Evolutionary Logic Trees: Enhancing Logical Formula Optimization with Genetic Programming and Fuzzy Logic

Marcus Vukojevic, Carlotta Giacchetta and Chiara Musso

***Abstract*—This work proposes a method to optimize Knowledge Bases (KBs) by integrating Logic Tensor Networks (LTN) with Genetic Programming (GP). Our approach leverages the strengths of LTNs to represent knowledge as differentiable predicates and constraints, while utilizing GP to iteratively evolve logical rules that improve the overall satisfaction of the KB. Through evolutionary mechanisms such as mutation, crossover, and selection, the proposed algorithm refines the structure of logical formulas to maximize the consistency of the KB, while also introducing novelty and penalizing complexity.**

Results demonstrate that the integration of LTN and GP effectively generates new, high-quality rules while preserving interpretability, scalability, and logical coherence. Moreover, the system is robust to initial knowledge gaps and can discover new insights that align with known facts or general domain knowledge.

I. Introduction

THE rapid growth of knowledge representation systems has brought the challenge of maintaining consistency and coherence in KBs, particularly when they are expanded with new information or rules. Addressing these issues requires methods that not only ensure logical coherence but also allow for the discovery of new insights within the KB. This paper introduces a novel approach that integrates LTN with GP to achieve these goals.

LTN offer a differentiable framework for embedding logical knowledge into a tensor-based system, enabling the evaluation of predicate satisfaction with numerical precision. By representing logical rules as continuous constraints, LTNs allow for the seamless integration of symbolic reasoning and sub-symbolic computation. However, while LTNs excel at assessing formula satisfiability, they lack mechanisms for exploring and evolving new logical structures.

To overcome these limitations, we incorporate GP, a powerful evolutionary optimization technique, to dynamically generate and refine logical rules. GP, with its ability to explore structured and hierarchical solution spaces, addresses the shortcomings of GAs in handling the complexities of logical formulas. Through evolutionary operators such as mutation, crossover, and selection, the proposed framework iteratively evolves rules that maximize the overall consistency of the KB.

A. Introduction to Logic Tensor Networks

LTN provide a framework to integrate logic-based reasoning with continuous optimization. Traditional logic systems evaluate the satisfiability of a formula as binary—true or false. However, this binary nature is insufficient for handling noisy, uncertain, or incomplete KB.

LTN address this by extending classical logic into fuzzy logic, expressing the satisfiability of formulas as values in the range $[0, 1]$. This enables soft evaluations of logical consistency and helps identify contradictions within the KB.

In our work, LTN's fuzzy satisfiability underpins the fitness function, measuring how well generated formulas align with the KB without introducing contradictions. Evolutionary algorithms, such as GP and GA, are used to iteratively refine these logical rules, with LTN evaluating their compatibility.

This integration transforms logical validation into a dynamic, adaptive process, enabling the generation of new and SAT formulas. [1]

II. METODOLOGIES

A. Tree-Based Representation and genetic operations of Logical Formulas - general framework

Before delving into the techniques employed in our evolutionary algorithm, it is essential to explain how a logical formula can be encoded in a genotype. To enable the dynamic generation and manipulation of logical formulas, we represent these formulas as syntactic trees. Each tree is composed of nodes, categorized as follows:

- **Operators** ($\wedge$, $\vee$, $\implies$): Internal nodes that connect subformulas.
- **Quantifiers** ($\forall$, $\exists$: Nodes that define the scope of variables within the formula.
- **Predicates** $(\text{Fly}, \text{Animal}, \text{Bird}, \text{Penguin}, \text{Swallow})$: Leaf nodes representing logical statements.
- **Variables** (x): The entities involved in the formulas.
- **Constants** $(\text{Marcus}, \text{Tweety})$: The entities involved in the formulas.

Example: Here is an example of a formula represented as a tree structure:

$$\forall x\,(\text{Bird}(x) \implies \text{Animal}(x))$$

This formula asserts that for all entities x, if x is a `Bird`, then x it is also an `Animal`. Its corresponding tree representation consists of:

- A quantifier node ($\forall$),
- A variable node (x),
- An operator node ($\implies$),
- Two predicate nodes ($\text{Bird}(x)$ and $\text{Animal}(x)$).

Each tree is constructed recursively, starting from a root quantifier node and expanding through operators and predicates. This hierarchical structure allows for efficient parsing, manipulation, and evaluation of formulas within the LTN framework.

1) Initial Population: To initialize the population in our evolutionary algorithm, we explore two distinct representations:

- **Classic population (list-based):** In this representation, the population consists of a simple list of individuals. Each individual is independent and does not have neighbors. Parent selection for reproduction occurs across the entire population.
- **Matrix population:** In this case, the population is represented as a matrix, where each individual has a set of neighbors. Parent selection is restricted to the neighbors of a given individual, promoting localized interactions.

2) Parent Selection: Determines which individuals from the current population will contribute to the next generation. In our approach, we employ two methods for parent selection: *fitness-proportionate selection* and *modern over-selection*, each designed to balance exploration and exploitation of the search space.

a) Fitness-Proportionate Selection: This method assigns a selection probability to each individual proportional to its fitness. This approach ensures that individuals with higher fitness have a greater chance of being selected.

b) Modern Over-Selection: The modern over-selection technique introduces an additional layer of control by dividing the population into two groups based on fitness: the top group (x% of the population) contains the best-performing individuals, while the bottom group (100-x% of the population) contains the rest of the population.

The selection process prioritizes the top group, with 80% of selections made from this group and 20% from the bottom group.

3) Mutation: The mutation operator alters a randomly selected subtree based on the node type:

- For binary operators, the operator can be replaced (e.g., $\wedge \to \vee$).
- For predicates, the name or arity can be modified (e.g., $\text{Bird}(x) \to \text{Penguin}(x)$).
- Predicates can also be wrapped with a unary operator $\neg$ (e.g., $\text{Fly}(x) \to \neg\text{Fly}(x)$).
- Alternatively, predicates can be expanded into a binary operator with a new predicate (e.g., $\text{Fly}(x) \to (\text{Fly}(x) \wedge \text{Penguin}(x))$).

4) Crossover: The crossover operator swaps compatible subtrees between two parent trees, ensuring structural consistency.

Example:

Parent A: $\forall x(\text{Animal}(x) \wedge \text{Fly}(x)) \implies \text{Bird}(x)$

Parent B: $(\neg\text{Fly}(x)) \vee \text{Penguin}(x)$

After swapping the subtrees $(\text{Animal}(x) \wedge \text{Fly}(x))$ and $(\neg\text{Fly}(x))$:

Child A: $\forall x(\neg\text{Fly}(x)) \implies \text{Bird}(x)$

Child B: (Animal(x) $\wedge\text{Fly}(x)) \vee \text{Penguin}(x)$

5) Fitness: For each method we have a different way to compute the fitness, see sections II-C1,II-D.

6) Final Population: As for the fitness function we employ different approaches to update the population, as you can see in sections II-C2, II-D

B. Knowledge Base

The KB used in this study is shared across all three experiments and consists of the following logical rules [2]:

- *Birds can fly:* $\forall x\,(\text{Bird}(x) \implies \text{Fly}(x))$
- *Penguins cannot fly:* $\forall x\,(\text{Penguin}(x) \implies \neg\text{Fly}(x))$
- *Birds are animals:* $\forall x\,(\text{Bird}(x) \implies \text{Animal}(x))$
- *Penguins are birds:* $\forall x\,(\text{Penguin}(x) \implies \text{Bird}(x))$
- *Swallows are birds:* $\forall x\,(\text{Swallow}(x) \implies \text{Bird}(x))$

For clarity, Marcus is a swallow and Tweety is a penguin. Although the KB is logically consistent, it contains an interesting contradiction from a domain perspective: if penguins are birds and birds can fly, one might conclude that penguins should be able to fly as well; yet this is explicitly not the case. This inherent imperfection makes the KB an intriguing starting point for our experiments [3].

C. EXP 1: GENERATING SAT FORMULAS

The primary objective of this first experiment is to generate all possible SAT formulas for the given KB. In this context, a SAT formula is defined as a logical formula that is consistent with the given KB, meaning it does not introduce contradictions and adheres to the logical rules of the domain.

To achieve this, we employ **two complementary strategies**: the first using GA and the second using GP.

The two strategies are fundamentally similar, with the key difference lying in how the population is represented. For GA, the initial population is represented as logical trees (see Section II-A), which are then converted into string representations to simplify genetic operations. In contrast, GP directly operates on logical trees, maintaining their hierarchical structure throughout the evolutionary process. For both cases, we use both the matrix and list representation methods to represent the population, as explained in Section II-A1.

1) Fitness function: The fitness function evaluates how well a generated formula aligns with the KB without introducing contradictions. To achieve this, the formula is converted into a representation compatible with LTN, which provides a soft, fuzzy logic evaluation of the formula's satisfiability. The fitness of a formula is computed as the degree of its satisfiability, ranging between 0 (completely unsatisfiable) and 1 (fully satisfiable).

Additional penalties are introduced in the fitness function to ensure the generation of high-quality formulas:

- **Predicate uniqueness:** Formulas with duplicate predicates are penalized to encourage diversity within the logical structure and to avoid trivial or tautological formulas, such as $\text{Penguin}(x) \to \text{Penguin}(x)$
- **Tree depth:** Excessively deep trees are penalized to prevent overly complex and unintelligible formulas. This is particularly important in our context, where we work with a single variable, as deeper trees often result in redundant or nonsensical logical structures.

2) Final population: In the matrix-based approach, the children with the highest fitness directly replace the parent. In contrast, the list-based method involves creating a list of all children, sorting them based on fitness, and then selecting the top `num_offspring` children to form the new population.

By iteratively evaluating and refining the population of formulas using mutation (II-A3) and crossover (II-A4), GA and GP identifies those that maximize the fitness function, ensuring consistency with the KB and revealing implicit logical relationships within the domain.

D. EXP 2: DISCOVERING NEW RULES FOR THE KB

The second experiment focuses on the discovery of new logical formulas to expand the existing KB. These new formulas aim to improve the overall logical coherence and satisfaction of the KB by introducing previously implicit relationships between concepts. The evolutionary process begins by initializing a population of formulas II-A1, represented as tree structures composed of logical operators, quantifiers, and predicates. Each candidate formula is evaluated based on its impact on the KB using a fitness function defined as:

$$\begin{aligned}\text{Fitness} = \Delta\text{SAT} + \lambda_{\text{nov}} \cdot \text{Nov} - \lambda_{\text{compl}} \cdot \text{Compl} \\ - \text{SinglePred} - \text{Taut} - \text{Repet}\end{aligned}$$

where:

- ΔSAT measures the improvement in the KB's satisfaction after adding the formula, compared to its baseline satisfaction.
- *Novelty* rewards the uniqueness of the formula, penalizing duplicates of previously discovered rules.
- *Complexity* penalizes overly complex formulas, ensuring that new rules remain interpretable and computationally manageable.
- *SinglePred* penalizes the formulas where only single predicates are used.
- *Taut* penalizes simple tautological formulas (for example: $\text{pred}(x) \lor \neg\text{pred}(x)$ or $\text{pred}(x) \implies \text{pred}(x)$).
- *Repet* penalizes the formulas where the predicates are repeated multiple times.

Every 10 generations, the highest-fitness formula is evaluated for inclusion into the KB. A formula is added only if it satisfies the following conditions:

- The formula achieves a fitness score that exceeds a predefined threshold (e.g., 0.98).
- The formula introduces a significant improvement in the satisfaction of the KB ($\Delta\text{SAT} > 0$).
- The formula is not a duplicate of any previously added rule.

When a formula is added to the KB, the system updates the KB, recalculates its baseline satisfaction, and performs a partial retraining of the predicates to effectively integrate the new rule.

III. DIFFICULTIES

Generating meaningful and interpretable formulas was challenging, as overly complex ones often became nonsensical with a single variable. To address this, we calibrated the fitness function to balance complexity and predicate uniqueness.

Maintaining population diversity was also difficult due to the limited predicates and variables in the KB, leading to premature convergence. Adjustments to selection and mutation strategies, such as probabilistic parent selection and higher mutation rates, helped prevent stagnation.

Integrating generated formulas into the KB required ensuring consistency and avoiding contradictions. Only unique, high-fitness formulas were included, using clear criteria to maintain logical coherence.

A. Limitations of Genetic Algorithm

Our experiments revealed that GA are less effective for this problem because GA's reliance on linear encoding of formulas further restricts flexibility, often resulting in redundant or trivial solutions. This inefficiency arises because GA lacks information about predicates, operators, quantifiers, and the overall structure of logical formulas. Consequently, when performing crossover and mutation, these operations are applied randomly, without considering the syntactic or semantic correctness of the formulas, leading to meaningless results.

For example, GA may generate invalid formulas such as:

- Swallow $\lor$ ($\forall x \lor$ Animal$(x) \land$)
- Penguin $\lor$ $\land$

These formulas lack logical coherence and highlight the limitations of GA in this context, as it fails to create meaningful and interpretable formulas during the evolutionary process.

IV. RESULTS

In this section, we present the results of our experiments, evaluating the effectiveness of the proposed approach in generating satisfiable formulas and discovering new rules to enhance the KB.

A. EXP1: Generating SAT Formulas

Using GP, we performed multiple runs, varying parameters and methods:

TABLE I
EXPERIMENTAL RUNS AND PARAMETERS

Run	Population Size	Generations	Number of Offsprings
Run 1	16	20	5
Run 2	16	50	5
Run 3	49	20	20
Run 4	49	50	20
Run 5	16	100	5
Run 6	49	100	20
Run 7	100	100	40

Note: Each run was repeated 4 times, combining selection methods (fitness proportionate and over-selection) with different population structures (matrix-based or list-based). For all runs, the crossover probability was fixed at 0.8, with mutations of 0.2 and 0.4 applied to the first and second offspring, respectively.

At the end of each run, a list of the best individuals (fitness value of 1) was created, representing the SAT individuals. The total number of SAT individuals in this list was counted and recorded.

The table below summarizes the number of SAT individuals for each run:

TABLE II
NUMBER OF SAT INDIVIDUALS ACROSS DIFFERENT RUNS

Method	**Run1**	**Run2**	**Run3**	**Run4**	**Run5**	**Run6**	**Run7**
Matrix+FitProp	4	2	12	15	5	20	18
Matrix+OverSel	5	5	22	18	7	18	38
List+FitProp	4	4	13	30	3	24	57
List+OverSel	5	4	17	27	1	24	63

Note: FitProp refers to Fitness Proportionate Selection, and OverSel refers to Over-Selection.

It is important to note that the runs exhibit significant variability: repeating the same run (e.g., Run 1) multiple times may yield different outcomes due to the stochastic nature of evolutionary algorithms.

From Table II, it can be observed that different selection methods (FitProp and OverSel) and population representations (Matrix and List) exhibit varying performance depending on the configuration. For instance, List+OverSel tends to achieve better results in runs with larger populations and more generations (e.g., Run 7). However, the results are not consistent: in some runs with smaller populations (e.g., Run 5), the number of SAT individuals generated is relatively low across all methods.

This variability highlights the importance of performing multiple repetitions for each configuration and analyzing the results in aggregate to draw reliable conclusions.

A complete list of the SAT formulas generated during the experiments can be found in Appendix A.

B. EXP2: Discovering New Rules for the KB

In the second experiment, we aimed to discover new logical rules that could enhance the existing KB by increasing its SAT score. Unlike the first experiment, we removed the *Existential Quantifier* operator ($\exists$) from the logical tree representation. This decision was made to prioritize the discovery of general rules applicable to the entire KB, as the inclusion of the existential quantifier often led to rules valid for isolated cases, which were less useful in improving the KB's overall coherence.

The primary objective of this experiment was to use GP in conjunction with the fitness function defined earlier to identify rules that, when added to the KB, would increase its SAT score. The initial average SAT score of the KB was 0.83, and through this process, we achieved a maximum SAT score of 0.93 ± 0.03.

We experimented with different run configurations, varying the population size of generated formulas. Through these tests, we observed that the optimal population size is highly dependent on the size of the KB. For smaller KBs, having a smaller population size (around 30) was particularly beneficial, as it led to better performance and quicker convergence. Conversely, for larger KBs, increasing the population size resulted in noticeable improvements, likely due to the higher diversity of formulas generated. All runs were performed with 100 generations and the hill climbing method.

All the formulas added to the KB were validated to be logically correct and consistent. Each formula contributed to improving the SAT score, effectively acting as an integration of previously implicit relationships that the KB "should" have known but had not explicitly expressed. This approach is particularly beneficial for large-scale KBs, where discovering new SAT formulas manually can be both challenging and computationally expensive.

In Appendix B, we provide a sample of the complete output of the experimental run. Below, we highlight the most interesting formulas that were added to the KB and their contribution to the improvement of SAT:

Key Formulas Added to the KB:

1. $\forall x \left((\text{Penguin}(x) \lor \text{Fly}(x)) \implies \neg \text{Bird}(x) \right)$
2. $\forall x \left(\text{Animal}(x) \lor (\text{Penguin}(x) \land \text{Fly}(x)) \right)$.
3. $\forall x \left(\text{Fly}(x) \land \neg \text{Penguin}(x) \right)$.
4. $\forall x \left(\text{Animal}(x) \lor (\text{Fly}(x) \implies \text{Penguin}(x)) \right)$,
5. $\forall x \left(\text{Fly}(x) \implies (\text{Bird}(x) \lor \text{Animal}(x)) \right)$.

These formulas not only improved the SAT score but also demonstrated the system's ability to infer general rules that align with the logical coherence of the domain. Such results underscore the utility of this approach in expanding the logical scope and expressiveness of KBs.

V. CONCLUSIONS

We proposed an innovative method to optimize KBs by combining LTN with GP. This approach leverages the differentiable predicates of LTNs and the evolutionary capabilities of GP to dynamically generate and refine logical rules.

The experiments showed that integrating LTN and GP effectively improves KB consistency and satisfaction, addressing knowledge gaps by discovering logical relationships aligned with domain knowledge.

Key contributions include:

- **Integration of LTN and GP**: Merging numerical evaluations with rule discovery for KB optimization.
- **Adaptive Rule Evolution**: Penalizing redundancy while encouraging high-quality rule generation.
- **Addressing Knowledge Gaps**: Resolving inconsistencies and enhancing incomplete KBs.

Future work will focus on:

- **Scalability**: Reducing complexity for larger KBs and diverse applications (e.g., medicine, robotics).
- **Integration with Learning Methods**: Enhancing knowledge discovery with supervised/unsupervised techniques.
- **CNF Formulas**: Exploring robustness with formulas in conjunctive normal form.

CONTRIBUTIONS

Carlotta and Chiara primarily focused on the first part of the experiment IV-A, respectively on GP and GA, while Marcus concentrated on the second part IV-B.

Carlotta also implemented the population_init function, the selection methods for the genetic algorithm, and the structure of the knowledge base. Marcus was responsible for designing and building the `Node` and `Albero` structures, which represent logical formulas. Chiara worked on comprehensive suite of utility functions, with particular attention to converting logical trees into string representations and ensuring compatibility with the LTN framework.

Appendix A
SAT formulas generated with GP

This section presents the satisfiable (SAT) formulas generated during *Run 3* using the *list population representation* combined with the *fitness proportionate selection method*. These formulas represent logical relationships that are consistent with the given KB and satisfy all constraints without introducing contradictions.

Below is the list of SAT formulas generated during the run.

$$\begin{aligned}
&\exists x\,\big(\text{Fly}(x) \implies \text{Animal}(x)\big)\,,\\
&\exists x\,\big((\text{Fly}(x) \land \text{Penguin}(x)) \implies \text{Bird}(x)\big)\,,\\
&\exists x\,\big((\neg\text{Bird}(x)) \implies (\neg\text{Fly}(x))\big)\,,\\
&\exists x\,\big((\neg(\text{Fly}(x) \lor (\neg\text{Swallow}(x)))) \implies \text{Animal}(x)\big)\,,\\
&\exists x\,\big(\text{Penguin}(x) \implies (\text{Fly}(x) \implies \text{Animal}(x))\big)\,,\\
&\exists x\,\big((\text{Swallow}(x) \implies \text{Animal}(x))\\
&\quad \implies ((\neg\text{Bird}(x)) \implies (\neg\text{Penguin}(x)))\big)\,,\\
&\exists x\,\big(\text{Swallow}(x) \lor (\text{Fly}(x) \implies \text{Bird}(x))\big)\,,\\
&\exists x\,\big((\neg\text{Swallow}(x)) \implies \text{Animal}(x)\big)\,,\\
&\exists x\,\big(\text{Fly}(x) \implies \text{Penguin}(x)\big)\,,\\
&\exists x\,\big((\neg\text{Bird}(x)) \implies (\neg(\text{Fly}(x) \land \text{Penguin}(x)))\big)\,,\\
&\exists x\,\big(\text{Bird}(x) \implies \text{Swallow}(x)\big)\,,\\
&\exists x\,\big(\text{Fly}(x) \implies (\text{Penguin}(x) \implies \text{Swallow}(x))\big)\,,\\
&\exists x\,\big(\text{Fly}(x) \implies \text{Swallow}(x)\big).
\end{aligned}$$

Appendix B
New Rules for the KB

Below is the list of SAT formulas generated during the run

$$\begin{aligned}
&\forall x\,\big((\text{Penguin}(x) \lor \text{Bird}(x)) \implies \text{Animal}(x)\big)\,,\\
&\forall x\,\big(\text{Fly}(x) \implies (\text{Animal}(x) \lor \text{Penguin}(x))\big)\,,\\
&\forall x\,\big((\text{Penguin}(x) \implies \text{Animal}(x)) \lor \text{Fly}(x)\big)\,,\\
&\forall x\,\big((\text{Penguin}(x) \land \text{Fly}(x)) \implies \text{Bird}(x)\big)\,,\\
&\forall x\,\big((\text{Animal}(x) \land \text{Fly}(x)) \implies \text{Penguin}(x)\big)\,,\\
&\forall x\,\big((\text{Penguin}(x) \implies \text{Bird}(x)) \lor \text{Fly}(x)\big)\,,\\
&\forall x\,\big(\text{Swallow}(x) \lor (\text{Fly}(x) \implies \text{Penguin}(x))\big)\,,\\
&\forall x\,\big((\text{Fly}(x) \land \text{Bird}(x)) \implies \text{Penguin}(x)\big)\,,\\
&\forall x\,\big((\text{Bird}(x) \implies \text{Animal}(x)) \lor \text{Fly}(x)\big).
\end{aligned}$$

References

[1] S. Badreddine, A. d'Avila Garcez, L. Serafini, and M. Spranger, *Logic Tensor Networks*. arXiv preprint arXiv:2012.13635, 2021. [Online]. Available: https://arxiv.org/abs/2012.13635

[2] *Evolutionary Logic Trees: Enhancing Logical Formula Optimization with Genetic Programming and Fuzzy Logic*, Journal of Unitn Bioinspiredclass, vol. 14, no. 8, pp. 1–5, Aug. 2015.

[3] K. Katsurada, M. Koyama, K. Ohara, N. Babaguchi, and T. Kitahashi, "Solving contradiction in knowledge-base without interaction," in *SMC'98 Conference Proceedings. 1998 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No. 98CH36218)*, vol. 2, pp. 1546–1551, IEEE, 1998.