Solving Satisfiability in Fuzzy Logics with Evolution Strategies

Tim Brys*, Yann-Michaël De Hauwere*, Martine De Cock† and Ann Nowé*
*Computational Modeling Lab
Vrije Universiteit Brussel, VUB
Pleinlaan 2, B-1050 Brussels
Email: {tim.brys, ydehauwe, ann.nowe}@vub.ac.be
†Dept. of Applied Math. and Comp. Sc.
Universiteit Gent, UGent
Krijgslaan 281 (S9), B-9000 Gent
Email: martine.decock@ugent.be

***Abstract*—Satisfiability in propositional logic is well researched and many approaches to checking and solving exist. In infinite-valued or fuzzy logics, however, there have only recently been attempts at developing methods for solving satisfiability. In this paper, we propose a new incomplete solver, based on a class of continuous optimization algorithms called evolution strategies. We show experimentally that our method is an important contribution to the state of the art in incomplete fuzzy-SAT solvers.**

I. Introduction

A logical formula, or a set of formulas, is said to be satisfiable if there exists a truth assignment to its variables that makes every formula true. Satisfiability checking is verifying whether such an assignment exists, and satisfiability solving means finding such an assignment. This problem is known as SAT in propositional logic [1] 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, the same principle of satisfiability exists, SAT_∞, and, like its classical counterpart, is 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]–[9]).

Solving satisfiability in fuzzy logics has however received much less attention than its counterpart in classical logics. In [10], Hähnle proposes a mixed integer programming (MIP) approach to satisfiability checking in Łukasiewicz logic, which unfortunately suffers from scalability issues that are inherent to MIP. Schockaert et al. conversely propose a solver in [11] which reduces the infinite-valued logic to a finite-valued one and then applies a constraint satisfaction solver to check satisfiability. This discretization makes the approach ineffective on certain classes of problems. We will elaborate on this further on.

In this paper, we consider satisfiability checking and solving as an optimization problem in a continuous domain. We propose an incomplete solver capable of deciding SAT_∞ but not UNSAT_∞[1], based on the state-of-the-art in evolution strategies (ES), a subclass of evolutionary computation algorithms. In the next sections, we provide the necessary background on fuzzy logic and ES, followed by the formulation of SAT_∞ as an optimization problem. Lastly, we experimentally evaluate our solver and compare it with the incomplete solver from [11], and conclude with a discussion.

II. Fuzzy Logic and SAT_∞

In fuzzy logics [12], 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 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 $\mathtt{T}(1, x) = x$. 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

[1] A set of formulas is $\text{UNSAT}_{(\infty)}$ when there exists no variable assignment that satisfies all formulas.

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 formulas α and β.

An interpretation $\mathcal{I}$ is said to be a model of a set of formulas Θ iff $l \leq [\alpha]_{\mathcal{I}} \leq u$ for every formula $\alpha \in \Theta$, given a lower l and upper bound 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 $\mathcal{I}_1(v_3) = 0.2$ 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$_\infty$ 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$_\infty$.

III. Evolution Strategies

Evolution strategies (ES) [13], [14] are continuous optimization techniques belonging to the class of evolutionary computation. They are algorithms that optimize an objective function, or fitness function, by iteratively applying the principles of selection, mutation and recombination to a population of candidate solutions. One candidate solution is an assignment of the problem's variables, i.e. the input of the objective function. A candidate solution is called an individual and the population of one iteration is a generation.

Each generation, a number of individuals is selected from the population based on their relative fitness, as given by the objective function. Most often, this is the μ best candidate solutions. These selected individuals then serve as the parents for the next generation; the information contained in these individuals is used to generate new solutions. Generally, some parents are recombined into new solutions, similar to sexual reproduction, either by averaging their genes (intermediate recombination) or taking different genes from different parents (discrete recombination). These new solutions or offspring are then mutated to introduce variation into the population. Mutation is achieved by adding a random value from a normal distribution to each gene.

An important aspect of modern ES is the concept of self-adaptation, which allows the algorithm's parameters to co-evolve with the population. In the state-of-the-art CMA-ES (Covariance Matrix Adaptation-Evolution Strategy) [15], the full covariance matrix of the mutation distribution is adapted to fit to the contour lines of the function to be optimized. As such, the probability of getting mutations that improve an individual's fitness is maximized. This algorithm has been shown to perform very well on hard optimization problems, i.e. non-separable, multi-modal functions [16], and will be used as the basis for our SAT$_\infty$ solver. Before continuing, we briefly outline how CMA-ES optimizes an objective function.

CMA-ES

CMA-ES generates λ new candidate solutions x_i from a multi-variate normal distribution with mean m and covariance matrix C (**mutation**):

$$x_i = m + \sigma y_i,\ y_i \sim \mathcal{N}(0, C),\ \text{for } i = 1, ..., \lambda \tag{2}$$

with σ the stepsize, controlling the strength of the mutation.

The mean and covariance matrix are adapted during optimization to perform intelligent exploration of the solution space. The mean is updated every generation by calculating a weighted average (**recombination**) of the μ best solutions (**selection**) in the current population:

$$m \leftarrow \sum_{i=1}^{\mu} w_i x_{i:\lambda} \tag{3}$$

In this formula, $x_{i:\lambda}$ indicates the i'th best solution in the population of size λ, and w_i is a weight assigned to that solution. A reasonable weight assignment satisfies the following equation [15]:

$$\frac{1}{\sum_{i=1}^{\mu} w_i^2} \approx \frac{\lambda}{4} \tag{4}$$

The new distribution mean maximizes the log-likelihood of successful solutions reoccurring, independently of the given covariance matrix.

The covariance matrix of the distribution used to generate new solutions is updated incrementally, based on the accumulated mean shifts of previous search steps (p_c), i.e. the general direction the mean has moved over consecutive steps, and the variation in the μ best new candidate solutions:

$$C^{(g+1)} \leftarrow (1 - c_1 - c_\mu) C^{(g)} + c_1 p_c p_c^T + c_\mu \sum_{i=1}^{\mu} w_i y_{i:\lambda} y_{i:\lambda}^T \tag{5}$$

with c_1 and c_μ controlling the contribution of the various terms to the new covariance matrix ($c_1 + c_\mu \leq 1$).

Essentially, the covariance matrix is adapted to maximize the likelihood of sampling better solutions, given the objective function. Only by using a full covariance matrix can this be achieved, as demonstrated in the rightmost distribution in Figure 1. The incorporation of the evolution path p_c allows the algorithm to move much faster in directions deemed favourable in previous iterations.

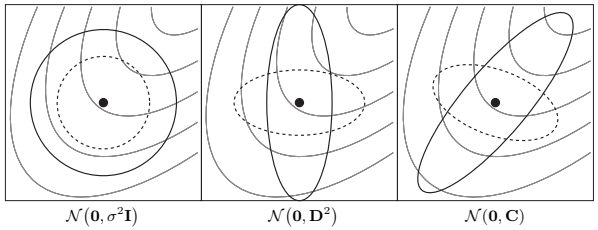

Fig. 1. Six ellipsoids, depicting one-σ lines of equal density of different normal distributions, where $\sigma \in \mathbb{R}_+$, D is a diagonal matrix, and C is a positive definite full covariance matrix. Thin lines depict exemplary objective function contour lines. Figure taken from [15]. The full covariance matrix is clearly best suited at matching the natural gradient of the objective function.

IV. Optimizing SAT_∞

As we will investigate 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 fuzzy formulas α_i, each of which must be satisfied to a certain degree, as defined by an upper and lower bound per formula (the upper bound is usually set to 1). Given these n formulas α_i, bounds (u_i, l_i), and an interpretation $\mathcal{I}$, we define the objective function f as follows:

$$f(\mathcal{I}) = \frac{\sum_i^n f_{\mathcal{I}}(\alpha_i)}{n} \tag{6}$$

and, for $1 \leq i \leq n$,

$$f_{\mathcal{I}}(\alpha_i) = \begin{cases} 1 & \text{if } l_i \leq [\alpha_i]_{\mathcal{I}} \leq u_i \\ \frac{[\alpha_i]_{\mathcal{I}}}{l_i} & \text{if } [\alpha_i]_{\mathcal{I}} < l_i \\ \frac{1-[\alpha_i]_{\mathcal{I}}}{1-u_i} & \text{if } [\alpha_i]_{\mathcal{I}} > u_i \end{cases} \tag{7}$$

with $[\alpha_i]_{\mathcal{I}}$ representing the degree of satisfaction of formula α_i under interpretation $\mathcal{I}$. Each f_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 2.

This function is formulated such that the global maximum 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 decide SAT_∞ sometimes, but never UNSAT_∞[2].

Note that since CMA-ES is a minimization technique, we must negate our objective function to place the problem's solutions in the global minima: $f'(x) = -f(x)$.

[2]CMA-ES converges to the global optimum on a large class of functions as shown empirically but not proven mathematically.

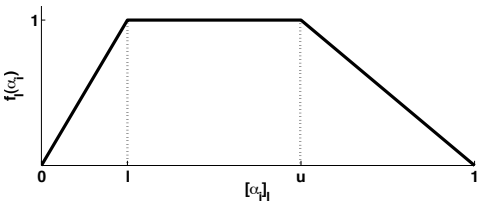

Fig. 2. $f_{\mathcal{I}}(\alpha_i)$ for a formula α_i with lower bound l, upper bound u and degree of satisfaction $[\alpha_i]_{\mathcal{I}}$.

V. Results

In this section, we compare our solver with the constraint satisfaction approach described by Schockaert et al. in [11][3] on the benchmark instances used in that paper. The CSP solver used is called Minion[4] and was also used in the original paper. We start by providing a brief outline of the method from [11] in Section V-A. Next, in Section V-B, we describe the benchmark instances on which the experiments are performed, and finally we present the results in Section V-C.

A. Constraint satisfaction-based SAT_∞ Solver

The reasoning behind the constraint satisfaction approach proposed by Schockaert et al. in [11] is based on two facts:

1) that in infinite-valued Łukasiewicz logic, $\mathcal{L}_\infty$, any satisfiable set of formulas Θ is also satisfiable in some finite-valued logic $\mathcal{L}_d$, with $d \in \mathbb{N}$ truth degrees [17], and,
2) that in a propositional finite-valued logic, every satisfiability problem can be translated into a constraint satisfaction problem (CSP) [11].

Their proposed solver follows the following procedure:

1) Assume the knowledge of the set $\mathbb{T}_k = \left\{0, \frac{1}{k}, \frac{2}{k}, ..., \frac{k-1}{k}, 1\right\}$ from which the constants, including the bounds, in the SAT_∞ problem are drawn.
2) Reduce the problem to a finite-valued logic $\mathcal{L}_k$ with truth degrees $\mathbb{T}_k$.
3) Translate the satisfiability problem in that finite-valued logic into a CSP.
4) If the CSP can be solved, the problem is satisfiable, otherwise, retry in $\mathcal{L}_{2k}$, $\mathcal{L}_{3k}$, etc., gradually refining the truth values.

B. Problem instances

The problem instances considered in [11] are randomly generated using a recursive procedure $form(p)$, which generates a formula in Łukasiewicz logic with a fixed number of variable occurrences. The base case, i.e. one variable in a formula, is as follows:

$$form(1) = \begin{cases} v_i & 50\% \text{ chance} \\ \neg v_i & 50\% \text{ chance} \end{cases} \tag{8}$$

[3]An implementation is available online: http://www.cwi.ugent.be/software/FuzzySAT.tar.gz

[4]http://minion.sourceforge.net/

v_i is a randomly chosen variable from a predefined set of variables $\mathcal{V} = \{v_1, v_2, ..., v_m\}$. The general case, for $p > 1$, is:

$$form(p) = \begin{cases} form(p_1) \otimes form(p_2) & 50\% \text{ chance} \\ \neg(form(p_1) \otimes form(p_2)) & 50\% \text{ chance} \end{cases} \quad (9)$$

with p_1 a random integer between 1 and $p-1$, and $p_2 = p - p_1$.

Note that in Łukasiewicz logic similar dependencies hold between connectives as in classical logic, e.g. $\alpha \oplus \beta = \neg(\neg\alpha \otimes \neg\beta)$ and $\alpha \rightarrow \beta = \neg\alpha \oplus \beta$. Still, as explained in [11], the way of generating test instances for a SAT_∞ solver as described above is somewhat different from what is usually done in the setting of classical SAT. This is due to the fact that not all formulas in Łukasiewicz logic can be converted to conjunctive-normal form, and that, when restricted to formulas in conjunctive-normal form (in the sense that only lower bounds are used, formulas are composed of conjunctions and negations, and all negations occur immediately in front of atoms), the satisfiability problem in Łukasiewicz logic can be reduced to linear programming, and can thus be decided in polynomial time. Hence, to generate interesting test problems, it is crucial to consider formulas which are not in conjunctive-normal form.

To generate bounds for a formula α in a way that creates challenging problems, Schockaert et al. propose the following:

- Randomly choose two interpretations $\mathcal{I}_1$, $\mathcal{I}_2$.
- Generate n formulas α according to the previously outlined method.
- Let λ_1 be the largest value from $\mathbb{T}_4 = \{0,\ 0.25,\ 0.5,\ 0.75,\ 1\}$ smaller than $[\alpha]_{\mathcal{I}_1}$.
- Let λ_2 be the smallest value from $\mathbb{T}_4$ larger than $[\alpha]_{\mathcal{I}_2}$.
- With probability 0.5, add $\alpha \geq \lambda_1$ to the set of formulas, otherwise add $\alpha \geq \lambda_2$.

The set of benchmark problems used in [11] was generated using this method, with the additional constraint that any variable can occur at most once in every formula. Each problem instance consists of 100 formulas ($n = 100$), with five variable occurrences per formula ($p = 5$). Sets of 10, 20, 30 and 40 variables were used, and for each set 50 problem instances were generated.

C. Experiments

Figure 3 shows the results of Minion and CMA-ES solving these benchmark problems. Instances are grouped by the magnitude of runtime (ms) needed to solve them (from bottom to top: 10^2, 10^3, 10^4, 10^5, timeout (undecided)). Timeout was set to 150000 ms, and results for CMA-ES were averaged over 10 runs, to account for its inherent stochasticity. As CMA-ES is virtually parameter-free, we only needed to choose population size λ and the number of individuals μ selected for recombination. We settled on $\lambda = 10$ and $\mu = 1$ as these generally yielded the best results. We can see that our method performs on par with Minion in terms of the number of instances they can solve, but Minion does this generally faster than our method.

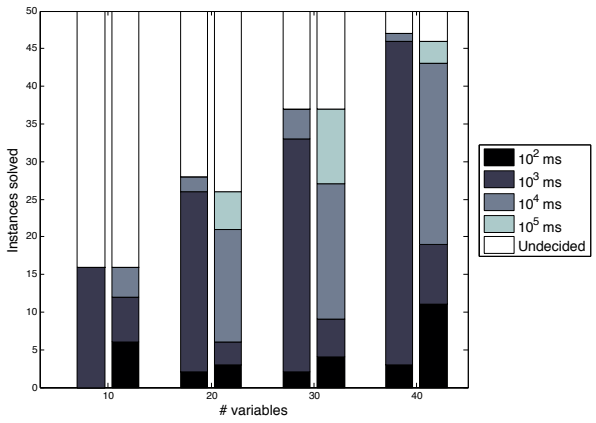

Fig. 3. Results on benchmark problems from [11]. The left bars represent results from Minion, right bars are CMA-ES results. Both methods perform on par in terms of the number of instances they can solve, but Minion does this generally faster than our method.

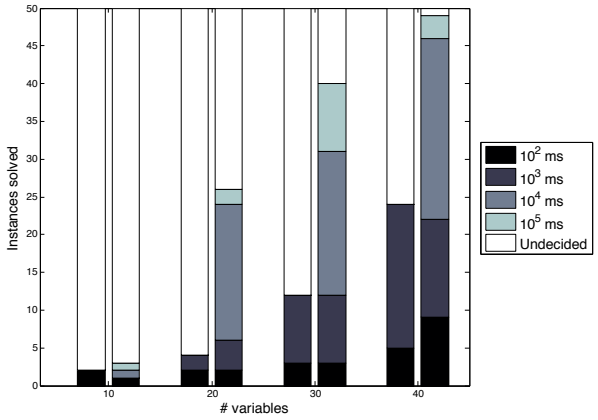

Fig. 4. Results on harder problems with a higher granularity in the bounds for formulas. The left bars represent results from Minion, right bars are CMA-ES results. Our approach is able to solve a significantly higher amount of problems than Minion before timeout.

However, the way these problem instances are constructed hide a drawback of the CSP approach, which is the discretization of a continuous problem. Namely, the upper and lower bounds generated for formulas are drawn from $\mathbb{T}_4$, which gives a low granularity. To test the influence the choice of this set has on performance, we generated a new set of benchmark problems using the same method, only now drawing bounds from the set $\mathbb{T}_{100} = \{0, 0.01, 0.02, ..., 0.99, 1\}$.

The same experiment is performed on this new set of problem instances and yields different results, see Figure 4. It is clear that the CSP approach has difficulties with such fine-grained constants in the formulas, while our approach's performance does not significantly degrade, as it truly models the problem in a continuous domain. The results of both exper-

TABLE I
MINION VS CMA-ES

Instances solved by ...	**Benchmark [11]**	**New benchmark**
Minion	128	42
CMA-ES	125	118
Combined	134	119
Minion faster	89 (15x)	31 (29x)
CMA-ES faster	30 (3x)	10 (2x)

iments are summarized in Table I. Note that combined, these approaches can solve more instances than each individually, i.e. some instances that are solvable by Minion are not by CMA-ES, and vice-versa.

VI. DISCUSSION AND FUTURE WORK

Overall, the CSP approach does not handle fine granularity of constants in the problem well, as it relies on an iteratively refining discretization of a continuous problem, which can lead to an exponential increase in complexity. Our approach does not suffer from this problem. On the other hand, of the problems that both methods can solve, three quarters are solved significantly faster (15x - 29x) by Minion than by CMA-ES, while one quarter of those problems is solved only slightly faster (3x - 2x) by our solver. This, and the fact that some instances can only be solved by either one of the two methods, shows that both methods are valuable and can contribute to a portfolio of SAT_∞ solvers. Note that some of the instances that could not be solved within the time limit by either method are potentially unsatisfiable, and thus can not be decided by these incomplete solvers.

Although our solver was only applied to Łukasiewicz logic in these experiments, it is in essence independent of the underlying logic and its operators. The only requirement for this approach is that a degree of satisfaction can be calculated for a formula and then compared to given bounds for incorporation into an objective function. The CSP approach is less independent of the logic used, as it requires specifications for the translation of formulas and operators in that logic to CSP.

As for future work on this solver, we are currently investigating two ways to improve its performance. First, we noticed that the optimal CMA-ES population size is not the same for all SAT_∞ instances. Some instances that could not be solved with the population size used in the experiments in this paper can easily be solved with a larger population size. Conversely, some instances that were quickly solved with a small population size could not be solved at all with a larger population. Thus, we intend to investigate extending our solver with a restart strategy with increasing population size [18]. This extension allows the CMA-ES algorithm to adapt its population size to best solve a problem instance.

Secondly, we are investigating how we can detect and exploit the underlying structure of a SAT_∞ instance to improve the performance of CMA-ES. We intend to introduce discrete recombination in CMA-ES, so that when several populations are solving a problem in parallel, we can help them escape local optima by allowing them to exchange and recombine optimized problem substructures.

VII. CONCLUSION

Satisfiability solving in fuzzy logics has received little attention compared to its counterpart in classical logics. Therefore, we introduced a novel approach to such a solver in this paper. In contrast to previous more analytical approaches, this incomplete solver models the satisfiability problem as an optimization problem in a continuous domain, using CMA-ES, the state-of-the-art evolution strategy algorithm, as its solution technique. We have empirically demonstrated that this new approach significantly outperforms the current state-of-the-art solver on certain classes of hard problems. Our solver is also not limited to Łukasiewicz logic, to which it was applied in this paper, but can be used to solve satisfiability in a wide range of fuzzy logics.

ACKNOWLEDGMENT

This work was partially funded by a joint VUB-UGent Research Foundation-Flanders (FWO) project.

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, 10.1007/3-540-45657-0_2. [Online]. Available: http://dx.doi.org/10.1007/3-540-45657-0_2

[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, 10.1007/s10472-007-9080-3. [Online]. Available: http://dx.doi.org/10.1007/s10472-007-9080-3

[10] R. Hähnle, "Many-valued logic and mixed integer programming," *Annals of Mathematics and Artificial Intelligence*, vol. 12, pp. 231–263, 1994, 10.1007/BF01530787. [Online]. Available: http://dx.doi.org/10.1007/BF01530787

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

[13] 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.

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

[15] 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.

[16] 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.

[17] D. Mundici, "Satisfiability in many-valued sentential logic is np-complete," *Theor. Comput. Sci.*, vol. 52, pp. 145–153, May 1987. [Online]. Available: http://dx.doi.org/10.1016/0304-3975(87)90083-1

[18] A. Auger and N. Hansen, "A restart CMA evolution strategy with increasing population size," in *The 2005 IEEE International Congress on Evolutionary Computation (CEC'05)*, B. McKay *et al.*, Eds., vol. 2, 2005, pp. 1769–1776.