Meta-Evolutionary Algorithms and Recombination Operators for Satisfiability Solving in Fuzzy Logics

Tim Brys*, Madalina M. Drugan* and Ann Nowé*
*Artificial Intelligence Lab, VUB
Pleinlaan 2, 1050 Brussels, Belgium
{timbrys, mdrugan, anowe} @vub.ac.be

***Abstract*—In this work, we develop a new paradigm, called Meta-Evolutionary Algorithms, motivated by the challenging, continuous problems encountered in the domain of satisfiability in fuzzy logics (SAT_∞). In Meta-Evolutionary Algorithms, the individuals in a population are optimization algorithms themselves. Mutation at the meta-population level is handled by performing an optimization step in each optimization algorithm, and recombination at the meta-population level is handled by exchanging information between different algorithms. We analyse different recombination operators and empirically show that simple Meta-Evolutionary Algorithms are able to outperform CMA-ES on a set of SAT_∞ benchmark problems.**

I. Introduction

In previous work [1]–[3], we have investigated optimization approaches to the problem of solving satisfiability in fuzzy logics. Although these problems are defined on a continuous domain, $[0, 1]^n$, they exhibit combinatorial properties like their propositional counterpart, and specifically the problems from Łukasiewicz logic are challenging due to the large number of plateaus contained therein. While the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [4] has been consistently among the best performing algorithms, we have shown that it is possible to improve significantly over standard CMA-ES when applying some of the principles outlined in this paper [3], i.e. employing multiple parallel populations with recombination between them.

The concept of running multiple populations or optimization algorithms in parallel is not new [5]–[10]. Having multiple algorithms (or multiple instances of the same algorithm) potentially allows the different algorithms to explore different parts of the search space, and increases the probability of one instance starting in the basin of attraction of a good optimum. This concept is not always useful though, as not all problem landscapes have many suboptimal attractors, are deceptive, or possess some other property that can result in premature convergence.

The two methods that come closest to what is being proposed in this paper are Meta-ES [7] and the Island Model in Genetic Algorithms [5], which are both special cases of the method we propose. In Meta-ES, or nested ES, an outer ES is built that manages a set of inner ES'. Each of these inner ES' is run γ generations independently, after which selection takes place in the outer ES, new inner ES' are produced and this procedure is repeated. This method allows the outer ES to tune the inner ES' parameters. In the Island Model GA framework, different populations are run in parallel. Individuals are allowed to migrate between populations at certain intervals, increasing individual populations' genetic diversity and reinvigorating their search.

In this work, we generalize these methods to Meta-Evolutionary Algorithms (MEA), by looking at multiple parallel populations and multiple parallel optimization algorithms (homogeneous or heterogeneous) as individuals of the MEA's population. Mutation, recombination and selection are applied at the meta-algorithm level. The exchange of individuals between populations, specific to Island Models, is then only one of many possible meta-recombination operators.

Another way to view MEA is as follows: MEA are a generalization of EA that apply intermediate steps of local search to individuals [11]. We generalize the individuals and local search to possibly any optimization algorithm, although with a preference for those that perform more local search. The former is a bottom-up view of MEA, the latter a top-down view. A last research domain that is related to the work proposed here is niching in evolutionary algorithms [9], [12], where multiple sub-populations are run on multimodal functions to find multiple optima of an objective function. We propose to implement interactions between these sub-populations, not to find multiple optima, but to find the best optimum.

After formalizing the MEA paradigm in the following section, we analyse different possible recombination operators in Section III, and empirically compare various MEA algorithms to CMA-ES on a benchmark set of SAT_∞ problems, see Section IV.

II. Meta-Evolutionary Algorithms

An important problem experienced by many optimization algorithms, and especially by the ones that search locally, is a tendency for the algorithms to get stuck in local sub-optima. Various mechanisms to handle this problem have been proposed, including random restart strategies, probabilistic acceptance of worsening steps and multiple parallel algorithm instances/populations. In this work, we are most interested in

this last strategy, and especially in combination with information exchange between algorithm instances running in parallel. In the MEA framework, we generalize the concepts of parallel algorithms with information exchange into an Evolutionary Algorithm (EA) that operates at a hierarchically higher level than the instances that run in parallel. This MEA considers each instance as an individual in its meta-population, and applies the archetypal EA operators of selection, recombination and mutation at this meta-population level. Pseudo-code for the Meta-Evolutionary Algorithm[1], is given in Figure 1. In the following sections, we discuss the specifics of MEA.

```
1: procedure MEA
2:     Individuals ← InitializeIndividuals()
3:     Fitness ← EvaluateFitness(Individuals)
4:     while not stopCriterion() do
5:         Meta-Select(Individuals, Fitness)
6:         Meta-Recombine(Individuals)
7:         Meta-Mutate(Individuals)
8:         Fitness ← EvaluateFitness(Individuals)
9:     end while
10: end procedure
```

Fig. 1. Meta-Evolutionary Algorithm Template

A. Individuals

In regular EA, an individual is a candidate solution, i.e. an assignment to the problem's variables. In MEA, an individual represents the state of an optimization algorithm. For example, in the case of a basic hillclimber, the individual will consist of a single candidate solution. In the case of a basic GA with population size μ, the individual is a list of μ candidate solutions. In the case of CMA-ES, the individual consists of the mean and covariance matrix of CMA-ES' multivariate distribution, the evolution paths, stepsize, etc., i.e. all parameters representing the state of the algorithm.

The functioning of an EA depends on a fitness evaluation of individuals, allowing it to decide which individuals are better than others. In MEA, depending on the formulation of individuals, different approaches for fitness evaluation strategies are possible. For the hillclimber, the single candidate solution is evaluated to represent the fitness of the individual. In the GA case, the best or average fitness in the list of candidate solutions (population) can be used. For CMA-ES, the mean of the distribution's fitness is representative of the individual's fitness.

B. Mutation

In MEA, mutation is performed by executing a single step in an individual's optimization algorithm. In the hillclimber example, the neighbourhood of the current candidate solution is evaluated, and the candidate solution is replaced with the most improving neighbour, if any. In the GA example, the list of candidate solutions will have been updated to the new generation through the GA's selection, recombination and mutation operators. In the CMA-ES example, the mean, covariance matrix and evolution paths will have been updated according to CMA-ES' mechanisms. This implementation of mutation will alter the individual in a non-random way; MEA individuals are self-improving.

A core necessity for the successful application of MEA is that the underlying optimization algorithms should be of such a nature that they are able to diversify, i.e. that they are likely to follow different search paths and/or converge to different solutions. Without this property, running multiple instances would only add redundant computational cost. In the simplest case, the individuals are basic hillclimbers that start from random locations and converge to different local attractors.

C. Recombination

Recombination is meant to exploit commonalities in the search space, i.e. to exploit the information contained in different individuals to construct better ones. In MEA, these individuals represent the state of specific instances of optimization algorithms, and recombination allows us to exchange information between these instances. This information exchange can take many forms, and a large part of this paper is dedicated to the analysis of various meta-recombination operators, see Section III.

Recombining two hillclimber individuals is straightforward, as classical EA recombination operators can be applied to their respective candidate solutions. Recombining two GA individuals can be handled by exchanging individuals between their populations, which is the Island Model GA case, or by calculating a representative candidate solution (e.g. by averaging), and using that candidate solution for recombination (e.g. crossover) and reconstructing the population from the child. Recombining two meta-individuals governed by CMA-ES can be done by exchanging elements from the mean of their multivariate distributions, and possibly on top of that exchanging corresponding elements from the covariance matrix and evolution path.

D. Selection and diversity

Selection is the final EA operator to consider. It acts on the diversity in the population generated through mutation and recombination, selecting those individuals from the population that are allowed to survive, reducing the population's diversity, biasing the search towards individuals with higher fitness.

Note that the dynamics of the MEA are not the same as those of a regular EA. At the meta-level, mutation of individuals is non-random, as they follow a search path dictated by the optimization algorithm governing them. Selection is already present at the individual's level, and therefore we believe that little explicit selection at the meta-population level is necessary. In this paper, selection is only considered in combination with the recombination operators.

[1]This concept of meta-evolutionary algorithms must not be confused with meta-evolutionary programming [13], where a meta-algorithm optimizes the parameters of an EA during search.

Random restart strategies are the most commonly used mechanism to overcome premature convergence to local sub-optima, and can be present in the optimization algorithms working at the lower level. Self-restarting individuals will increase the diversity at the meta-population level, addressing the problem of premature convergence at that level too.

III. Recombination Operators

In this section, we propose recombination operators for MEA, and analyse their impact on search space exploration and optimization performance. Recombination operators should be tailored to the structure of individuals. For example, recombination of hillclimbers is straightforward as classical recombination operators can be directly applied to the candidate solutions that represent their state. On the other hand, when not only candidate solution(s), but also other algorithmic parameters are included in the representation, the recombination operator should intelligently recombine them, e.g. the covariance matrices of two CMA-ES instances. The operators considered in this section are built for recombining single candidate solutions. In Section IV-D, we discuss how other algorithm parameters can be recombined, with CMA-ES as an example.

```
1: procedure META-RECOMBINE(Individuals)
2:     R ← ∅
3:     while size(Individuals) ≥ ρ do
4:         R ← R ∪ OptimalMixing(Sample(Individuals, ρ))
5:     end while
6:     Individuals ← R
7: end procedure
```

Fig. 2. Recombination - For ρ the number of parents necessary for recombination, randomly select ρ individuals from the meta-population without replacement, and apply optimal mixing until less than ρ individuals are left.

A. Optimal mixing

Before considering the actual recombination operators, we discuss the optimal mixing procedure [14], which is a type of selection after recombination that guarantees improvement through recombination. This will ensure that the underlying optimization algorithms' search is only changed when it is beneficial. Figure 2 shows the implementation of meta-recombination using optimal-mixing. Our implementation of optimal mixing for two parents is shown in Figure 3 and goes as follows:

- For a pair of individuals, the set of problem variables is divided into s disjunct sets of variables, each containing at least one variable. Variables are randomly assigned to sets.
- For a single variable set, a recombination operator is applied to the two parents, recombining the variables in that set. In the original paper proposing optimal mixing, simple value exchange was used.
- If at least one of the modified offspring has greater fitness than their parents, the recombination is accepted. This ensures global improvement through mixing, avoiding local oscillations when optimized substructures go back and forth between algorithms.
- This recombination is performed for each disjunct variable set, each time continuing with the accepted offspring from the previous step. The children at the end of the procedure replace the original parents.

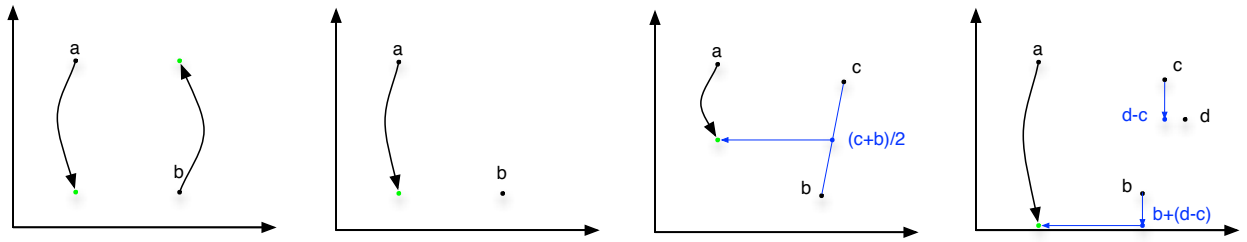
```
1: procedure OPTIMALMIXING(parent1, parent2)
2:     variableSets ← DisjunctVariableSets(s)
3:     fp1 ← fitness(parent1)
4:     fp2 ← fitness(parent2)
5:     child1 ← parent1
6:     child2 ← parent2
7:     for each set ∈ variableSets do
8:
9:         //Recombination
10:        for each var ∈ set do
11:            ROperator(var, child1, child2)
12:        end for
13:
14:        //Selection
15:        fc1 ← fitness(child1); fc2 ← fitness(child2)
16:        if acceptRecombination(fp1, fp2, fc1, fc2) then
17:            parent1 ← child1; parent2 ← child2
18:            fp1 ← fc1; fp2 ← fc2
19:        else
20:            //Undo recombination
21:            for each var ∈ set do
22:                child1[var] ← parent1[var]
23:                child2[var] ← parent2[var]
24:            end for
25:        end if
26:    end for
27: end procedure
```

Fig. 3. Optimal Mixing of Two Individuals

The procedure outlined here is optimal mixing with the marginal product structure (disjunct sets of variables). Ideally, these sets match the inherent structure of the problem being solved, if any. In the original work, this structure was learned using information analysis. Due to the limited number of meta-individuals practically possible, using information analysis yields no improvement in this setting. Therefore, the division of variables into disjunct sets is random.

B. Recombination operators

It is possible to vary on this template by implementing different recombination operators (ROperator in the pseudocode), and having different numbers of parents for recombination. In the following paragraphs, we outline four different recombination operators for continuous spaces that will be analysed within the MEA framework. For a visual representation of these operators, see Figure 4. In the equations that illustrate the operators, we show the modifications executed by the recombination operator on a single variable var. All other



(a) Exchange (b) Stealing (c) Average (d) Differential

Fig. 4. Example recombinations with different operators in a two-dimensional problem. Only the second, or y-variable is considered for recombination, x stays fixed. Black arrows and green dots indicate the actual recombination, blue arrows and dots indicate intermediate steps.

variables in $child_i$ are set to the values of the corresponding $parent_i$:

$$\forall var, i : child_i[var] = parent_i[var]$$

Exchange. The simplest of recombination operators; the classical crossover. A two-parent operator, which exchanges variable values between parents. Recombinations are accepted if one of the children is better than both parents.

$$child_1[var] = parent_2[var]$$
$$child_2[var] = parent_1[var]$$

Stealing. A two-parent recombination operator, where one parent 'steals' variable values from the other. A recombination is accepted if child$_1$ has improved over parent$_1$.

$$child_1[var] = parent_2[var]$$

Average. A three-parent recombination operator, where one parent's variable values are set to the average of those of the other two parents. A recombination is accepted if child$_1$ has improved over parent$_1$.

$$child_1[var] = \frac{parent_2[var] + parent_3[var]}{2}$$

Differential. The Differential Evolution operator [15], [16]. A four-parent recombination operator, where one parent's variable values are set to the difference between the values of the second and third parent, added to those of the fourth. a in the equation controls the stepsize. A recombination is accepted if child$_1$ has improved over parent$_1$.

$$child_1[var] = parent_4[var] + a(parent_2[var] - parent_3[var])$$

Applying these operators to the state of optimization algorithm instances will allow us to control their search. While stealing and averaging are expected to intensify the search in certain promising regions of the search space by bringing instances closer together, exchange is expected to have a neutral effect on meta-population diversity. The envisioned effect of the differential operator on diversity is unclear. The effect of these operators on search behaviour will be analysed in the following section, which starts with a description of SAT$_\infty$ problems, which will serve as a benchmark.

IV. Experimental results

A. SAT$_\infty$

In fuzzy logics [17], truth is expressed as a real number taken from the unit interval $[0, 1]$. Essentially, there are an infinite number of truth degrees possible (in boolean logic, there are only two truth degrees, i.e. 0 and 1). As an example of a particularly popular fuzzy logic, in Łukasiewicz logic, negation $\neg$, conjunction $\otimes$, disjunction $\oplus$ and implication $\rightarrow$ are interpreted as follows (with $[\alpha]_\mathcal{I}$ indicating the truth degree of that formula under variable assignment (interpretation) $\mathcal{I}$.):

- $[\neg\alpha]_\mathcal{I} = 1 - [\alpha]_\mathcal{I}$
- $[\alpha \otimes \beta]_\mathcal{I} = \max([\alpha]_\mathcal{I} + [\beta]_\mathcal{I} - 1, 0)$
- $[\alpha \oplus \beta]_\mathcal{I} = \min(1, [\alpha]_\mathcal{I} + [\beta]_\mathcal{I})$
- $[\alpha \rightarrow \beta]_\mathcal{I} = \min(1 - [\alpha]_\mathcal{I} + [\beta]_\mathcal{I}, 1)$

for formulas α and β.

A formula in a fuzzy logic is said to be satisfied when its truth degree, given a certain interpretation, lies between a given lower and upper bound. The upper bound is usually 1 (in boolean SAT, both lower and upper bounds are 1). An example formula follows:

$$0.5 \leq \neg(v_1 \otimes v_2 \otimes \neg v_3) \leq 1 \tag{1}$$

To solve a SAT$_\infty$ instance containing n formulas using an optimization approach, we define an objective function $f(\mathcal{I})$ whose global optima correspond to solutions of the SAT$_\infty$ instance [1]. $\mathcal{I}$ is an assignment to the variables, and α_i, l_i and u_i are the ith formula and its lower and upper bounds respectively:

$$f(\mathcal{I}) = \frac{\sum_{i=1}^{n} g(\mathcal{I}, \alpha_i, l_i, u_i)}{n} \tag{2}$$

and,

$$g(\mathcal{I}, \alpha_i, l_i, u_i) = \begin{cases} 1 & \text{if } l_i \leq [\alpha_i]_\mathcal{I} \leq u_i \\ \frac{[\alpha_i]_\mathcal{I}}{l_i} & \text{if } [\alpha_i]_\mathcal{I} < l_i \\ \frac{1-[\alpha_i]_\mathcal{I}}{1-u_i} & \text{if } [\alpha_i]_\mathcal{I} > u_i \end{cases} \tag{3}$$

g is a trapezoid function, with a plateau of value 1 when formula α_i's degree of satisfaction lies between the given bounds, and a slope leading to the plateau when the satisfaction lies outside these bounds. When the global maximum of f has

a function value of 1, the SAT_∞ instance is satisfiable, and every global maximum is then a solution to the problem.

We use the problem benchmark set built in [1], which consists of 50 problems instances in the Łukasiewicz logic, with problem bounds from $\mathbb{T}_{100} = \left\{0, \frac{1}{100}, \frac{2}{100}, ..., 1\right\}$. These problems were shown to be hard for the state-of-the-art analytical solver due to the high granularity in the bounds[2].

B. (μ, λ)-ES as MEA component

The optimization algorithm we will use in our MEA is a simple (μ, λ)-Evolution Strategy without self-adaptation. Specifically, it consists of a multi-variate distribution from which λ individuals are sampled. The μ best are retained and used to calculate a new mean for the distribution by averaging the μ individuals. The next generation of candidate solutions is sampled from this distribution, and this procedure is repeated. The algorithm employs an exponentially decreasing schedule for the stepsize, while the variance is the same in all directions. This algorithm was shown to perform reasonably well on the problems considered in this paper [2]. MEA individuals of this ES are represented by the distribution's mean.

Figure 5 shows pseudo-code for the specific ES that will be used in this paper.

Boundary constraints are handled by repairing solutions that fall outside the $[0, 1]^n$ box.

C. Analysis of recombination operators

In this section, we empirically analyse the effect of various recombination operators on the search behaviour of the ES outlined in the previous section. We build a MEA with a population of 10 randomly initialized instances of this ES, and for each of the 50 problem instances, 50 runs are performed, with runs limited to 1000 generations (one generation in MEA means executing one generation in each of its individuals). Parameters for the ES are $\mu = 1$, $\lambda = 10$, and stepsize is decayed from 2.5 to 0.025 by multiplying with 0.99 each generation. To get a good comparison of the different operators, we apply all of them the same number of times. By this we mean that, while exchange could be applied to 5 pairs of parents in a population of 10, and differential only to two quadruples, we apply both operators only twice. Of course, this comparison is still not perfect, as exchange and stealing then only access 40% of the information contained in the population (2 recombinations, each on 2 parents out of 10 potential parents), while average has access to 60% and differential to 80% (2 recombinations on 4 parents each). Stepsize a for the differential operator is set to 0.5. For the number of sets the variables are divided into for mixing, we propose $s = \frac{n}{\log n}$ as a general guideline, with n the number of dimensions in the problem.

We measure the ratio of accepted versus tried mixings, which indicates an operator's ability to propose solutions improving on the parents, see Figure 6(a). In the first generation, the differential evolution operator can only propose improving solutions in about 15% of the cases, while the other operators more than double that figure. All operators converge to almost no improving recombinations after 100 generations.

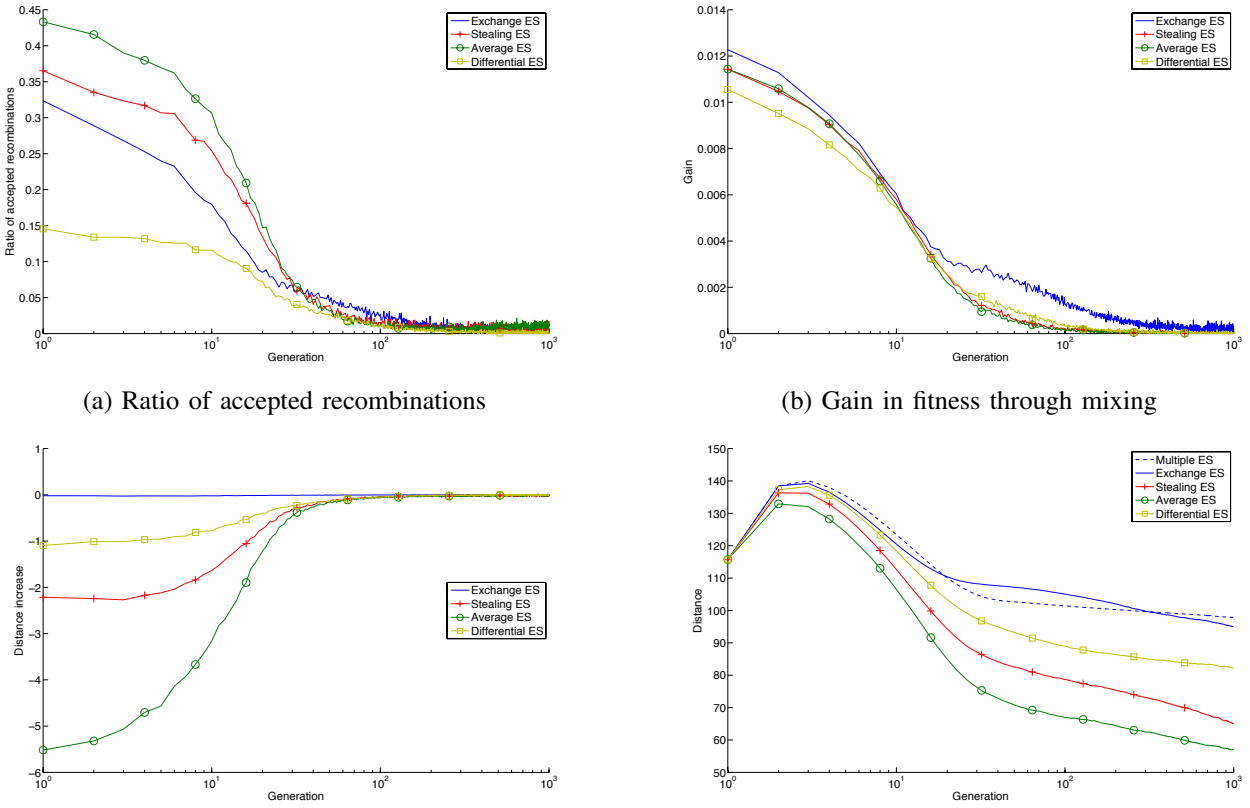
If we look at the absolute increase in fitness of the accepted recombinations, Figure 6(b), all methods behave similarly, except for the exchange operator which is able to maintain a relatively high level of fitness increases much longer than the other three methods.

Figure 6(c) shows the direct effect mixing has on the distance between ES instances, which is a measure of the diversity in the MEA population. The exchange operator is neutral in this respect; on average, the distance between meta-individuals does not change. Differential will slightly decrease the diversity by bringing individuals closer together, and stealing and averaging bring them much closer together. Figure 6(d) shows the evolution of the actual inter-ES distance over a number of generations. A MEA with independent ES', i.e. without recombination, is included in this graph, to isolate

```
1:  procedure ES
2:      // Initialization
3:      mean ← RandomPoint()
4:      stepsize ← InitialStepsize()
5:      for i ∈ Individuals do
6:          Individuals_i ← Sample(N(mean, stepsize × I) )
7:      end for
8:      Fitness ← EvaluateFitness(Individuals)
9:
10:     // Main loop
11:     while not stopCriterion() do
12:         // Selection
13:         Individuals ← Sort(Individuals, Fitness)[1..μ]
14:
15:         // New Distribution
16:         mean ← 0⃗
17:         for i ∈ Individuals do
18:             mean ← mean + i
19:         end for
20:         mean ← mean / nrIndividuals
21:         stepsize ← Decay(stepsize)
22:
23:         // Restarting
24:         if restartCriterion() then
25:             mean ← RandomPoint()
26:             stepsize ← InitialStepsize()
27:         end if
28:
29:         // Sampling next generation
30:         for i ∈ Individuals do
31:             Individuals_i ← Sample(N(mean, stepsize × I))
32:         end for
33:         Fitness ← EvaluateFitness(Individuals)
34:     end while
35: end procedure
```

Fig. 5. Evolutionary Strategy. **I** is the identity matrix. Mutation in a MEA individual equals executing one iteration of the main loop.

[2]These problems can be downloaded here: http://ai.vub.ac.be/members/tim-brys, along with Java code to interpret them.



(a) Ratio of accepted recombinations

(b) Gain in fitness through mixing

(c) Increase in meta-population diversity through mixing

(d) Average meta-population diversity

Fig. 6. (a) Percentage of accepted recombinations. (b) Increase in fitness through recombination (c) Increase in distance between populations through mixing (negative values indicate approaching populations) (d) Average distance between populations.

the effect of mutation (optimization at the lower level) on diversity. The distance between independent ES' decreases as some of them converge to similar regions in the search space. If we consider the addition of recombination operators, we can see that the direct effect of stealing, average and differential on diversity (Figure 6(c)) carries over to the actual diversity evolution over time. They all decrease diversity with the same relative effect. Only the results for the exchange operator are noteworthy. Applying the exchange operator, while in itself it has a neutral effect on diversity, actually maintains diversity longer than when there is no interaction among the ES. We believe this follows from the fact that exchange constructs two new individuals, one containing the best variable assignment combinations from both parents, while the alternative values survive in the other child, which is often worse off. This second ES is then forced to optimize these variable values himself, resulting in extended exploration of the search space.

Average and stealing clearly intensify the search in promising regions of the search space. We hypothesize that this behaviour will lead to faster solving performance. On the other hand, exchange preserves higher diversity among the ES', which should allow them to solve more successfully hard, deceptive problems by exploring different regions in the search space. We hypothesize that using exchange will result in a higher probability of solving an instance in the long run, although at the cost of slower performance in general. Differential holds the middle ground when considering ES diversity, but has a much lower percentage of successful recombinations, and therefore seems to offer little advantage. In the next sections, we compare the performance of the algorithmic variants described before, with CMA-ES and a MEA using CMA-ES.

D. CMA-ES as MEA component

A CMA-ES meta-individual is represented by the mean of its distribution, its covariance matrix, and the evolution paths. The recombination of this representation is less trivial, as it incorporates more information than a single candidate solution would.

Consider exchange recombination. Values from the mean and evolution paths can simply be exchanged, as these are vectors in which each element describes the current state of CMA-ES for each variable independently. The recombination of the covariance matrix is less simple, as it encodes information between pairs of variables, which furthermore may or may not be kept together during recombination. The way the matrix exchange recombination is implemented is as follows: if variables i and j stay together, the covariance information for element (i, j) comes from the corresponding parent. If the

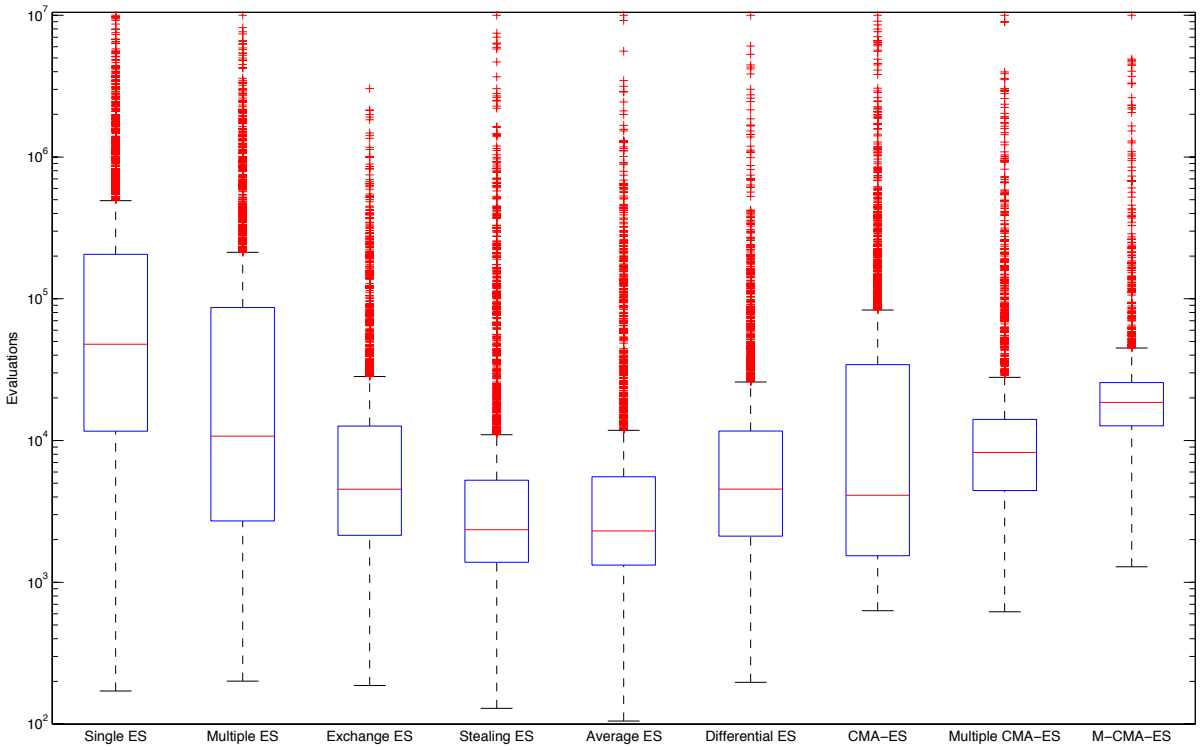


Fig. 7. Boxplot of the number of evaluations each algorithm requires to reach a solution on the Łukasiewicz $\mathbb{T}_{100}$ problems.

	Single ES	Ind. ES	Exch. ES	Steal. ES	Avg. ES	Diff. ES	CMA-ES	Ind. CMA-ES	M-CMA-ES
Median	44.33×10^3	9.71×10^3	4.53×10^3	2.33×10^3	$\mathbf{2.29 \times 10^3}$	4.53×10^3	3.93×10^3	8.16×10^3	18.56×10^3
Mean	32.44×10^4	17.57×10^4	$\mathbf{3.36 \times 10^4}$	5.54×10^4	4.08×10^4	4.31×10^4	11.45×10^4	6.10×10^4	5.20×10^4
Success	96.64%	96.84%	**100%**	98.84%	98.32%	99.8%†	98.32%	98.28%	99.96%†

TABLE I

MEDIAN AND MEAN NUMBER OF EVALUATIONS FOR SUCCESSFUL RUNS, AND PERCENTAGE OF SUCCESSFUL RUNS ON THE ŁUKASIEWICZ $\mathbb{T}_{100}$ PROBLEMS. THOSE RESULTS MARKED WITH † ARE NOT SIGNIFICANTLY DIFFERENT FROM THE BEST (WILCOXON SIGNED-RANK TEST, $\alpha = 0.05$)

variables come from different solutions, the average of the covariance information from both solutions is calculated and stored. This covariance matrix recombination can only take place after every variable set is exchanged, since only then can we know of all variables whether they stay together or not. This Meta-EA with CMA-ES and exchange recombination was proposed in [3] and coined Mixing-CMA-ES (M-CMA-ES).

In previous work, we used a penalty function to handle the box-constraint in CMA-ES, as this is the method recommended by Hansen [18]. Meanwhile, we have determined that boundary repair performs much better for CMA-ES on these problems, even though it violates various assumptions about the distribution of individuals (results not included). Therefore, we use boundary repair throughout the experiments in this paper.

E. Performance results

The actual performance of the ES, five variations of MEA containing ES (independent, exchange, stealing, average and differential), CMA-ES, and two variations of MEA containing CMA-ES (independent and exchange) is shown in Figure 7 and summarized in Table I. 50 runs are performed on each of the 50 Łukasiewicz problem instances, with a maximum number of 10^7 evaluations allowed. We show mean and median number of evaluations required to solve an instance in successful runs, as well as the percentage of successful runs. Note that we do not compare multiple populations with a single large population, as we have previously determined that performance decreases with larger populations for these algorithms solving these problems (given a budget of 10^7 evaluations) [2]. In this experiment, the number of recombinations is not equalized among the different operators, and thus exchange and stealing will perform more recombinations than average, which in turn will perform more than differential. Other parameters are the same as before, and μ and λ for CMA-ES are the same as for the simple ES.

While running multiple independent instances of the simple ES already greatly improves performance over a single ES, applying recombination operators further improves this. Averaging and stealing have better median performance, but

also have a heavier tail, as indicated by their mean performance and success percentage which are worse than those for exchange and differential. When not considering median evaluations to a solution, exchange is the best performing operator. These results confirm our expectations concerning stealing and average, which converge faster to solutions, but trade this off with a lower final success rate. Exchange on the other hand is slower, but more robust in solving instances. Surprisingly, the differential operator yields performance very similar to the exchange operator. With exchange, the ES perform much better than CMA-ES with exchange (M-CMA-ES), establishing this Meta-EA as the new state-of-the-art algorithm for solving satisfiability problems in Łukasiewicz logic.

A last factor not yet considered is that of implementation complexity. The covariance matrix adaptation mechanism in CMA-ES is very powerful, but also very difficult to implement. On the other hand, the mechanisms proposed in this paper are relatively straightforward and do not require complex mathematical operations to achieve similar performance to CMA-ES (although we must not forget that our basic approach here is not completely black-box, given the optimized stepsize decay schedule, while CMA-ES is).

V. Conclusions

In this paper, we proposed Meta-Evolutionary Algorithms (MEA), which consist of a set of (possibly interacting) instances of optimization algorithms. Each instance applies its own search procedure, while information exchange between instances is possible and can alter the instance's search process. This method is inspired by, and generalizes, Meta-ES, the Island Model in GA, niching methods in EA, and EA with intermediate local search steps. It could be called a parallel portfolio approach with information exchange between algorithms.

Four meta-level recombination operators were proposed and their effect on search behaviour analysed. While stealing and averaging were shown to intensify search in promising regions of the search space, this is at the cost of long term solving success. Exchange and differential operators are able to maintain higher diversity in the meta-population, resulting in slower convergence to solutions, but better exploration of the search space, and are therefore more robust to deceptive features in the landscape. We showed that MEA consisting of simple ES, using any of these recombination operators, were able to perform at least as good as standard CMA-ES on a set of satisfiability instances from Łukasiewicz logic. A great advantage is the relative simplicity of implementing MEAs, which exploit the parallel search of different optimization algorithms' instances to explore the search space more thoroughly than a single would (depending on how local the algorithm's search is).

In this paper, we have only considered homogeneous MEAs, with all individuals executing the same randomized search procedure. In a heterogeneous approach, we could have one CMA-ES instance, coupled with hundreds of very simple hillclimbers, combining CMA-ES' robustness with a fast exploration of the search space by hillclimbers. In the future, we will evaluate this MEA framework on more conventional benchmark problems, with better known properties, including propositional SAT and continuous benchmark functions from the BBOB competition. Preliminary results suggest that applying MEA with CMA-ES and exchange (M-CMA-ES) to benchmark functions such as Bohachevsky, Rastrigin and Schwefel greatly improves performance, as it exploits the functions' separability.

Acknowledgment

Tim Brys is funded by a Ph.D grant of the Research Foundation Flanders (FWO). Madalina M. Drugan performed this work funded by a Visiting Fellowship of the FWO.

References

[1] T. Brys, Y.-M. De Hauwere, M. De Cock, and A. Nowé, "Solving satisfiability in fuzzy logics with evolution strategies," in *Proceedings of the 31st Annual North American Fuzzy Information Processing Society Meeting*, 2012.

[2] T. Brys, M. M. Drugan, P. A. N. Bosman, M. D. Cock, and A. Nowé, "Local search and restart strategies for satisfiability solving in fuzzy logics," in *Proceedings of the 6th International Workshop on Genetic and Evolutionary Fuzzy Systems (GEFS), 2013 IEEE Symposium Series on Computational Intelligence*, 2013.

[3] T. Brys, M. M. Drugan, P. A. N. Bosman, M. De Cock, and A. Nowé, "Solving satisfiability in fuzzy logics by mixing cma-es," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2013.

[4] N. Hansen, "The CMA evolution strategy: a comparing review," in *Towards a new evolutionary computation. Advances on estimation of distribution algorithms*, J. Lozano, P. Larranaga, I. Inza, and E. Bengoetxea, Eds. Springer, 2006, pp. 75–102.

[5] P. B. Grosso, "Computer simulations of genetic adaptation: Parallel subcomponent interaction in a multilocus model." *Dissertation Abstracts International Part B: Science and Engineering[DISS. ABST. INT. PT. B-SCI. & ENG.],*, vol. 46, no. 7, 1986.

[6] J. Branke, T. Kaußler, C. Schmidt, and H. Schmeck, "A multi-population approach to dynamic optimization problems," *Adaptive computing in design and manufacturing*, vol. 2000, p. 299, 2000.

[7] H.-G. Beyer and H.-P. Schwefel, "Evolution strategies – a comprehensive introduction," *Natural Computing*, vol. 1, pp. 3–52, 2002.

[8] B. Niu, Y. Zhu, and X. He, "Multi-population cooperative particle swarm optimization," *Advances in Artificial Life*, pp. 874–883, 2005.

[9] O. M. Shir and T. Bäck, "Niching with derandomized evolution strategies in artificial and real-world landscapes," *Natural Computing*, vol. 8, no. 1, pp. 171–196, 2009.

[10] P. A. N. Bosman, "On empirical memory design, faster selection of bayesian factorizations and parameter-free gaussian edas," in *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*. ACM, 2009, pp. 389–396.

[11] M. W. S. Land, "Evolutionary algorithms with local search for combinatorial optimization," Ph.D. dissertation, Citeseer, 1998.

[12] S. W. Mahfoud, "Niching methods for genetic algorithms," *Urbana*, vol. 51, p. 61801, 1995.

[13] D. B. Fogel, L. J. Fogel, and J. W. Atmar, "Meta-evolutionary programming," in *Conference record of the twenty-fifth asilomar conference on Signals, systems and computers, 1991*. IEEE, 1991, pp. 540–545.

[14] D. Thierens and P. A. N. Bosman, "Optimal mixing evolutionary algorithms," in *GECCO*, N. Krasnogor and P. L. Lanzi, Eds. ACM, 2011, pp. 617–624.

[15] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[16] M. M. Drugan and D. Thierens, "Geometrical recombination operators for real-coded evolutionary mcmcs," *Evolutionary computation*, vol. 18, no. 2, pp. 157–198, 2010.

[17] P. Hájek, *Metamathematics of Fuzzy Logic*. Springer, 1998.

[18] N. Hansen, "The CMA Evolution Strategy: A tutorial," 2011.