Local Search and Restart Strategies for Satisfiability Solving in Fuzzy Logics

Tim Brys*, Madalina M. Drugan*, Peter A.N. Bosman†, Martine De Cock§ and Ann Nowé*
*Artificial Intelligence Lab, VUB
Pleinlaan 2, 1050 Brussels, Belgium
{timbrys, mdrugan, anowe} @vub.ac.be
†Centrum Wiskunde & Informatica (CWI)
P.O. Box 94079, 1090 GB Amsterdam, The Netherlands
peter.bosman@cwi.nl
§Dept. of Applied Math. and Comp. Sc. UGent
Krijgslaan 281 (S9), 9000 Gent, Belgium
martine.decock@ugent.be

***Abstract*—Satisfiability solving in fuzzy logics is a subject that has not been researched much, certainly compared to satisfiability in propositional logics. Yet, fuzzy logics are a powerful tool for modelling complex problems. Recently, we proposed an optimization approach to solving satisfiability in fuzzy logics and compared the standard Covariance Matrix Adaptation Evolution Strategy algorithm (CMA-ES) with an analytical solver on a set of benchmark problems. Especially on more finegrained problems did CMA-ES compare favourably to the analytical approach. In this paper, we evaluate two types of hillclimber in addition to CMA-ES, as well as restart strategies for these algorithms. Our results show that a population-based hillclimber outperforms CMA-ES on the harder problem class.**

I. Introduction

The problem of SAT in propositional logic [1] is well known and is of interest to researchers from various domains [2], [3], as many problems can be reformulated as a SAT problem and subsequently solved by a state-of-the-art SAT solver.

In fuzzy logics – logics with an infinite number of truth degrees – the same principle of satisfiability exists, SAT_∞, and it is, like its classical counterpart, useful for solving a variety of problems. Indeed, many fuzzy reasoning tasks can be reduced to SAT_∞, including reasoning about vague concepts in the context of the semantic web [4], fuzzy spatial reasoning [5] and fuzzy answer set programming [6], which in itself is an important framework for non-monotonic reasoning over continuous domains (see e.g. [7], [8], [9]).

In contrast to SAT, SAT_∞ has received much less attention from the various research communities; there are only two types of analytical solvers to be found in the literature, one using mixed integer programming [10], and one using a constraint satisfaction approach [11]. Both methods suffer from exponential complexity issues associated respectively with mixed integer programming itself and an iteratively refining discretization.

Recently, we proposed a new approach to solving SAT_∞ [12] that models satisfiability as an optimization problem over a continuous domain, and has the advantage of being independent of the underlying logic and its operators, as well as not suffering from the complexity issues associated with analytical solvers, as our results showed. The disadvantage of this approach is that only given an optimization algorithm proven to converge to the global optimum can it be a complete solver, i.e. deciding both SAT_∞ and UNSAT_∞. The optimization algorithm used in [12] was the Covariance Matrix Adaptation Evolution Strategy (CMA-ES)[13], which is considered to be state-of-the-art in black-box optimization on a continuous domain. In this paper we explore some alternatives to this algorithm as the core for this SAT_∞ solver. Our findings are somewhat surprising in that CMA-ES is outperformed by a hillclimber on one problem class.

The rest of the paper is structured as follows: in Section II, we give a brief description of fuzzy logics and formally define SAT_∞ as an optimization problem. Section III contains the description of a hillclimber algorithm (III-A), a population-based version of that algorithm (III-B) and CMA-ES (III-C), as well as restart strategies for these algorithms. These algorithms are then evaluated and compared on a set of benchmark problems in Section IV.

II. Fuzzy Logic and SAT_∞

In fuzzy logics [14], truth is expressed as a real number taken from the unit interval $[0, 1]$. Essentially, there are an infinite number of truth degrees possible. A formula in a fuzzy logic is built from a set of variables $\mathcal{V}$, constants taken from $[0, 1]$ and n-ary connectives for $n \in \mathbb{N}$. An interpretation is a mapping $\mathcal{I} : \mathcal{V} \to [0, 1]$ that maps every variable to a truth degree. We can extend this fuzzy interpretation $\mathcal{I}$ to formulas as follows:

- For each constant c in $[0, 1]$, $[c]_\mathcal{I} = c$.
- For each variable v in $\mathcal{V}$, $[v]_\mathcal{I} = \mathcal{I}(v)$.
- Each n-ary connective f is interpreted by a function $\mathtt{f} : [0, 1]^n \to [0, 1]$. Furthermore we define

$$[f(\alpha_1, \ldots, \alpha_n)]_\mathcal{I} = \mathtt{f}([\alpha_1]_\mathcal{I}, \ldots, [\alpha_n]_\mathcal{I})$$

for formulas α_i with $1 \leq i \leq n$.

The connectives in fuzzy logics typically correspond to connectives from classical logic, such as conjunction, disjunction, implication and negation, which are interpreted respectively by a t-norm, a t-conorm, an implicator and a negator. A triangular norm or t-norm $\mathtt{T}$ is an increasing, associative and commutative $[0,1]^2 \to [0,1]$ mapping that satisfies the boundary condition $\mathtt{T}(1,x) = x$ for all x in $[0,1]$. Similarly, a triangular conorm or t-conorm $\mathtt{S}$ is an increasing, associative and commutative $[0,1]^2 \to [0,1]$ mapping that satisfies the boundary condition $\mathtt{S}(0,x) = x$. An implicator $\mathtt{I}$ is a $[0,1]^2 \to [0,1]$ mapping that is decreasing in its first argument, increasing in its second argument and that satisfies the properties $\mathtt{I}(0,0) = \mathtt{I}(0,1) = \mathtt{I}(1,1) = 1$ and $\mathtt{I}(1,0) = 0$. A negator $\mathtt{N}$ is a decreasing $[0,1] \to [0,1]$ mapping that satisfies $\mathtt{N}(0) = 1$ and $\mathtt{N}(1) = 0$.

As an example of a particularly popular fuzzy logic, in Łukasiewicz logic, negation $\neg$, conjunction $\otimes$, disjunction $\oplus$ and implication $\to$ are interpreted as follows:

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

for any formulas α and β.

An interpretation $\mathcal{I}$ is said to be a model of a set of formulas Θ iff $\forall\alpha \in \Theta : l \leq [\alpha]_{\mathcal{I}} \leq u$, given lower and upper bounds l and u for that formula (usually u is 1, and in classical logic even both l and u are 1). An example of a formula with three variables v_1, v_2, and v_3 in Łukasiewicz logic, with bounds is:

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

One can easily verify that $\mathcal{I}_1$ with $\mathcal{I}_1(v_1) = 0$, $\mathcal{I}_1(v_2) = 0$ and $\mathcal{I}_1(v_3) = 1$ is a model of this formula as $[\neg(v_1 \otimes v_2 \otimes \neg v_3)]_{\mathcal{I}_1} = 1$. Similarly, $\mathcal{I}_2$ with $\mathcal{I}_2(v_1) = 0.6$, $\mathcal{I}_2(v_2) = 0.7$ and $\mathcal{I}_2(v_3) = 0.2$ is a model too because $[\neg(v_1 \otimes v_2 \otimes \neg v_3)]_{\mathcal{I}_2} = 0.9$. Even though the formula is not perfectly satisfied under interpretation $\mathcal{I}_2$, the degree of satisfaction is still high enough to meet the lower bound $l = 0.5$. The existence of the models $\mathcal{I}_1$ and $\mathcal{I}_2$ show that formula (1) is satisfiable. Solving SAT_∞ amounts to finding a model for the set of formulas given, or deciding that there is no interpretation that satisfies all formulas and that the set is UNSAT_∞.

A. SAT_∞ as an Optimization Problem

As we propose an optimization approach to solving satisfiability in fuzzy logics, we need to reformulate SAT_∞ instances as optimization problems, i.e. defining a function over the solution space such that optimizing this function corresponds to solving the SAT_∞ instance. A SAT_∞ problem consists of a set Θ of formulas α_i, each of which must be satisfied to a certain degree, as defined by an upper and lower bound per formula. Given these n formulas α_i, bounds (u_i, l_i), and an interpretation $\mathcal{I}$, we define the objective function f as follows:

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

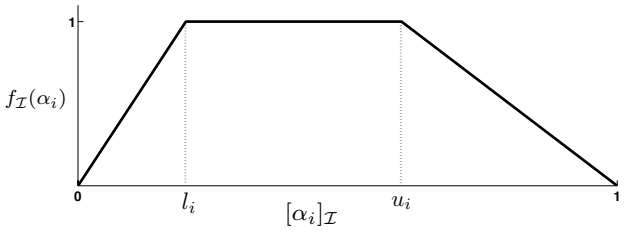

Fig. 1. $f_{\mathcal{I}}(\alpha_i)$ for a formula α_i with lower bound l_i, upper bound u_i and degree of satisfaction $[\alpha_i]_{\mathcal{I}}$.

and,

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

with $[\alpha_i]_{\mathcal{I}}$ representing the degree of satisfaction of formula α_i under interpretation $\mathcal{I}$. Each $f_{\mathcal{I}}$ 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, as visualised in Figure 1. This formulation is similar to that typically used for SAT in propositional logic, where the number of satisfied clauses is divided by the number of clauses. The difference here is that in case of non-satisfaction, we do not always return 0 as in SAT, but we give gradient information that can point an optimization algorithm to satisfying configurations.

The objective function is formulated such that the global maxima will always have a function value 1 if the SAT_∞ instance is satisfiable. In that case, every global maximum corresponds to a model of the problem. Given an algorithm of which we can prove convergence to the global maximum on this function, we have a complete SAT_∞ solver. As we can not provide such an algorithm, we are left with an incomplete solver, being able to solve SAT_∞ sometimes, but never concluding UNSAT_∞.

III. Algorithms

In this section, we discuss three optimization algorithms that we evaluate and compare on a set of benchmark problems further on in this paper.

A. Hillclimber

The first algorithm we consider is a hillclimber. Such local search algorithms often perform very well given their simplicity. The most basic version iteratively updates a candidate solution by evaluating all possible neighbouring solutions, given a neighbourhood function, and picks the most improving one. This method is called steepest ascent hill climbing (or descent, when considering a minimization problem), because it stops when it no longer finds better solutions in the neighbourhood, i.e. you are at the top of the hill.

Such algorithms are very efficient at moving quickly to a local optimum. When a local optimum has been found, the hillclimber restarts from a random location. Restart strategies for (local) search algorithms have often been shown to improve an algorithm's qualitative performance. The most cited reason for this is that performance almost always depends on starting conditions, and especially the starting location in the search space, although to what extent depends on the algorithm itself. The most simple hillclimber (without restart strategy) simply converges to the local optimum in which basin of attraction it has started. Other algorithms, such as CMA-ES – see below –, have mechanisms that allow them to overcome such problems, although only to a certain extent. A restart strategy allows an algorithm that may have started in a very bad region of the search space, with little to no chance of finding good solutions, move to another location in the search space that may prove to be much better.

In the particular hill climber we propose, we use a continuous neighbourhood, where all neighbours lie in a hypersphere around the current solution at a certain distance. Because of the continuous neighbourhood, we cannot consider all potential neighbours, and therefore cannot definitely tell whether there are no improving neighbours and whether the current solution is a local optimum. Therefore, we determine that we are in a local optimum when we have not observed an improvement for i iterations; at that point the hillclimber restarts.

For the same reason that we cannot evaluate all neighbours, we generate and evaluate only one single neighbour and decide how to proceed solely based on that neighbour's fitness. This neighbour is selected by taking a random direction and taking the point on the hypersphere that lies in that direction. If that new candidate solution is an improvement or a status quo, we accept the step, otherwise, depending on probability p, we accept the worsening step anyway, or we pick another neighbour in a different direction. This added stochasticity can help the hillclimber to escape suboptimal local optima.

A last mechanism is a changing neighbourhood over time (iterations) [15]. In the experimental section, we evaluate both constant neighbourhoods (a hypersphere with fixed radius) and exponential decay neighbourhoods (a hypersphere with exponentially decaying radius). This concept of a decaying neighbourhood is conceived to allow the hillclimber to initially make large steps to locate promising regions in the search space, and then gradually refine the solutions by searching more locally. With a decaying stepsize, the schedule is reset after a restart, so that the hillclimber can again start making large steps to quickly improve the candidate solution. Besides this, we add some gaussian noise to the stepsize.

Note that this particular hillclimber could arguably be referred to as simulated annealing with a fixed acceptance probability. In simulated annealing, the probability of accepting worsening steps is decayed over time, and the neighbourhood usually stays the same. In our algorithm the inverse is true, i.e. the probability of accepting worsening steps is fixed and the neighbourhood can be varied, by decaying the distance between neighbouring solutions.

Of course, solutions need to adhere to the box-constraint for fuzzy truth values $\forall v : v \in [0, 1]$. When a step takes the algorithm outside the box defined by the possible truth values ($[0, 1]^n$), we force it back in by setting the violating variables to the closest value in the box, i.e. 0 or 1.

Figure 2 shows the pseudo-code for this hillclimber. Initially, a random interpretation (variable assignment) is generated and assigned to the current solution $\mathcal{I}$. Then, as long as no model (satisfying assignment) is found or the maximum number of allowed function evaluations is not exceeded, we iteratively update the current solution. To select a neighbour on the hypersphere, we generate a random normalized vector to select a direction, and multiplied with the stepsize or radius of the hypersphere, added to the current solution, we obtain a neighbour. If it is an improvement, we continue with that solution. If not, with probability p we continue with that solution anyway, and otherwise we select another neighbour. If no improvement has been made over the past i function evaluations, we reassign the current solution to a random location in the search space, i.e. a restart.

```
1:  procedure HILLCLIMBER
2:      I = randomInterpretation()
3:      while not satisfied and not maxEvaluations do
4:          s = randomNormalizedVector()
5:          tmp = I + (s × stepSize)
6:          if f(tmp) ≤ f(I) || rand() < p then
7:              I = tmp
8:          end if
9:          if restartCondition() then
10:             I = randomInterpretation()
11:         else
12:             stepSize = updateStepsize(iteration)
13:         end if
14:     end while
15: end procedure
```

Fig. 2. Hillclimber

B. Population-based Hillclimber

Instead of working with a single solution, we can build a hillclimber that works on a population of candidate solutions. Populations typically allow an algorithm to better estimate the local shape of the search space and make more informed decisions. In this case, we implement some principles borrowed from Evolution Strategies (ES) [16], [17], which optimize an objective function by iteratively applying the principles of selection, mutation and recombination to a population of candidate solutions.

We have implemented these principles using a multivariate distribution with the same variance in all dimensions, i.e. no covariances. This is used to sample λ solutions for the next generation, from which the μ best are selected, which are subsequently used to calculate the mean of the distribution that will generate the next generation. This mean is simply the average of the μ best solutions. As with the no-population

hillclimber, we allow this population-based hillclimber to restart when it has not encountered better solutions for a certain number of function evaluations, and we will also look at both constant and decaying neighbourhoods (standard deviation). Boundary violations are handled in the same way as in the previous hillclimber. Figure 3 shows pseudocode for this algorithm.

```
1:  procedure POP-HILLCLIMBER
2:      mean = randomInterpretation()
3:      while not satisfied and not maxEvaluations do
4:          generation_λ = sampleDistribution(mean, std)
5:          generation_λ = sort(generation_λ)
6:          if restartCondition() then
7:              mean = randomInterpretation()
8:          else
9:              best_μ = generation_λ[1..μ]
10:             mean = avg(best_μ)
11:             std = updateStd(iteration)
12:         end if
13:     end while
14: end procedure
```

Fig. 3. Population-based Hillclimber

C. CMA-ES

The Covariance Matrix Adaptation-Evolution Strategy (CMA-ES) [13] is an algorithm belonging to the class of Evolution Strategies, and is considered to be state-of-the-art in black-box optimization on a continuous domain. CMA-ES is an (μ, λ)-ES with the addition of a non-random adaptation of the multivariate distribution that generates candidate solutions. This additional mechanism allows CMA-ES to efficiently explore promising search directions by adapting the distribution to the local shape of the search space.

For a full description of this algorithm, we refer the reader to [13]. Pseudocode is shown in Figure 4.

```
1:  procedure CMA-ES
2:      mean = randomInterpretation()
3:      cov = initialCovarianceMatrix()
4:      while not satisfied and not maxEvaluations do
5:          generation_λ = sampleDistribution(mean, cov)
6:          generation_λ = sort(generation_λ)
7:          if restartCondition() then
8:              mean = randomInterpretation()
9:              cov = initialCovarianceMatrix()
10:         else
11:             best_μ = generation_λ[1..μ]
12:             mean = avg(best_μ)
13:             cov = updateCovMatrix(mean, best_μ)
14:         end if
15:     end while
16: end procedure
```

Fig. 4. Covariance Matrix Adaptation Evolution Strategy

Although we could also force the candidate solutions that violate the box constraint back in the box, we apply a penalty function as suggested in [18], because CMA-ES makes assumptions about the distribution of these solutions. This penalty is quadratic in the distance between the candidate solution and the box. The new objective function is:

$$f(\mathcal{I}) = \begin{cases} \frac{-\sum_{i=1}^{n} f_{\mathcal{I}}(\alpha_i, l_i, u_i)}{n}, & \mathcal{I} \in [0,1]^v \\ 10^4 \sum_{i=1}^{v} \theta(|(\mathcal{I}(i) - 0.5)| - 0.5)(\mathcal{I}(i) - 0.5)^2, & \mathcal{I} \notin [0,1]^v \end{cases} \tag{4}$$

with θ the heaviside function (0 for negative input, otherwise 1) and v the number of variables. Note the minus in the first part, which is necessary because CMA-ES is a minimization technique.

We make the new function conditional instead of adding the penalty to the objective function as is commonly done, because numbers beyond [0,1] have no meaning in fuzzy logics and prevent the formulas from being evaluated. We do realise that now the objective function is different for the hillclimbers and CMA-ES, but note that this is necessary. As we want each algorithm to perform as good as possible, we chose not to use this penalty function for the hillclimbers, but to repair solutions, as this gives better performance. CMA-ES unfortunately does not perform well when repairing solutions and requires this penalty function.

Auger and Hansen have introduced a restart strategy for CMA-ES called LR-CMA-ES [19]. Restart conditions are, amongst others: little or no variation in the fitness of the last x generations, extremely small standard deviation of the normal distribution and no change in fitness function when perturbing any coordinate of the mean with 0.2 standard deviation. These are all indications that CMA-ES has converged and that it will not very likely find better solutions in subsequent generations. When this happens, LR-CMA-ES restarts in another random location in the search space. In our experiments, we will consider both these sophisticated restart conditions, and our simple 'x function evaluations without improvement' restart condition.

IV. EXPERIMENTS

In this section, we evaluate the previously described algorithms on a testbed of SAT_∞ problems. These benchmark problems come from [11] and [12] and can be divided into two problem classes, namely those with constants (formulas' upper and lower bounds) from $\mathbb{T}_4 = \left\{0, \frac{1}{4}, \frac{2}{4}, \frac{3}{4}, 1\right\}$, and those problems with constants from $\mathbb{T}_{100} = \left\{0, \frac{1}{100}, .., \frac{99}{100}, 1\right\}$. The latter problems were much harder for the analytical approach from [11] to solve, due to the higher granularity in constants. The testbed contains 50 instances of each problem class, each problem having 40 dimensions or variables, making for 100 problem instances in total.

The algorithms we will evaluate all have some parameters that need to be set, such as the stepsize (or standard deviation) in the hillclimbers, and the population size in population-based

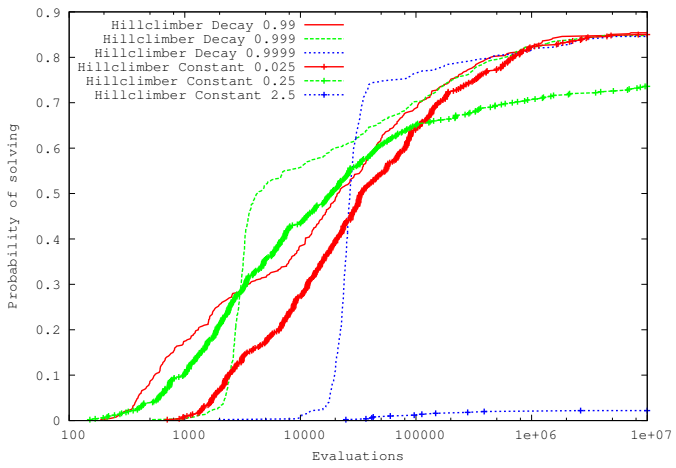

Fig. 5. Probability of the hillclimber solving an instance for various stepsize schedules on Łukasiewicz $\mathbb{T}_4$ problems.

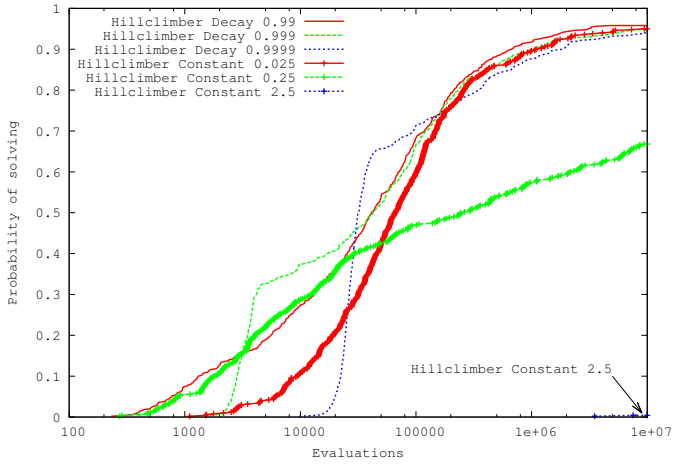

Fig. 6. Probability of the hillclimber solving an instance for various stepsize schedules on Łukasiewicz $\mathbb{T}_{100}$ problems.

hillclimber and CMA-ES. Before we compare these algorithms, we perform a performance analysis of each algorithm with various parameter settings, and select those parameter values that yield the best results. These settings are then used in the final comparison.

A. Hillclimber

Figures 5 and 6 show the performance of the hillclimber with different stepsize schedules, either constant or exponentially decaying. Table I summarises these results. When the stepsize is decaying exponentially, it starts at value 2.5 and does not go lower than 0.025. These values were selected for their good performance. The probability of accepting worsening solutions is set to 0.1. Restarts are performed after 1000 iterations without improvement. Evaluations are limited to 10^7. For each instance, 10 runs were performed, all algorithms using the same list of initial and restart positions (randomly generated for each run). This setup stays the same for all preliminary experiments in this paper.

Hillclimber	Configuration	Success
Łukasiewicz $\mathbb{T}_4$	**Decay 0.99**	**85.4%**
	Decay 0.999	84.8%
	Decay 0.9999	84.6%
	Constant 0.025	85.0%
	Constant 0.25	73.6%
	Constant 2.5	2.2%
Łukasiewicz $\mathbb{T}_{100}$	**Decay 0.99**	**95.8%**
	Decay 0.999	94.8%
	Decay 0.9999	94.0%
	Constant 0.025	95.0%
	Constant 0.25	66.8%
	Constant 2.5	0.4%

TABLE I
PERCENTAGE OF SUCCESSFUL RUNS FOR DIFFERENT HILLCLIMBER CONFIGURATIONS. ALL OBSERVED DIFFERENCES ARE STATISTICALLY SIGNIFICANT (WILCOXON SIGNED RANK TEST).

Both the hillclimbers with the decaying stepsizes and the hillclimber with the smallest constant stepsize solve an apartenly similar number of instances at the end of the allowed number of evaluations, although the hillclimber with the fastest decay schedule is statistically significantly better.

We can differentiate between the hillclimbers with decay schedules by looking at the average number of evaluations necessary to solve an instance. Slower decay schedules do not find solutions early on, but after a certain number of evaluations, they quickly climb to a much higher probability of solving an instance than those hillclimbers with faster decay schedules. This is a result of a higher amount of exploration with a large stepsize, allowing them to locate better regions in the search space before performing more local search with a small stepsize. From these experiments, we select a decay schedule with rate 0.99 to use in further experiments, as it has a statistically significantly higher probability of solving a problem instance in the benchmark set.

B. Population-based Hillclimber

Considering the population-based hillclimber, we can both optimize the population size λ, as well as the ratio of selected individuals versus the number of individuals in a population μ/λ, as well as the size of the distribution used to generate candidate solutions, i.e. the standard deviation or stepsize. As many combinations of settings are possible, we restrict ourselves to evaluating different values for λ $(10 - 1000)$, as well as two μ/λ ratios (0.5 and 0.1). We assume that the performance of various stepsize schedules will not differ much compared to that found for the previous hillclimber, and as such, we use the 0.99 decay schedule. Restarting is after $1000 \times \lambda$ evaluations without improvement.

Figures 7 and 8 show the relation between population size and hillclimber performance. Table II summarises these results. As the population size grows, the hillclimbers perform worse because of the limited function evaluation budget. Selecting less individuals from the population μ/λ is also clearly better. From this analysis, we select a restarting hillclimber, with $\lambda = 10$ and $\mu/\lambda = 0.1$ for both problem classes. We also checked our assumption for taking the best stepsize schedule

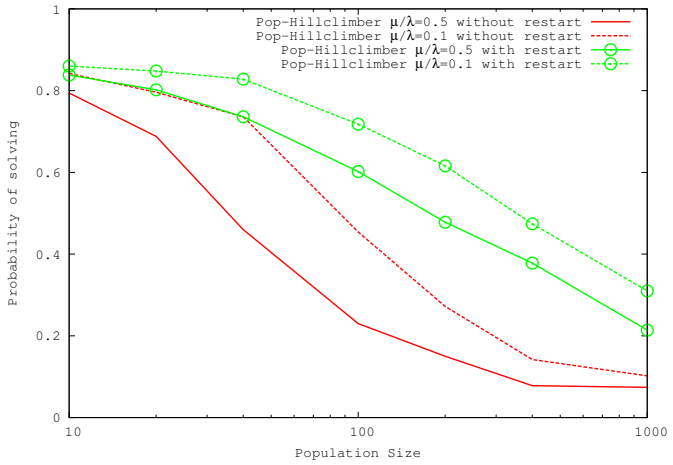

Fig. 7. Probability of the population-based hillclimber solving an instance in function of the population size on Łukasiewicz $\mathbb{T}_4$ problems.

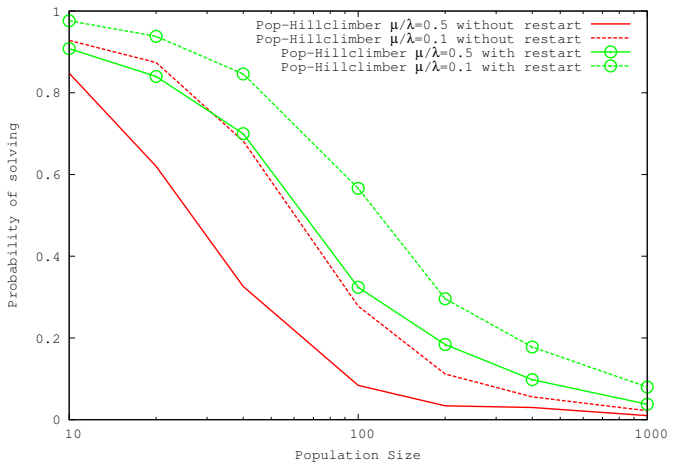

Fig. 8. Probability of the population-based hillclimber solving an instance in function of the population size on Łukasiewicz $\mathbb{T}_{100}$ problems.

from the first hillclimber for the population-based one, and the results validated our assumption (results not included in this paper).

Pop-Hillclimber	Configuration	Success
Łukasiewicz $\mathbb{T}_4$	$\mu = 10$, $\mu/\lambda = 0.5$, no restart	79.4%
	$\mu = 10$, $\mu/\lambda = 0.1$, no restart	84.2%
	$\mu = 10$, $\mu/\lambda = 0.5$, restart	83.8%
	$\mu = 10$, $\mu/\lambda = 0.1$, restart	**86.0%**
Łukasiewicz $\mathbb{T}_{100}$	$\mu = 10$, $\mu/\lambda = 0.5$, no restart	84.4%
	$\mu = 10$, $\mu/\lambda = 0.1$, no restart	92.8%
	$\mu = 10$, $\mu/\lambda = 0.5$, restart	90.8%
	$\mu = 10$, $\mu/\lambda = 0.1$, restart	**97.6%**

TABLE II
PERCENTAGE OF SUCCESSFUL RUNS FOR DIFFERENT POPULATION-BASED HILLCLIMBER CONFIGURATIONS. ALL OBSERVED DIFFERENCES ARE STATISTICALLY SIGNIFICANT (WILCOXON SIGNED RANK TEST).

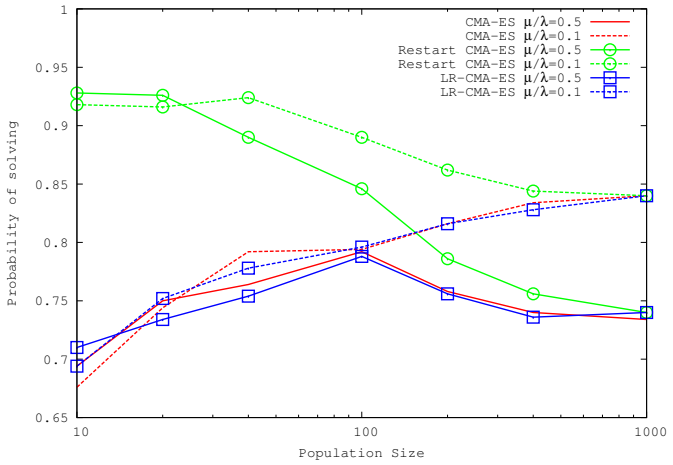

Fig. 9. Probability of CMA-ES solving an instance in function of the population size on Łukasiewicz $\mathbb{T}_4$ problems.

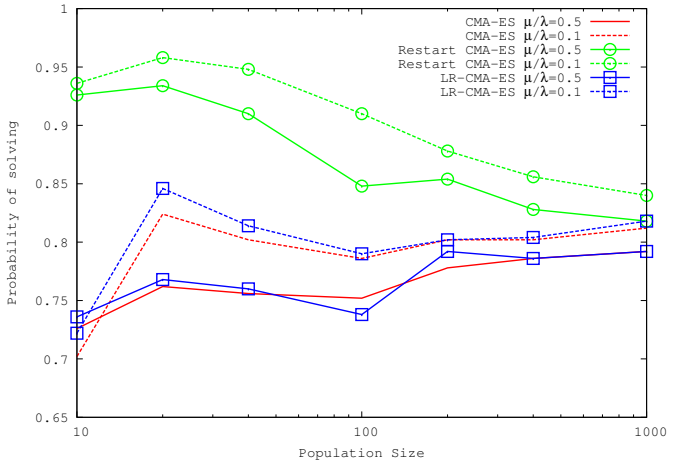

Fig. 10. Probability of CMA-ES solving an instance in function of the population size on Łukasiewicz $\mathbb{T}_{100}$ problems.

C. CMA-ES

Although CMA-ES is always claimed to be virtually parameter free [18], we can still play around with λ, the number of offspring generated each iteration, and the ratio μ/λ, the number of individuals selected each generation versus the number of individuals in that generation as before with the population-based hillclimber. Setting λ is usually left to the implementer of the system, although a guideline of $4+\lfloor 3\ln(n)\rfloor$ is suggested in [18]. For μ it is strongly suggested to set it to $\lambda/2$ [13], yet we found in [12] a μ/λ ratio of 0.1 to work better on the types of problems considered. We perform a similar experiment as the population-based hillclimber experiment before, as we compare CMA-ES, with and without restarting strategies (both the LR-CMA-ES and our '$1000\times\lambda$ iterations without improvement' restart strategy) using μ/λ ratios 0.1 and 0.5, and varying population sizes, to determine the best parameters for later comparison with the hillclimbers.

CMA-ES	Configuration	Success
Łukasiewicz $\mathbb{T}_4$	$\mu = 100$, $\mu/\lambda = 0.5$ CMA-ES	79.2%
	$\mu = 1000$, $\mu/\lambda = 0.1$ CMA-ES	84.0%
	$\mu = 10$, $\mu/\lambda = 0.5$, Restart CMA-ES	**92.8%**
	$\mu = 40$, $\mu/\lambda = 0.1$, Restart CMA-ES	92.4%
	$\mu = 100$, $\mu/\lambda = 0.5$, LR-CMA-ES	78.8%
	$\mu = 1000$, $\mu/\lambda = 0.1$, LR-CMA-ES	84.0%
Łukasiewicz $\mathbb{T}_{100}$	$\mu = 1000$, $\mu/\lambda = 0.5$ CMA-ES	79.2%
	$\mu = 20$, $\mu/\lambda = 0.1$ CMA-ES	82.4%
	$\mu = 20$, $\mu/\lambda = 0.5$, Restart CMA-ES	93.4%
	$\mu = 20$, $\mu/\lambda = 0.1$, Restart CMA-ES	**95.4%**
	$\mu = 200$, $\mu/\lambda = 0.5$, LR-CMA-ES	79.2%
	$\mu = 20$, $\mu/\lambda = 0.1$, LR-CMA-ES	84.6%

TABLE III
PERCENTAGE OF SUCCESSFUL RUNS FOR DIFFERENT CMA-ES CONFIGURATIONS. ALL OBSERVED DIFFERENCES ARE STATISTICALLY SIGNIFICANT (WILCOXON SIGNED RANK TEST).

Figures 9 and 10 show how the likelihood of these variants solving a SAT_∞ instance scales with the population size on the same benchmark problems as before. Table III summarises these results.

Surprisingly, the LR-CMA-ES restart strategy performs almost identical to the standard CMA-ES, suggesting that the conditions for restarting are not suited for the function landscapes encountered in the kind of problems considered here. On the other hand, the simple 'restart after $\lambda \times 1000$ evaluations without improvement' strategy does improve over CMA-ES' performance a lot. The best performing parameter configurations are CMA-ES with our simple restart strategy and $\lambda = 10$, $\mu/\lambda = 0.5$ for the $\mathbb{T}_4$ problem class, and the same with $\lambda = 20$, $\mu/\lambda = 0.1$ for the $\mathbb{T}_{100}$ problem class.

D. Final Comparison

Given this analysis, we compare the best performing parameter configurations for the hillclimbers and CMA-ES by looking at the percentage of successful runs in function of both evaluations and CPU time. The experimental setup is the same, with the same benchmark instances, and a maximum of 10^7 function evaluations, except now does every algorithm solve each instance 50 times instead of 10 in the preliminary experiments, for more significant results.

Figures 11 and 12 show the performance of the algorithms discussed previously with their best parameter configurations in function of the number of function evaluations. Table IV summarizes these results. The most notable result we find is that CMA-ES is marginally, but statistically significantly, outperformed by the population-based hillclimber on the Łukasiewicz $\mathbb{T}_{100}$ problems in terms of successful runs, although CMA-ES needs much fewer function evaluations to come to that result. Also note that this order could again be reversed given a larger budget of function evaluations, as both methods do not seem to have converged yet and thus we cannot conclusively call one method better than the other.

The relative performance on the Łukasiewicz $\mathbb{T}_4$ problems lies more in the line of expectation, where the population-based hillclimber has more successful runs than the hillclimber and needs less function evaluations to get there. CMA-ES finds

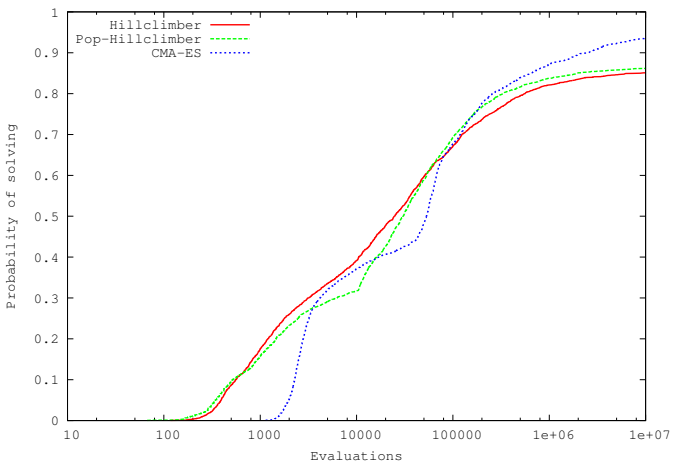

Fig. 11. Probability of solving an instance in function of the number of evaluations for the best population size on Łukasiewicz $\mathbb{T}_4$.

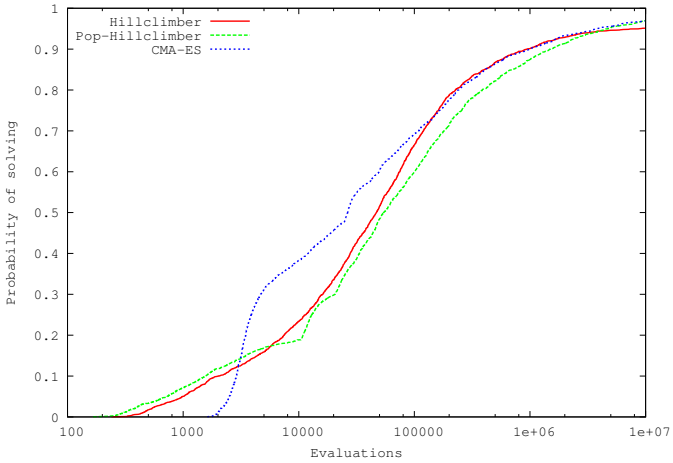

Fig. 12. Probability of solving an instance in function of the number of evaluations for the best population size on Łukasiewicz $\mathbb{T}_{100}$.

more solutions than both hillclimbers, although it needs on average more evaluations to find a solution. One caveat: this average number of evaluations needed is higher for CMA-ES, partly because the hillclimbers start finding solutions an order of magnitude faster, but also because CMA-ES still manages to find solutions near the end of a run, in contrast to the hillclimbers. Therefore, this measure must be viewed with caution. The same is true for the CPU time.

V. CONCLUSION

In previous work, we proposed an optimization approach for solving satisfiability problems in the Łukasiewicz fuzzy logic. We applied the CMA-ES algorithm out-of-the-box, as it is considered state-of-the-art in optimization on a continuous domain, and found it to be a significant contribution to the state-of-the-art in satisfiability solving in fuzzy logics, as it outperformed the standard analytical approach by a large margin on the hard problem class analysed. In this paper, we analysed the performance of different optimization

	Algorithm	Success	Evaluations	CPU
Łukasiewicz $\mathbb{T}_4$	Hillclimber	85.08%	169477.6	22219
	Pop-Hillclimber	86.16%	**144521.8**	**14816**
	CMA-ES	**93.56%**	309105.6	42792
Łukasiewicz $\mathbb{T}_{100}$	Hillclimber	95.12%	**224423.4**	31904
	Pop-Hillclimber	**96.96%**	417474.1	54547
	CMA-ES	96.84%	295679.0	**29516**

TABLE IV
SUMMARY OF THE RESULTS FOR THE HILLCLIMBER, POPULATION-BASED HILLCLIMBER AND CMA-ES. PERCENTAGE OF SUCCESSFUL RUNS, AVERAGE NUMBER OF EVALUATIONS IN A SUCCESSFUL RUN AND CPU TIME OF A SUCCESSFUL RUN ARE RECORDED. ALL OBSERVED DIFFERENCES ARE STATISTICALLY SIGNIFICANT (WILCOXON SIGNED RANK TEST).

algorithms, including CMA-ES, in order to test our assumption that CMA-ES would be the best optimization algorithm for these problems, an assumption based on CMA-ES' status in the community. On the Łukasiewicz $\mathbb{T}_4$ problems, CMA-ES proved to be the best performing algorithm, although this result must be seen in the light of our previous work [12], where CMA-ES did not outperform the state-of-the-art analytical approach on this problem class. The most interesting result described in this paper concerns the Łukasiewicz $\mathbb{T}_{100}$ problems, where CMA-ES is outperformed by the population-based hillclimber we described in this paper.

In future work, we intend to investigate how well this optimization approach works for other fuzzy logics.

ACKNOWLEDGMENT

This work was partially funded by a joint VUB-UGent Research Foundation-Flanders (FWO) project. Also, Tim Brys is funded by a Ph.D grant of the Research Foundation-Flanders (FWO).

REFERENCES

[1] L. Zhang and S. Malik, "The quest for efficient boolean satisfiability solvers," in *Computer Aided Verification*, ser. Lecture Notes in Computer Science, E. Brinksma and K. Larsen, Eds. Springer Berlin / Heidelberg, 2002, vol. 2404, pp. 641–653.

[2] H. Kautz and B. Selman, "Planning as satisfiability," in *Proceedings of the 10th European conference on Artificial intelligence*, ser. ECAI '92. New York, NY, USA: John Wiley & Sons, Inc., 1992, pp. 359–363. [Online]. Available: http://dl.acm.org/citation.cfm?id=145448.146725

[3] N. Tamura, A. Taga, S. Kitagawa, and M. Banbara, "Compiling finite linear CSP into SAT," in *Principles and Practice of Constraint Programming - CP 2006*, ser. Lecture Notes in Computer Science, F. Benhamou, Ed. Springer Berlin / Heidelberg, 2006, vol. 4204, pp. 590–603, 10.1007/11889205_42. [Online]. Available: http://dx.doi.org/10.1007/11889205_42

[4] U. Straccia and F. Bobillo, "Mixed integer programming, general concept inclusions and fuzzy description logics," *Mathware & Soft Computing*, 2007.

[5] S. Schockaert, M. De Cock, and E. E. Kerre, "Spatial reasoning in a fuzzy region connection calculus," *Artificial Intelligence*, vol. 173, no. 2, pp. 258 – 298, 2009. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S000437020800146X

[6] J. Janssen, S. Schockaert, D. Vermeir, and M. De Cock, "Reducing fuzzy answer set programming to model finding in fuzzy logics," *CoRR*, vol. abs/1104.5133, 2011.

[7] T. Lukasiewicz and U. Straccia, "Tightly integrated fuzzy description logic programs under the answer set semantics for the semantic web," in *Proceedings of the 1st international conference on Web reasoning and rule systems*, ser. RR'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 289–298. [Online]. Available: http://dl.acm.org/citation.cfm?id=1768725.1768750

[8] N. Madrid and M. Ojeda-Aciego, "Measuring inconsistency in fuzzy answer set semantics," in *Transactions on Fuzzy Systems*, vol. 19, 2011, pp. 605–622.

[9] D. Van Nieuwenborgh, M. De Cock, and D. Vermeir, "An introduction to fuzzy answer set programming," *Annals of Mathematics and Artificial Intelligence*, vol. 50, pp. 363–388, 2007.

[10] R. Hähnle, "Many-valued logic and mixed integer programming," *Annals of Mathematics and Artificial Intelligence*, vol. 12, pp. 231–263, 1994.

[11] S. Schockaert, J. Janssen, and D. Vermeir, "Satisfiability checking in Łukasiewicz logic as finite constraint satisfaction," *Journal of Automated Reasoning*, 2012. [Online]. Available: http://dx.doi.org/10.1007/s10817-011-9227-0

[12] 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.

[13] 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.

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

[15] N. Mladenović and P. Hansen, "Variable neighborhood search," *Computers & Operations Research*, vol. 24, no. 11, pp. 1097 – 1100, 1997.

[16] T. Bäck, F. Hoffmeister, and H.-P. Schwefel, "A survey of evolution strategies," in *Proceedings of the Fourth International Conference on Genetic Algorithms*. Morgan Kaufmann, 1991, pp. 2–9.

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

[18] N. Hansen and S. Kern, "Evaluating the cma evolution strategy on multimodal test functions," in *Parallel Problem Solving from Nature - PPSN VIII*, ser. Lecture Notes in Computer Science, X. Yao, E. Burke, J. Lozano, J. Smith, J. Merelo-Guervós, J. Bullinaria, J. Rowe, P. Tino, A. Kabán, and H.-P. Schwefel, Eds. Springer Berlin / Heidelberg, 2004, vol. 3242, pp. 282–291.

[19] A. Auger and N. Hansen, "Performance Evaluation of an Advanced Local Search Evolutionary Algorithm." in *IEEE Congress on Evolutionary Computation*, 2005, pp. 1777–1784.