oriented, stateful entities, designed by MAS programmers, which are controllable and observable, and that are used to model the tools and resources used by agents [30].

The *System variables artifact* keeps the description of the optimization problem, composed of its fitness function, and the number of variables. This artifact also holds the MOABHH parameters: (i) how many generations to spend during the initialization (δ), (ii) how many generations to process until the Copeland voting happens (γ), and (iii) the decreasing percent of the population size that should be given to the best evolutionary algorithms agent after

Copeland voting (β). This artifact is readable by all MOEA agents and the HH agent, but the Problem Manager Agent is the only one that can write in it.

There are two solutions related Artifacts. The first one, named *Population artifact*, keeps the main current population of solutions. This artifact is used by *Indicator voters*. When MOABHH starts, the Problem Manager agent randomly generates the first population, and then assigns it to this artifact.

The second artifact, named *Population artifact*, contains which solutions will be used by each evolutionary algorithm during the next generation. This is necessary because no MOEA agent can execute the whole population unless it proves clearly that it has the best evolutionary algorithm. The Population artifact is updated by the HH Agent using information from the Copeland Artifact.

The *Copeland artifact* keeps all the necessary rankings to generate the final rank. This artifact is used by the HH Agent, who defines how many solutions generations each meta-heuristic agent can perform.

4.2 Agents

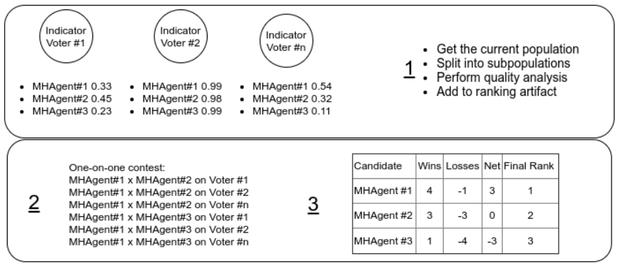
The *Problem Manager agent* is responsible for receiving MOABHparameters, instantiating the problem and setting all global parameters in the *System Variables* artifact.

The *MOEA* agent contain a particular multi-objective evolutionary algorithm instance. This agent receives a share of the current population, creates a new generation of solutions, selects which solution will compose the new population (with the same size of the received share), and replaces the received population share in the *Population Artifact*. All generated solutions are associated with their respective generator agent. Thus, it is possible to evaluate the agents' performance considering the solutions that were generated by them.

The *Indicator Voter* agent contains an instance of a single multi-objective quality indicator metric. This agent is also capable of generating the Copeland final rank, used by the *Hyper-heuristic* agent. However, just the last *Indicator Voter* is allowed to perform this task. Figure 2 shows the whole voting method. First (Step 1), each *Indicator Voter* reads the current population; splits it into subpopulations, where *MOEA* agents are associated to a subpopulation filled with their generated solutions; and evaluate each subpopulation following the instanced quality indicator. Thus, all MOEA agents have an associated quality value that is used to generate a ranking from the best to worse value. This ranking is then added to the Copeland artifact. So, as we can see in Figure 2 (part 1), the Copeland Artifact will have n ranks.

Finally, the *Indicator Voter* who generated its ranking at last reads all the indicators ranks existing in Copeland Artifact, performs the pairwise comparisons (*one-on-one contest*) (Step 2), generates the win-loss table, and sorts the table in ascending order according to final rank values (Step 3). This final rank values are accessible to all other agents, since it is available in the *Copeland Artifact*.

The *Hyper-heuristic* agent defines how many solutions, i.e., which population share, each *MOEA* agent will receive. For this purpose, this agent uses information provided by the *Copeland Artifact* to determine which *MOEA* deserves a bigger share of the population. First, in the initialization, all *MOEA* agents receive the same proportion of the population, and no Copeland voting happens. This strategy gives more time to *MOEA* agents to find solutions before measuring the qualities of the provided solutions. In every generation, the population is split into shares. Each share is filled with randomly selected solutions from the main population, obtained from the *population Artifact*. When the number of training generations (δ) is reached, at every γ generations, the Copeland voting procedure happens. In this way, the *HH Agent* uses updated information from the *Copeland Artifact* to define which *MOEA* will receive a bigger share, and which *MOEA* will receive a lower share. The best-ranked agent takes β% solutions from the worse ranked agent in the final Copeland Rank. This allows MOABHH to explore more the most suitable evolutionary algorithm rather than the other evolutionary algorithms.


Indicator Voter #1
Indicator Voter #2
Indicator Voter #n
MHAgent#1 0.33
MHAgent#2 0.45
MHAgent#3 0.23
MHAgent#1 0.99
MHAgent#2 0.98
MHAgent#3 0.99
MHAgent#1 0.54
MHAgent#2 0.32
MHAgent#3 0.11
1
Get the current population
Split into subpopulations
Perform quality analysis
Add to ranking artifact
2
One-on-one contest:
MHAgent#1 x MHAgent#2 on Voter #1
MHAgent#1 x MHAgent#2 on Voter #2
MHAgent#1 x MHAgent#2 on Voter #n
MHAgent#1 x MHAgent#3 on Voter #1
MHAgent#1 x MHAgent#3 on Voter #2
MHAgent#1 x MHAgent#3 on Voter #n
3

Candidate	Wins	Losses	Net	Final Rank
MHAgent #1	4	-1	3	1
MHAgent #2	3	-3	0	2
MHAgent #3	1	-4	-3	3

Figure 2: Voting method. In step 1 all Indicator voter agents rank MOEA agents based on their results. In step 2 the Copeland voting is performed. In step 3 the Copeland ranking is generated.

4.3 MOABHH Execution Flow

Algorithm 1 shows the interactions between Agents and Artifacts during MOABHH execution. First, all component (agents and artifacts) are initialized (Line 3), system parameters are defined (Line 4) and the first population is randomly generated and added to Population Artifact (Line 6). In the sequence, MOABHH performs its initialization sharing ($N/numOfMHAgents$) solutions for each MOEA agent (Line 10); this happens until a δ number of generations is not reach (Line 8). In thsi way, all MOEA agents can perform one generation and allocate the resulting solutions on the Population Share Artifact (Line 11 and Line 12). Finally, the HH Agent can use existing solutions in Population Share Artifact to set in Population Artifact. During this initialization process, there is no voting procedure.

After the initialization, MOABHH execution flow is performed while there remains evaluations to perform (Line 16). After each γ generations, the Copeland voting is performed (Lines 18 and 19). If there are more than two MHAgents (Line 20), the last voted losses β% of its population share, the most voted receives more $\beta * 0.75$% and the second most voted receives $\beta * 0.25$%. In the case of having just two MHAgents,the winner receives the whole β% of the last voted population share.

After voting, each MHAgent generates more solutions using the current solutions from the Population Share Artifact and replaces them by the new generated solutions (Line 27 and Line 28). Finnaly the HH agent uses existing solutions in Population Share Artifact to set in Population Artifact (Line 29).

Algorithm 1: MOABHH Pseudocode.

```
Input: $Problem$, $\gamma$ - generations before voting, $\delta$ -
       generations during initialization, $\beta$ - decreasing
       percent, $N$ population size, $maxGen$ - max
       number of generations
begin
    Start all agents and artifacts;
    Set MOABHH parameters to Problem Manager;
    Problem manager sets parameters to System
      Variables Artifact;
    Problem Manager randomly generates a population
      of solutions, then adds the generated population to
      Population Artifact;
    $gen = 0$;
    while $gen < \delta$ do
        //perform the initialization;
        HH Agents sets $(N/numOfMHAgents)$
          solutions for each MH Agent in Population
          Share Artifact;
        Each MHAgent generates new solutions using its
          population share;
        Each MHAgent replaces current solutions in
          Population Share Artifact by their new
          generated solutions;
        HH Agent takes solutions from Population Share
          Artifact and sets in Population Artifact;
        $gen++$;
    end
    while $gen < maxGen$ do
        if $gen\%\gamma = 0$ then
            Indicator Voter Agents votes considering
              solutions provided by Population Artifact;
            The last voter generates the final Copeland
              Ranking and assign it to Copeland Artifact;
            if There is more than two MHAgents active
              then
                HH Agent assigns more $\beta * 0.75$ percent
                  of the population share for the election
                  winner, more $\beta * 0.25$ for second place
                  winner and removes $\beta$ percent from the
                  last voted;
            end
            else
                HH Agent assigns more $\beta$ percent of the
                  population share for the election winner
                  and removes $\beta$ percent of the
                  population share from the less voted;
            end
        end
        Each MHAgent generates new solutions using its
          population share;
        Each MHAgent replaces current solutions in
          Population Share Artifact by their new
          generated solutions;
        HH Agent takes solutions from Population Share
          Artifact and sets in Population Artifact;
        $gen++$;
    end
end
```

5. EXPERIMENTAL SETTINGS

In our experiments, we used three independent evolutionary algorithms as MOEAs: NSGA-II, SPEA2, and IBEA, described in section 2.1. These three algorithms have been extensively applied in several multi-objective optimization problems, so they are suitable for our approach.

We then compared the performance of each of these individual algorithms with two MOABHH instances:

- $MOABHH_{rnd}$, which does not apply any voting method. It just randomly selects an order of evolutionary algorithms and set it on the Copeland artifact. Thus, this instance shows how MOABHH behaves without *Indicator voters*;

- $MOABHH_{cpl}$, uses *Indicator voters* and performs the Copeland method. We set five *Indicator voters* agents, each of them using one of the indicators detailed in section 2.2: RNI, Hypervolume, GD, IGD or Spread. Before calculating the indicator quality, we perform a normalization for each objective values, within the [0,1] range using the maximum and minimum values for each objective. We set both $MOABHH_{rnd}$ and $MOABHH_{cpl}$ parameters as $\delta = 112$, $\gamma = 12$ and $\beta = 3$.

All the evolutionary algorithms were set according to [13], using as heuristics SBX Crossover (with distribution 30 and rate 1.0) and Polynomial Mutation (with distribution 20 and rate de $1/n$, where n =number of problems variables). The population size was defined as 100.

As a benchmark, we have employed the well-known Walking Fish Group (WFG) [20] (WFG1 to WFG9) multi-objective benchmark. The WFG is a flexible and scalable suite which contains different kinds of optimization problems, showed in Table 1, with different Pareto optimal geometry, bias, separability (if a Pareto front is disconnected or not) and modality.

The experiments were performed using two and three objectives ($m = \{2, 3\}$) for all problems, totalizing 18 different experiments. The WFG suite problems have two parameters: number of distance-related and number of position-related. We set them according to to [7], where the number of distance-related variables was set $l = 20$, and the number of the position-related variables was set to 4.

Table 1: WFG characteristics, extracted from [20].

Problem	Separability	Modality	Bias	Geometry
WFG1	separable	uni	polynominal, flat	convex, mixed
WFG2	non-separable	uni	-	convex, disconnected
WFG3	non-separable	uni	-	linear, degenerate
WFG4	separable	multi	-	concave
WFG5	separable	deceptive	-	concave
WFG6	non-separable	uni	-	concave
WFG7	separable	uni	parameter dependent	concave
WFG8	non-separable	uni	parameter dependent	concave
WFG9	non-separable	multi, deceptive	parameter dependent	concave

All experiment were executed 40 times, during 750 generations, according to [7]. In the end, each algorithm execution generated a population of solutions. So, the population quality was calculated using all metrics showed in Section 2.2.

For Hypervolume, IGD, GD and Spread indicators we used the Pareto Fronts available in jMetal Framework as the reference. For RNI calculation, we joined all experiment population, for each problem, to create an approximated Pareto Front to be used as the reference Front.

Posteriorly, all indicator averages values for each problem were taken and statistically compared using Kruskal-Wallis test with 5% of significance.

Regarding our implementation, we developed MOABHH in Java JDK7, using the Cartago framework [29] to code the artifacts, and jMetal framework [15] for evolutionary algorithms and WFG problem instances.

6. RESULTS

In this section, we present the indicator averages for all tested algorithms, applied to all WFG's problems. The best and statistically tied averages are highlighted in the following figures.

Figure 3 shows hypervolume average values. This figure shows that $MOABHH_{cpl}$ at least tied with the best results in all problems. In some of them, $MOABHH_{cpl}$ got even better averages than any other algorithm: this is the case for problems WFG1, WFG3, WFG6 for two objectives and WFG2, WFG3 for three objectives. On the other hand, $MOABHH_{rnd}$ just got competitive results in two problems: WFG1 and WFG2 for two objectives, and has got a worse result in all other problems.

Figure 4 shows IGD average values. It shows $MOABHH_{cpl}$ as the best algorithm, considering the statistical difference, to solve WFG3 with two objectives. This algorithm also got better results, statistically tied, in WFG1, WFG2, and WFG6 with two objectives and WFG3 with three objectives. However, $MOABHH_{cpl}$ got its worst results in WFG4 and WFG5 for two objectives and WFG7 for three objectives, where $MOABHH_{rdn}$ overcome all other algorithms. In most problems, IBEA evolutionary algorithm got better results than NSGA-II and SPEA2. However, it was overcome by these algorithms in two objectives WFG4 and WFG5, problems where $MOABHH_{cpl}$ was also overcome.

Figure 5 shows RNI averages values. This figure shows $MOABHH_{cpl}$ competitive in most problems. There were some problems where $MOABHH_{cpl}$ got worse averages than the individual evolutionary algorithm IBEA. This happened in WFG1, WFG7 and WFG8 for three objectives. However, this does not mean a worse quality, but fewer non-dominated solutions. In most of the problems (4/18). $MOABHH_{rdn}$ did not find adequately a set of non-dominated solutions.

Figure 6 shows GD averages values. $MOABHH_{cpl}$ has a similar performance compared to IBEA, tied with the best algorithm in 10/18 problems. On the other hand, $MOABHH_{rdn}$ performed well in just 6/18 problems. For this indicator, SPEA2 got the best results.

Finally, Figure 7 shows Spread averages values. It shows a worse performance of $MOABHH_{cpl}$ when compared to $MOABHH_{rdn}$. This happens because this indicator focuses on how much spread solutions are present in the search space. This doesn't guarantee that these solutions can be considered as good ones, as Hypervolume and IGD indicators tell us. Obviously, $MOABHH_{rdn}$ finds more spread solutions because of its random evolutionary algorithm selection, which allows a larger exploration of the search space. We can also see in this figure that SPEA2 was the worst algorithm according to this Indicator.

Our results may be better explained considering the main characteristics of each of the quality indicators. According

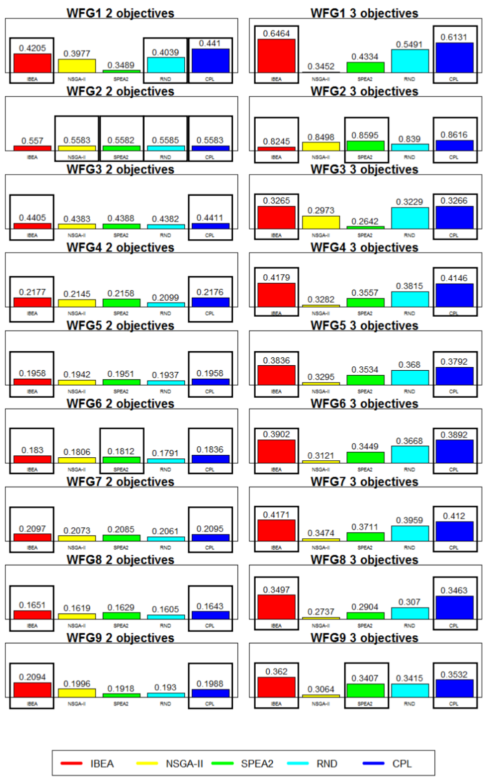


Figure 3: Hypervolume results for two and three objectives.

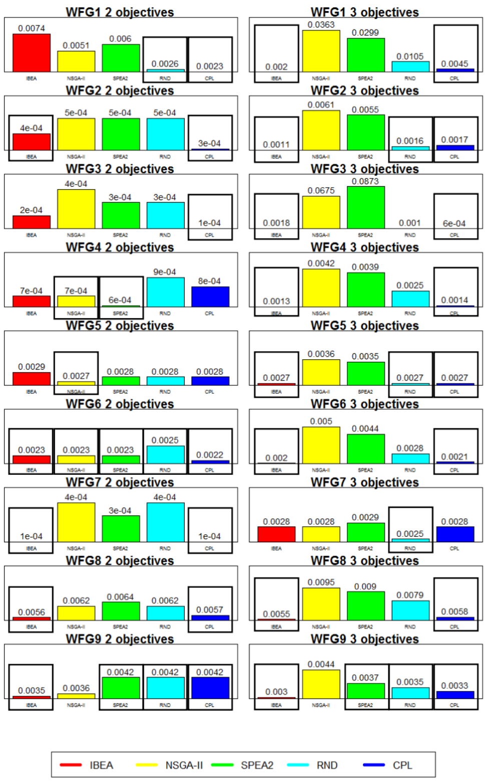


Figure 4: IGD results for two and three objectives.

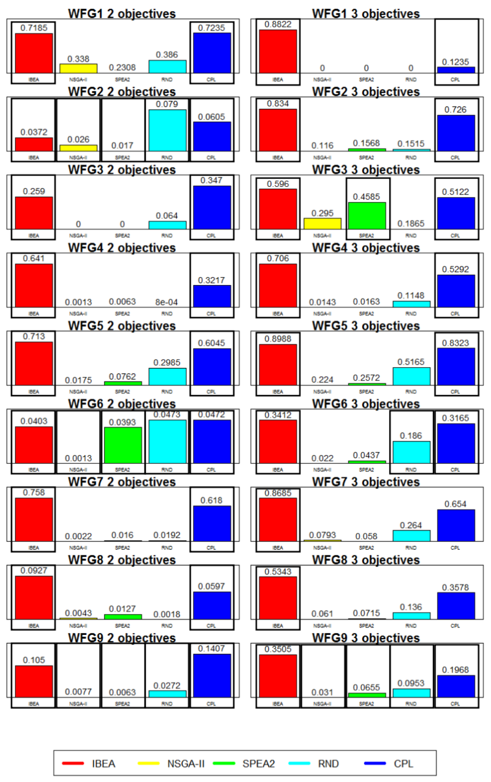


Figure 5: RNI results for two and three objectives.

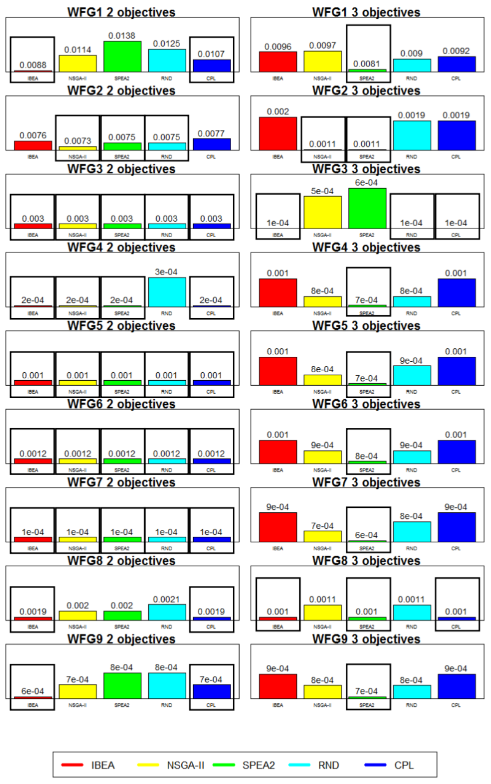


Figure 6: GD results for two and three objectives.

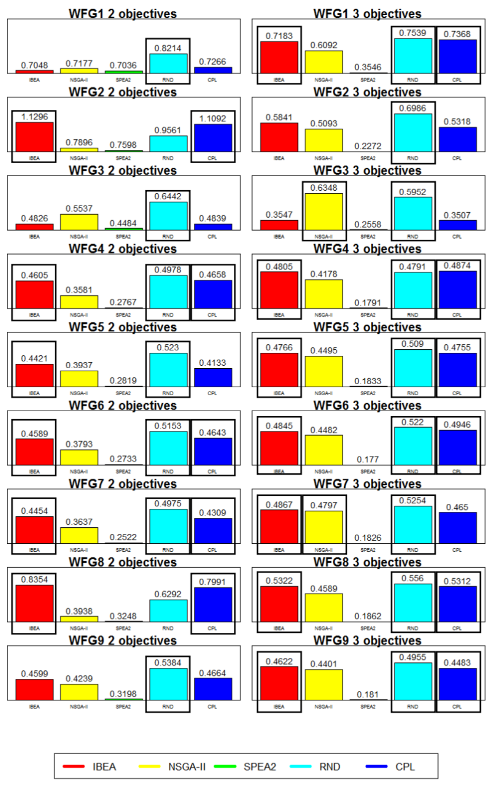


Figure 7: Spread results for two and three objectives.

to [21], the GD indicator measures the convergence performance, while the Spread indicator measures the diversity considering distribution and spread. Hypervolume and IGD measure both convergence and diversity. Our results show that $MOABHH_{cpl}$ has better averages on hypervolume and IGD indicators, competitive results on RNI and Spread indicators, but got its worse results considering the GD indicator. This happens due to the fact that this indicator is the only one which considers just convergence. In our experiments, we can see that the GD indicator voter tends to choose SPEA2 instead of IBEA, which is the preferred MOEA by most other indicator voters. IBEA is clearly superior to SPEA2 and NSGA-II considering Hypervolume and IGD, but according to the GD indicator it does not find good quality solutions in some problems.

Considering $MOABHH_{rdn}$, we can see that it is better according to the Spread indicator, but worse than $MOABHH_{cpl}$ with respect to Hypervolume, IGD, GD and RNI indicators. This happens because $MOABHH_{rdn}$ does not prioritize any indicator, and as a consequence it shares the population more uniformly. This allows $MOABHH_{rdn}$ to find more different solutions, although not necessarily non-dominated solutions.

7. RELATED WORK

In the literature, hyper-heuristics that were proposed to solve multi-objective optimization problems didn't apply agent-based techniques. Maashi et al. [24] proposed a evolutionary algorithm selection method, based on the Choice Function [12], which aimed to select, one at a time, during a given number of generations, an evolutionary algorithm from a set of three evolutionary algorithms, composed by NSGA-II, SPEA2, and MOGA [16]. For this purpose, they used a choice function based on a two ranking system. This ranking system employed the metric indicators Hypervolume, RNI, Spread and Computational time.

Vázquez-Rodríguez and Petrovic [34] combined a genetic algorithm with a mixture experiment to create a hyper-heuristic. Mixture experiment is a design of experiments technique that allows to efficiently exploit accumulated knowledge and to express it as a probability [11]. This hyper-heuristic selects solutions to compose the main population applying four different selection criteria. Each criterion was used considering an associated probability based on its performance, calculated during the search. This work employed NSGA-II, SPEA2, and IBEA evolutionary algorithms.

On the other hand, some researchers used agent-based approaches and reduced the difficulty of selecting the most suitable evolutionary algorithm: however, most of these works focused on solving single-objective optimization problems. Aydin and Fogarty [4] proposed use mono-objective evolutionary algorithms, such as Genetic Algorithm, Simulated Annealing [1] and Tabu Search [17], to solve the Job Shop Scheduling problem. In their approach, each agent can take a solution from the population and improve it. Milano and Roli [25] aimed to solve the MAXSAT problem using as agents GRASP, ACO, ILS, and MA mono-objective evolutionary algorithms. Talbi and Bachelet [32] used Tabu Search, Genetic Algorithms and a kick operator as agents that tried to solve the QaP problem; in their approach, GA explored the search space, the kick operator intensified these solutions, and Tabu Search was used to improve the solutions' quality.

Recently, Acan and Lotfi [2] proposed a multi-objective agent-based approach to solve the CEC2009 multi-objective optimization benchmark. Their work keeps a set of MOEA agents who act on a share of the main population, and selected the population following a cyclic or round-robin order. They keep a main archive of non-dominated solutions, and also each of the MOEA agents keeps an own archive as well. However, this makes this approach harder to compare against pure evolutionary algorithms (not agent-based), due to the use of many populations. Evolutionary algorithms employ just one or two populations, with a fixed maximal size ($maxSize$ parameter), to keep found solutions. When an approach employs multiple populations, one for each of N agents, it creates a much bigger pool to save solutions, with size $N * maxSize$. This increases the number of saved solutions, helping this approach to explore more the search space than a simple evolutionary algorithm with a single population.

8. CONCLUSIONS

In this work, we proposed the MOABHH (Multi-Objective Agent-Based Hyper-Heuristic) framework, a multi-agent hyper-heuristic focused on finding automatically the best evolutionary algorithm among a set of evolutionary algorithms. This selection is performed by a Copeland voting procedure among agents who employ different quality indicators values as a vote. Thus, the population of solutions is shared among MOEA agents by the hyper-heuristic agent. MOEA agents then generate new solutions, apply their acceptance criteria and return their updated population share. Finally, *Indicator voters* can evaluate MOEAs performance and perform the Copeland voting.

We conducted a set of experiments using the WFG benchmark suite with two and three objectives, following the literature recommendations. The obtained results showed that MOABHH can find the most suitable algorithm for these problems, and in the majority of cases gets results compatible with each individual evolutionary algorithm. Moreover, in some problems, MOABHH outperformed these individual algorithms.

Regarding future research, we intend to use different meta-heuristics such as MOEA/D-DRA [36], MOEA/D-DD [23] and MOMBI-II [19] to solve many-objectives problems, scaling up to ten objectives. We also intend to apply MOABHH to different benchmarks such as mQAP [22].

Acknowledgments

Vinicius Renan de Carvalho was supported by CNPq, Brazil, under grant agreement no. 140974/2016-4. Jaime Simão Sichman was partially supported by CNPq, Brazil, under grant agreement no. 303950/2013-7.

REFERENCES

[1] E. Aarts and J. Korst. *Simulated Annealing and Boltzmann Machines: A Stochastic Approach to Combinatorial Optimization and Neural Computing*. John Wiley & Sons, Inc., New York, NY, USA, 1989.

[2] A. Acan and N. Lotfi. A multiagent, dynamic rank-driven multi-deme architecture for real-valued multiobjective optimization. *Artificial Intelligence Review*, pages 1–29, 2016.

[3] S. F. Adra. *Improving Convergence, Diversity and Pertinency in Multiobjective Optimisation*. PhD thesis, Department of Automatic Control and Systems Engineering, The University of Sheffield, 2007.

[4] M. E. Aydin and T. C. Fogarty. Teams of autonomous agents for job-shop scheduling problems: An experimental study. *Journal of Intelligent Manufacturing*, 15(4):455–462, 2004.

[5] L. Bianchi, M. Dorigo, L. M. Gambardella, and W. J. Gutjahr. A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing*, 8(2):239–287, 2009.

[6] I. Boussaid, J. Lepagnot, and P. Siarry. A survey on optimization metaheuristics. *Information Sciences*, 237:82 – 117, 2013.

[7] L. Bradstreet, L. Barone, L. While, S. Huband, and P. Hingston. Use of the WFG toolkit and PISA for comparison of MOEAs. In *Proceedings of the 2007 IEEE Symposium on Computational Intelligence in Multicriteria Decision Making, MCDM 2007*, pages 382–389, 2007.

[8] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, and R. Ozcan, E.and Qu. Hyper-heuristics: a survey of the state of the art. *Journal of the Operational Research Society*, 64(12):1695–1724, Dec 2013.

[9] C. Coello. *Evolutionary algorithms for solving multi-objective problems*. Springer, New York, 2007.

[10] A. H. Copeland. A reasonable social welfare function. In *Mimeographed notes from a Seminar on Applications of Mathematics to the Social Sciences, University of Michigan*, 1951.

[11] J. A. Cornell. *Experiments with mixtures: designs, models, and the analysis of mixture data*, volume 895. John Wiley & Sons, 2011.

[12] P. I. Cowling, G. Kendall, and E. Soubeiga. A hyperheuristic approach to scheduling a sales summit. In

Selected Papers from the Third International Conference on Practice and Theory of Automated Timetabling, pages 176–190, 2001.

[13] K. Deb and H. Jain. An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints. *IEEE Transactions on Evolutionary Computation*, 18(4):577–601, Aug 2014.

[14] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, Apr 2002.

[15] J. J. Durillo and A. J. Nebro. jmetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, 42:760–771, 2011.

[16] C. M. Fonseca and P. J. Fleming. Multiobjective optimization and multiple constraint handling with evolutionary algorithms. i. a unified formulation. *Trans. Sys. Man Cyber. Part A*, 28(1):26–37, Jan. 1998.

[17] F. Glover, M. Laguna, E. Taillard, and D. De Werra. *Tabu search*. Baltzer Basel, 1993.

[18] D. E. Goldberg and J. H. Holland. Genetic algorithms and machine learning. *Machine learning*, 3(2):95–99, 1988.

[19] R. Hernández Gómez and C. A. Coello Coello. Improved metaheuristic based on the r2 indicator for many-objective optimization. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO '15, pages 679–686, New York, NY, USA, 2015. ACM.

[20] S. Huband, P. Hingston, L. Barone, and L. While. A review of multiobjective test problems and a scalable test problem toolkit. *Evolutionary Computation, IEEE Transactions on*, pages 477–506, 2006.

[21] S. Jiang, Y. S. Ong, J. Zhang, and L. Feng. Consistencies and contradictions of performance metrics in multiobjective optimization. *IEEE Transactions on Cybernetics*, 44(12):2391–2404, Dec 2014.

[22] J. Knowles and D. Corne. Instance generators and test suites for the multiobjective quadratic assignment problem. In C. Fonseca, P. Fleming, E. Zitzler, K. Deb, and L. Thiele, editors, *Evolutionary Multi-Criterion Optimization, Second International Conference, EMO 2003, Faro, Portugal, April 2003, Proceedings*, number 2632 in LNCS, pages 295–310. Springer, 2003.

[23] K. Li, K. Deb, Q. Zhang, and S. Kwong. An Evolutionary Many-Objective Optimization Algorithm Based on Dominance and Decomposition. *IEEE Transactions on Evolutionary Computation*, 19(5):694–716, oct 2015.

[24] M. Maashi, E. Özcan, and G. Kendall. A multi-objective hyper-heuristic based on choice function. *Expert Systems with Applications*, 41(9):4475–4493, 2014.

[25] M. Milano and A. Roli. Magma: A multiagent architecture for metaheuristics. *Trans. Sys. Man Cyber. Part B*, 34(2):925–941, Apr. 2004.

[26] H. Moulin. Condorcet's principle implies the no show paradox. *Journal of Economic Theory*, 45(1):53–64, 1988.

[27] V. Pareto. *Manuel D Economie Politique*. Ams Press, Paris, 1927.

[28] J. Pomerol and S. Barba-Romero. *Multicriterion decision in management: principles and practice*, volume 25. Springer Science & Business Media, 2012.

[29] A. Ricci, M. Piunti, M. Viroli, and A. Omicini. Environment programming in cartago. In A. El Fallah Seghrouchni, J. Dix, M. Dastani, and R. H. Bordini, editors, *Multi-Agent Programming:*, pages 259–288. Springer US, 2009.

[30] A. Ricci, M. Viroli, and A. Omicini. Give agents their artifacts: the a&a approach for engineering working environments in MAS. In E. H. Durfee, M. Yokoo, M. N. Huhns, and O. Shehory, editors, *6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2007), Honolulu, Hawaii, USA, May 14-18, 2007*, page 150. IFAAMAS, 2007.

[31] N. Srinivas and K. Deb. Multiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation, IEEE Transactions on*, 2:221–248, 1994.

[32] E. Talbi and V. Bachelet. COSEARCH: A Parallel Cooperative Metaheuristic. *Journal of Mathematical Modelling and Algorithms*, 5(1):5–22, 2006.

[33] K. C. Tan, T. Lee, and E. Khor. Evolutionary algorithms for multi-objective optimization: Performance assessments and comparisons. *Artificial Intelligence Review*, 17(4):251–290, 2002.

[34] J. Vázquez-Rodríguez and S. Petrovic. A mixture experiments multi-objective hyper-heuristic. *Journal of the Operational Research Society*, 64(11):1664–1675, 2012.

[35] L. While, L. Bradstreet, and L. Barone. A fast way of calculating exact hypervolumes. *IEEE Transactions on Evolutionary Computation*, 16(1):86–95, Feb 2012.

[36] Q. Zhang, W. Liu, and H. Li. The performance of a new version of MOEA/D on CEC09 unconstrained MOP test instances. Technical Report CES-491, School of CS & EE, University of Essex, Feb 2009.

[37] E. Zitzler and S. Künzli. Indicator-based selection in multiobjective search. In *PPSN*, volume 3242 of *Lecture Notes in Computer Science*, pages 832–842. Springer, 2004.

[38] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. In *Evolutionary Methods for Design Optimization and Control with Applications to Industrial Problems*, pages 95–100. International Center for Numerical Methods in Engineering, 2001.

[39] E. Zitzler and L. Thiele. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *Trans. Evol. Comp*, 3(4):257–271, nov 1999.

[40] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *Trans. Evol. Comp*, 7(2):117–132, Apr. 2003.