ISSN 2007-9737

Policy Gradient MaxSAT Solver

Omar Gutierrez-De-La-Paz, Ricardo Menchaca-Méndez*, Erik Zamora-Gómez, Uriel Corona-Bermudez, Rolando Menchaca-Méndez, Bruno Gutiérrez-De-La-Paz

Instituto Politécnico Nacional,
Centro de Investigación en Computación,
Mexico

{ogutierrezd2019, ric, rmen, bgutierrezd2019}@cic.ipn.mx, {ezamorag, ucoronab}@ipn.mx

Abstract. This paper presents a comparative study of various elements and strategies that can be incorporated into an autoregressive model to address the MaxSAT problem. Building upon a sequential architecture as our foundation, we optimize the model's parameters by maximizing the expected number of satisfied clauses. This optimization enables the model, given a SAT formula, to predict a distribution over potential solutions using the policy gradient method. Our controlled experiments pinpoint elements that guide the optimization process towards superior results[1].

Keywords. MaxSAT problem, policy gradient, NP-hard.

1 Introduction

The Maximum Satisfiability Problem (MaxSAT) is an optimization variant of the Boolean Satisfiability Problem (SAT). Its goal is to identify a truth assignment that maximizes the number of satisfied clauses in a given boolean formula. MaxSAT represents a classical challenge in computational theory and has wide-ranging applications across numerous domains due to its generic representation of optimization problems [10, 16, 24, 11]. While recent advances in machine learning, especially deep learning, provide promising approaches to combinatorial problems [5, 4, 25, 19], the adaptation of these techniques to the MaxSAT problem is still an emerging area of study.

This research delves into the comparative analysis of various elements and strategies, all aimed at enhancing the efficiency of an autoregressive model tailored to address the MaxSAT problem. Central to our exploration is the role of sequential models that capture current and past relevant information to generate convenient variables' values, baseline methods used to mitigate variance during optimization, and the technique used to represent variables and incorporate semantic information.

By systematically examining these components through our controlled hyperparameter searches across a diverse set of random 3-SAT instances, we aim to figure out the contributions of each element to the solution quality. The emphasis is on discerning the configurations that consistently yield superior results.

A crucial aspect of our methodology is the employed optimization technique. We leverage the policy gradient theorem, optimizing the parameters of the model on individual MaxSAT instances by maximizing the expected number of satisfied clauses to learn a distribution over potential solutions. This process dynamically adjusts the probability associated with variables taking specific values, true or false, as illustrated in fig. 1. It is crucial to underscore, however, that the ambition of this paper isn't to compete with the state-of-the-art solvers such as the MaxHS [9, 3], Open-WBO [22, 23], EvalMaxSAT [2], or Loandra [6].

Instead, our goal is centered on discovering strategies and elements that enhance the effectiveness of Deep Learning approaches.The remainder of the paper is organized as follows. In Section 2, we review recent approaches to

[1] https://github.com/omargup/Policy-Gradient-MaxSAT-Solver

ISSN 2007-9737

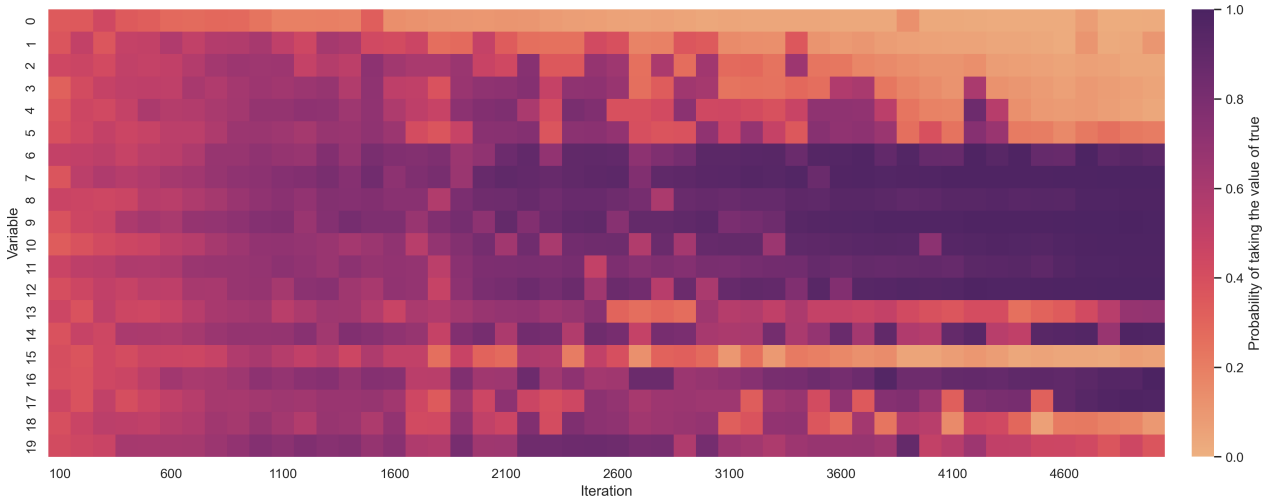

Fig. 1. Evolution of the probability that variables assume the value of true, $p(x_i = 1)$, over 5000 iterations for a random 3-SAT instance with 20 variables and 90 clauses. The ith row showcases the changes in the probability of variable x_i being true throughout the optimization process. Purple indicates a high probability of the value being true, whereas yellow denotes a lower probability (i.e., a higher likelihood of the value being false). At the beginning, there is uncertainty regarding the values the variables will assume; however, as the process unfolds, the probabilities converge towards either one or zero

solving the MaxSAT and related problems with Deep Learning. Subsequently, in Section 3, we detail the problem, the optimization process, and the configurable components of our model. In Section 4, we outline the proposed experiments and, finally, in Section 5, we discuss our findings.

2 Related Work

A variety of Machine Learning-based methods for combinatorial optimization problems have been developed that construct solutions sequentially through Reinforcement Learning [4, 17, 19]. These methods are well-suited for integration with search strategies like sampling and beam search, providing strong guidance in the search process. Bello et al. [4] introduced a generic search strategy known as active search. This strategy allows for guided exploration in solution construction without the need for problem-specific components. Unlike traditional methods that only sample solutions at inference time with a fixed model, active search operates iteratively on a single test input, modifying the model's parameters to increase the likelihood of generating high-quality solutions in future iterations.

Their focus was predominantly on the Traveling Salesman Problem (TSP). In their study, Bello et al. compared two methodologies of active search. The first approach involves adjusting the weights of a pre-trained model specific to a test instance, utilizing Reinforcement Learning. The second approach initiates active search on a single instance with an untrained model.

Their findings showed enhanced performance compared to random sampling when starting the search with a pre-trained model. Notably, initiating the search with an untrained model also led to satisfactory results. Hottung et al. [15] developed and evaluated three efficient active search strategies that update only a selected subset of parameters during the search. This approach contrasts with the full-scale model weight updates implemented in [4].

Their results indicated significant enhancements in search performance of models, outperforming some of the leading Machine Learning-based methods in various combinatorial problems, such as the TSP, Capacitated Vehicle

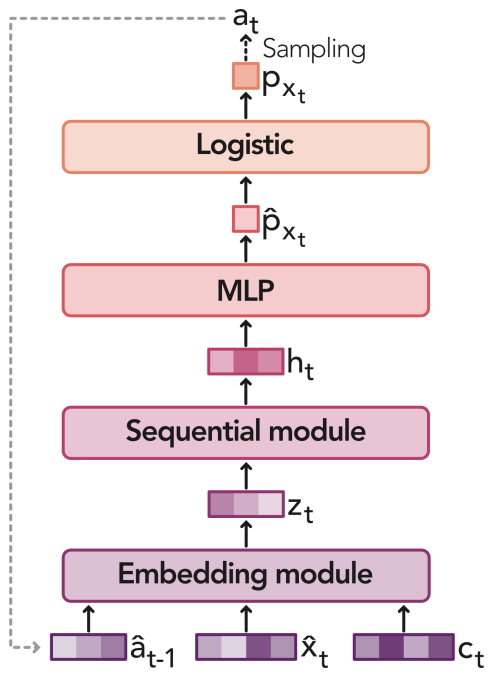

Fig. 2. High-level description of the model. At each time step t, the model ingests a representation $\hat{x}_t$ of the current variable, a representation $\hat{a}_{t-1}$ of the previous variable's value, and the context c_t. It then produces a 1-dimensional output p_{x_t} in the range [0,1] which is interpreted as the probability that the variable's value is true. Subsequently, the truth value a_t for x_t is sampled from p_{x_t}

Routing Problem (CVRP), and Job Shop Scheduling Problem (JSSP). Our research explores the latter approach in [4], particularly focusing on active search using an untrained model for a single input.

We aim to exploit the capabilities of neural networks, gradient descent, and Reinforcement Learning as distribution-independent optimization tools to find efficient solutions for the MaxSAT problem by transitioning the optimization approach from a discrete to a continuous domain. While Hottung et al. focused on improving active search by updating a subset of parameters starting from a trained model, our research seeks to enhance active search beginning with an untrained model. We explore a range of architectural and optimization elements. For example, while Bello et al. used an exponential moving average (EMA) baseline, we extend our exploration to include various sequential models, multiple baselines (including EMA), and the benefits of incorporating problem-specific information into the input via Node2Vec.

3 Methods

3.1 Problem Definition

A boolean variable x_i can assume values true (one) or false (zero). A literal l_i refers to the variable x_i or its negation $\neg x_i$. A clause c is represented as a disjunction (logical OR) of literals. A Conjunctive Normal Form (CNF) formula ϕ is expressed as a conjunction (logical AND) of clauses.

The SAT problem requires determining a truth assignment π that provides a truth value (true or false) to every variable in a CNF formula ϕ, such that all clauses are satisfied, or stating that no such assignment exists. The MaxSAT problem is an optimization variant of SAT that seeks an assignment which maximizes the number of satisfied clauses in ϕ.

Thus, even if not all clauses in the formula can be simultaneously satisfied, MaxSAT will identify an assignment that satisfies the maximum possible number of clauses. In this work, our primary focus is on the MaxSAT problem. Formally, given a CNF formula ϕ containing n variables and m clauses, our objective is to identify a truth assignment π maximizing the number of satisfied clauses, represented as $S(\pi|\phi)$.

3.2 Objective Function and Optimization Procedure

We employ an autoregressive model to approximate optimal solutions. Let $S(\pi^*|\phi)$ be the optimal number of satisfiable clauses, and $p_\theta(\pi|\phi)$ be a multi-dimensional distribution with learnable parameters θ.

Our aim is to approximate $S(\pi^*|\phi)$ by maximizing the expected number of satisfiable

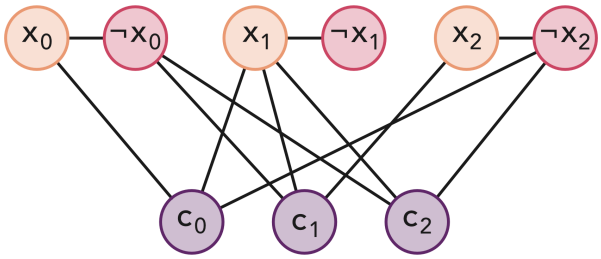

Fig. 3. Graph representation of the formula for the Node2Vec algorithm. The illustration corresponds to the formula $(x_0 \vee x_1 \vee \neg x_2) \wedge (\neg x_0 \vee x_1 \vee x_2) \wedge (\neg x_0 \vee x_1 \vee \neg x_2)$. Nodes symbolize literals and clauses. Undirected edges link literals to the clauses they appear in, while supplementary edges connect literals of the same variable

clauses $S(\pi|\phi)$ under the distribution $p_\theta(\cdot|\phi)$ represented by the model, i.e.:

$$S(\pi^*|\phi) \approx \max_\theta \mathbf{E}_{\pi\sim p_\theta(\cdot|\phi)} S(\pi|\phi), \tag{1}$$

where

$$J(\theta|\phi) = \mathbf{E}_{\pi\sim p_\theta(\cdot|\phi)} S(\pi|\phi). \tag{2}$$

Is the objective function.

To maximize this objective function for a specific instance of the MaxSAT problem, we optimize the model's parameters using stochastic gradient ascent using the ADAM optimizer [18], leveraging the policy gradient theorem. The gradient of $J(\theta|\phi)$ with respect to θ is given by:

$$\nabla_\theta J(\theta|\phi) = \mathbf{E}_{\pi\sim p_\theta(\cdot|\phi)} \left[S(\pi|\phi) \nabla_\theta \log p_\theta(\pi|\phi) \right], \tag{3}$$

where, due to the autoregressive nature of the model, the probability of an assignment can be factorized using the chain rule:

$$p_\theta(\pi|\phi) = \prod_{i=1}^{n} p_\theta(\pi_i|\pi_{i-1}, \pi_{i-2}, \ldots, \pi_1). \tag{4}$$

And hence, we can approximate the gradient by sampling assignments π from the model.

3.3 Architecture

To address the MaxSAT problem, we employ a sequential model. This model accepts as inputs a representation of the formula's variables and another representation of the formula itself, termed as context. Sequentially, for each variable, it produces an output probability from which the truth value of that variable is sampled.

The design of the model is autoregressive [12], ensuring that each step takes into account a representation of the previously generated output when generating the probability for the next variable's truth value.

In a high-level overview, the functioning of the model can be described as follows (refer to fig. 2): for a given time t in the range [0, $n-1$], given a sequence of variables' representation $\hat{\mathbf{x}} = (\hat{x}_0, \hat{x}_1, \ldots, \hat{x}_{n-1})$, the embedding module processes the current variable's representation $\hat{x}_t$ along with the context c_t and a representation of the previous assignment $\hat{a}_{t-1}$ to yield a vector z_t. Subsequently, the sequential model transforms z_t into a state h_t taking into account information from previous time steps.

Finally, a Multi-Layer Perceptron (MLP) processes h_t to produce a 1-dimensional output $\hat{p}_{x_t}$. We then map $\hat{p}_{x_t}$ to the range [0,1] using the logistic function, interpreting p_{x_t} as the probability of the variable's value being true. A truth value a_t for x_t is sampled from p_{x_t} considering 0 and 1 as false and true, respectively.

We initialize $a_0 = 2$ as our start of sequence (sos) token, and $\hat{a}_t$ is the corresponding 3-dimensional One-Hot encoded vector of a_t. We introduce several adjustable components in the architecture and the training methodology.

This is done to pinpoint features that contribute to a better approximation of the solutions, such as the sequential model, baseline methods, and variables' representations. These components are elaborated further in the subsequent sections.

3.3.1 Variables Representations and Context

To underscore the importance of integrating problem instance information into the representation of variables when addressing the MaxSAT problem, we delve into two distinct methods: One-Hot encoding and Node2Vec [13] embeddings.

The former offers a direct, albeit potentially limited, approach that signifies

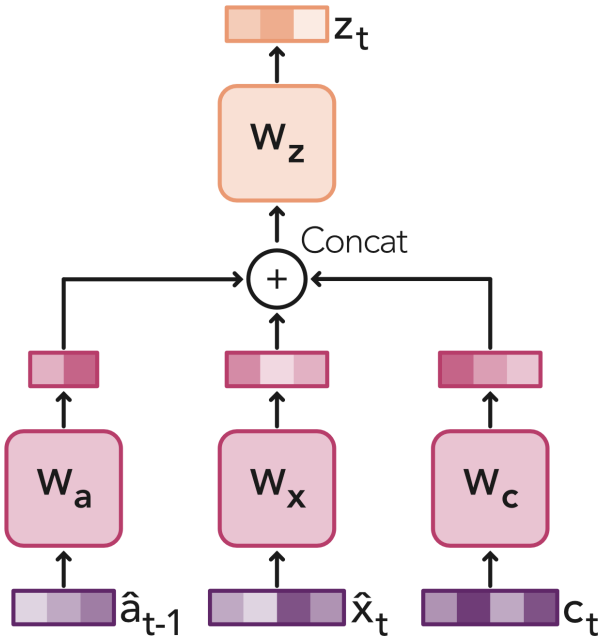

Fig. 4. Embedding module. The module intakes three inputs: the previous variable value representation $\hat{a}_{t-1}$, the current variable representation $\hat{x}_t$, and the context c_t. Each input is linearly transformed into its respective embedded vector with dimensions determined by configurable hyperparameters. These vectors are then concatenated to form a unified representation, encapsulating the essence of the initial inputs. This unified structure is further linearly transformed to generate the resultant vector z_t, defined by a designated hyperparameter dimension

the index of variables without encapsulating formula-specific information. On the other hand, Node2Vec is an algorithmic framework devised for learning continuous feature representations of nodes within networks.

It adopts a graph-based approach that seamlessly blends the strengths of structural equivalence (nodes sharing similar roles) with homophily (nodes associating with similar neighbors), thereby crafting versatile node embeddings. Building on this, our adaptation of the Node2Vec methodology entails crafting a graph representation of the formula, drawing inspiration from [25]. In this graph, nodes correspond to literals and clauses.

Undirected edges link literals and clauses that interrelate within the formula, and supplementary edges connect literals belonging to the same variable (see fig. 3). This comprehensive graph approach allows Node2Vec to encode formula-specific insights, potentially enhancing the optimization processes and solution quality.

While both methods provide unique perspectives, Node2Vec delves deeper into the instance's structure, though it necessitates a separate learning phase to acquire the feature representations of the nodes. For the Node2Vec variables' representation, we construct a vector for a specific variable by concatenating the Node2Vec embeddings of its corresponding two literals. This manner of representation captures insights concerning both literals of the variable.

Regarding the context, we average the embeddings corresponding to the literals, and similarly, average the embeddings linked to the clauses. The resultant context vector is the concatenation of these two averaged vectors, ensuring that the context includes comprehensive information about the instance, which could be invaluable when attempting to solve the problem.

3.3.2 Sequential Module

In our study, we compare and analyze three distinct models within the sequential module of the architecture: a Transformer encoder [26], an Long Short-Term Memory (LSTM) [14] network and Gated Recurrent Units (GRU) [8].

Both recurrent neural networks (RNNs) and the Transformer offer unique characteristics and capabilities for approximating solutions to the MaxSAT problem.

The Transformer encoder leverages self-attention mechanisms to capture dependencies among variables and effectively model long-range dependencies. In contrast, the LSTM and GRU models, also capture sequential information and dependencies over time but typically exhibit better memory efficiency when compared to Transformers. By examining and comparing these three models, we aim to assess their performance and determine the most

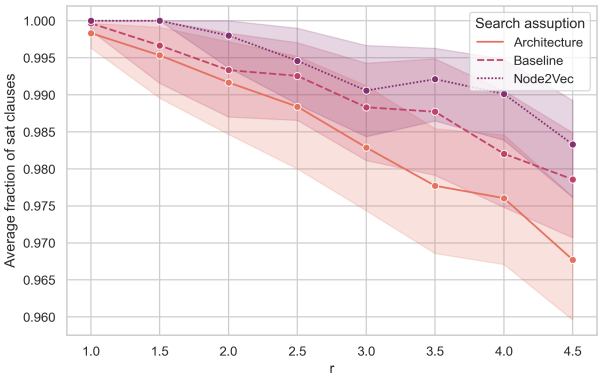

Fig. 5. Average fraction of satisfied clauses as the radius r increases. The average encompasses the six different values of n and their respective five different instances

effective approach for approximating MaxSAT solutions within our proposed architecture.

3.3.3 Embedding Module

To ensure robustness in our model and adaptability across different experiments, there was a need to address potential compatibility and expressiveness issues that arise when altering various components of the model. An example is the variability in representation sizes of the variables. For instance, while One-Hot encoding has an n-dimensional representation, the Node2Vec embedding's size is determined by a hyperparameter.

Additionally, unlike recursive networks that accommodate input vectors of any dimension, the Transformer architecture necessitates that the input dimension aligns with the model's dimension. To navigate these constraints and ensure a harmonious flow in the architecture, we introduced an embedding module.

This module was conceptualized to bridge the gap, linking the diverse inputs to the sequential module in a seamless and expressive manner. The core functionality of the embedding module is centered around transforming the three inputs, namely the previous variable value representation $\hat{a}_{t-1}$, the current variable representation $\hat{x}_t$, and the context c_t.

Each of these inputs undergoes a linear transformation to produce their respective embedded vectors, the dimensions of which are guided by configurable hyperparameters. This flexibility in defining dimensions allows the model to adapt effectively across a variety of experimental setups.

After these transformations, the three embedded vectors are concatenated to create a unified representation that captures the essence of all initial inputs while maintaining structural consistency.

This concatenated vector is subsequently linearly transformed to produce z_t, a resultant vector with a dimension defined by a specific hyperparameter. This vector serves as the input for the sequential module, ensuring it receives information in a consistent and streamlined format (see fig. 4).

3.3.4 Baselines

The policy gradient theorem can be generalized to include a baseline. Within our framework, a baseline $b(\phi)$ can be any function as long as it does not vary with the assignment π. It serves as a reference value (or baseline) against which the number of satisfied clauses from a particular assignment is compared.

Utilizing a baseline reduces the variance of the gradient estimate, which in turn stabilizes and accelerates the learning process. When incorporating a baseline $b(\phi)$, the equation (3) is adjusted reflecting the formulation of the REINFORCE [27] algorithm:

$$\nabla_\theta J(\theta|\phi) = \mathbf{E}\left[\delta \cdot \nabla_\theta \log p_\theta(\pi|\phi)\right], \qquad (5)$$

where the expectation is drawn from the distribution $p_\theta(\cdot|\phi)$ and $\delta = S(\pi|\phi) - b(\phi)$.

Acknowledging the importance of incorporating a suitable baseline, we experimented with three distinct baseline methods, in addition to the approach that operates without a baseline. Each of these methods aims to provide an estimate of the expected number of satisfied clauses:

– **Exponential Moving Average (EMA)**. This technique employs the exponential moving average to track the number of clauses satisfied by the model's assignment over time.

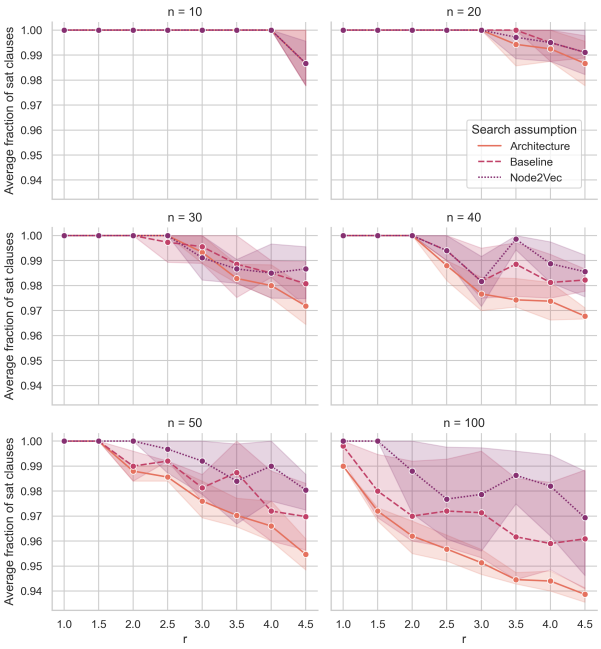

Fig. 6. Average fraction of satisfied clauses as the radius r increases for n =10, 20, 30, 40, 50, and 100. The average spans the five different instances for each n, and the plots are sequenced from left to right and top to bottom

– **Greedy Baseline**. With the current model parameters held constant, this baseline selects, at each time step, the most probable value for the current variable. It then calculates the number of clauses satisfied by the assignment.

– **Sampling Baseline**. In contrast to the greedy approach, this method draws B solutions and sets the baseline value as the average number of clauses satisfied by the drawn samples.

3.3.5 Exploration

To foster exploration within the model, and inspired by [4], we integrate two distinct strategies.

– **Logit Temperature**. At each time step t, the model's output is adjusted as:

$$\hat{p}_{x_t} = \frac{\hat{p}_{x_t}}{T}, \tag{6}$$

where T represents a temperature hyperparameter. During training, T is set to 1. However, during evaluation, when $T > 1$, the output $\hat{p}_{x_t}$ becomes less pronounced, thereby inhibiting the model from exuding overconfidence.

– **Logit Clipping**. At each time step t, the model's output undergoes another modification:

$$\hat{p}_{x_t} = C \cdot \tanh(\hat{p}_{x_t}), \tag{7}$$

where C serves as a hyperparameter, dictating the range of the logits and subsequently the entropy of the resultant output.

4 Experimental Setup

We conducted a series of experiments to assess the potential of a sequential model for approximating solutions to the MaxSAT problem. Our goal was to discern the architectural and procedural elements that enhance the optimization process to maximize the number of satisfied clauses.

Unlike the traditional SAT solvers, where the aim is to find a complete assignment that satisfies all clauses or determine its non-existence, our focus was on obtaining solutions that closely resemble optimality.

In our study, we carefully examined three pivotal elements: the decoder architecture, the baseline used to mitigate variance during optimization, and the embedding technique used to represent variables and incorporate semantic information. We posited that these components significantly influence the performance of the autoregressive model and, consequently, the quality of the MaxSAT solution.

To rigorously evaluate the contributions of these elements, we organized our experiments into three stages of hyperparameter searches. In the initial stage, the spotlight was on different decoder architectures, using a fixed zero baseline and rudimentary variables' representation to isolate the effect of the decoder structure on performance.

The subsequent stage introduced an extra layer of complexity by incorporating the various baselines into the hyperparameter search. Lastly,

ISSN 2007-9737

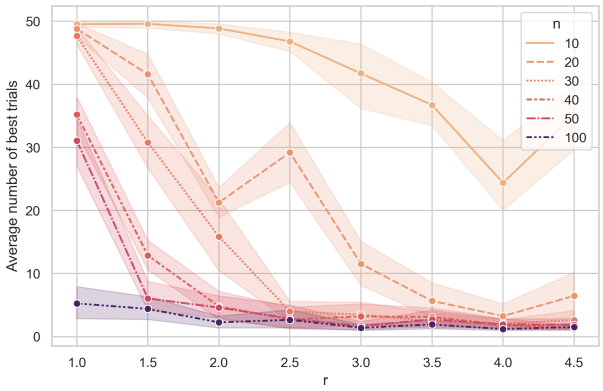

Fig. 7. Average number of best trials. The average is over the five different instances and the three different search assumptions

the Node2Vec embedding method was integrated into the final stage, broadening the search parameters and allowing for a comprehensive assessment of these elements' combined influence on model behavior.

Throughout these stages, we integrated several other hyperparameters into our searches, such as learning rate, number of layers, embedding dimensionality, logit clipping, logit temperature, and more. It is worth noting that crafting Node2Vec representations involves its own intricate procedure, detached from the parameter tweaking of the sequential model.

Consequently, we executed an independent hyperparameter search specifically for the Node2Vec embedding process, ensuring the availability of high-quality embeddings when required. By systematically varying these elements and exploring their interactions through the hyperparameter searches, our intention was to pinpoint the most efficient configurations for the autoregressive model in addressing the MaxSAT problem. This methodology enabled a examination of each component's individual contributions and their synergistic effects, steering the optimization process towards near-optimal solutions. Furthermore, to bolster the rigor and impartiality of our assessments, our experimental setup was tested across a diverse set of random 3-SAT instances.

4.1 Dataset

We generate a random k-SAT dataset comprising not necessarily satisfiable random formulas. For producing a random k-SAT instance with n variables and m clauses, we begin by choosing a small integer for k. Subsequently, for each clause c_i, where $i \in \{1, 2, \ldots, m\}$, we sample k variables uniformly at random without replacement. Each variable is then negated with a 50% independent probability.

The dataset encompasses 5 instances for each of the following configurations: for $n \in \{10, 20, 30, 40, 50, 100\}$, formulas with radius $r \in \{1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5\}$ and $k = 3$. Here, the radius signifies the ratio between the number of clauses m and the number of variables n.

4.2 Hyperparameters Searches

Our ablation study encompassed three distinct rounds of hyperparameter searches for each instance. The inaugural search assumption was concentrated on the exploration of various decoders—LSTM, GRU, and Transformer—along with adjusting associated hyperparameters. However, during this phase, we did not contemplate baselines beyond the zero value nor did we consider variables' representation techniques beyond One-Hot encoding.

To compare against the first round, the subsequent round introduced an additional degree of flexibility by integrating all the proposed baselines. This phase also presented the option of eschewing the utilization of any baseline, i.e., making the baseline equivalent to zero. Despite this change, the variables' embedding technique remained unaltered as One-Hot encoding.

In the third assumption, the exploration expanded to include Node2Vec embeddings, which are adept at capturing intricate relationships among variables and clauses. This provided a platform to ascertain the influence of variables' representation on the model's performance.

It's noteworthy that this investigation also encapsulated all previously delved into elements, including decoder architectures, baselines, and One-Hot encoding embeddings. As mentioned before, each of these rounds integrated additional

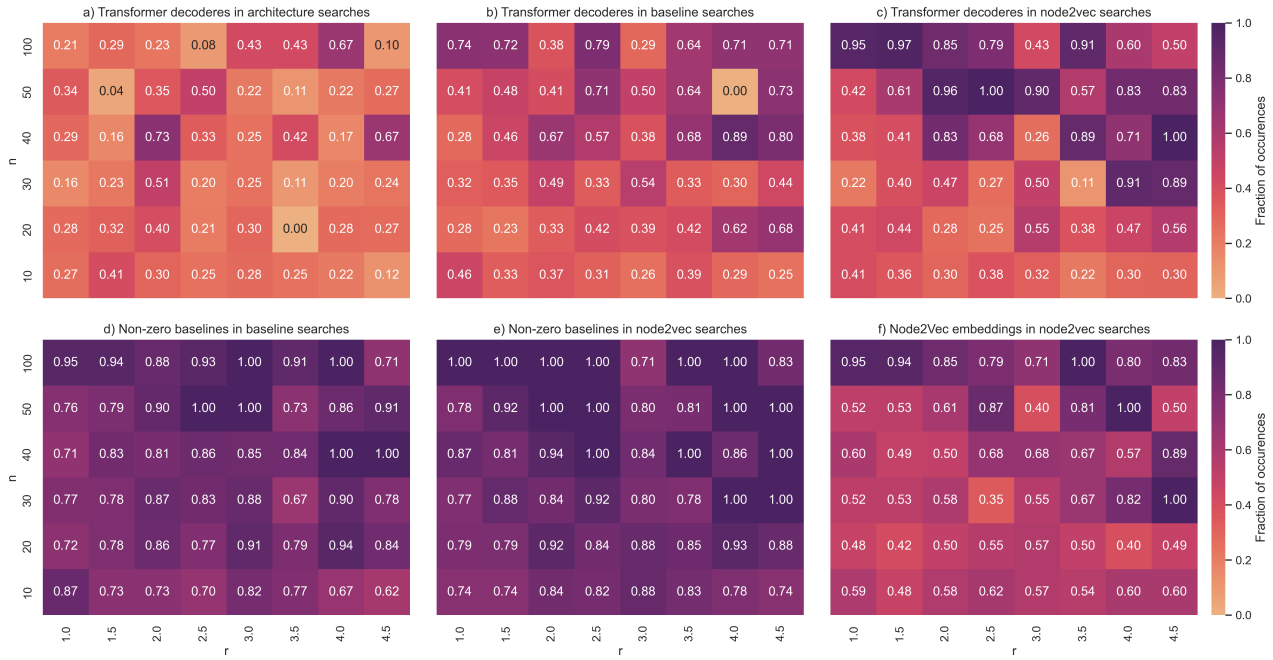

Fig. 8. Fraction of times Transformer decoders, non-zero baselines, and Node2Vec embeddings are present in the best trials for different values of n, r, and search assumptions. Sub-figures *a)*, *b)*, and *c)*, correspond to the fraction of times a Transformer decoder is present in the best trials when the search assumption was architecture, baseline, and Node2Vec, respectively. Sub-figures *d)* and *e)* show the fraction of times a non-zero baseline is present in the best trials when the search was baseline and Node2Vec, Sub-figure *f)* presents the fraction of times a Node2Vec embedding was present in the best trials in the Node2Vec search

hyperparameters into the searches, such as the learning rate, the number of layers, embedding size, logit clipping, logit temperature, among others.

For the systematic execution of each hyperparameter search, we leveraged the Tune [21] and Optuna [1] frameworks. The Tree-structured Parzen Estimator (TPE) [7] was employed as our hyperparameter search algorithm, complemented by the Asynchronous HyperBand Scheduler (ASHA) [20] to preemptively terminate unpromising trials.

The configuration for the ASHA scheduler was set with a grace period of $((2 \cdot n) + m) \times 4$ samples, and a cap at $((2 \cdot n) + m) \times 64$ samples. The TPE was designed to conduct 50 trials, with each batch constituted by 32 replicas of the identical instance.

The primary objective was to pinpoint the optimal configuration that maximized the satisfied clauses during the model's evaluation

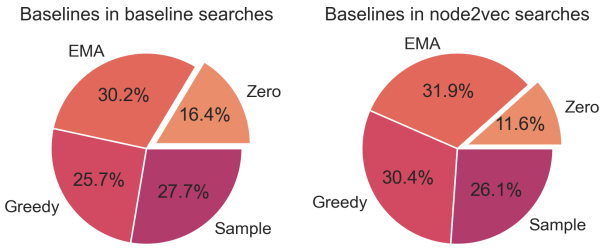

Fig. 9. Average fraction of times the different baselines are present in the best trials

phase. Pertaining to the hyperparameter search associated with the Node2Vec representations, we deployed the TPE for a total of 35 trials.

The ASHA scheduler in this context operated with a grace period of 5, peaking at 25 epochs. To guarantee a robust representation of the variables, the embedding size was set at 128.

4.3 Evaluation

To evaluate the performance of a specific model configuration on a given instance (i.e., a trial within a hyperparameter search), we adopted the following procedure: After processing every 320 samples, which comprise 10 batches with each batch having a size of 32 during the optimization process, we sampled 128 potential assignments (that is, we ran an episode on a single batch of size 128).

Subsequently, we computed the number of clauses satisfied (sat) by each of these assignments. The count of sat clauses at this point is determined by the highest number of sat clauses among the 128 assignments. The performance metric for the model, represented by the number of sat clauses achieved by a particular configuration, is the maximum number observed throughout the entire trial.

Furthermore, the number of sat clauses attained during a hyperparameter search on a given instance is derived from the maximum number of sat clauses observed across all 50 trials. Any trial that matches this maximum value is deemed as a best trial.

5 Results

After obtaining the number of sat clauses and identifying the best trials from the hyperparameter searches conducted for each combination of n, r, and search assumption (architecture, baseline, and Node2Vec embedding), we analyzed the impact of these search assumptions on the number of clauses the model satisfies.

Additionally, we delved into the architectures, baselines, and variables' representations involved in the top trials as both the number of variables and radius increased. This provided valuable insights into the configurations that yield superior solutions in terms of satisfied clauses.

5.1 Does the Search Assumption Impact the MaxSAT Solutions?

We used line graphs to illustrate the fraction of satisfied clauses achieved by each search assumption as the radius r increased. Moreover, a 99%-ile confidence interval was included to highlight the uncertainty in the results. Figure 5 shows the average fraction of sat clauses over different n values.

The graph reveals that, on average, integrating baselines into the search led to an increase in the number of satisfied clauses. This trend persisted across various r values. Additionally, incorporating Node2Vec representations into the search process yielded further enhancements in the number of satisfied clauses. This evidence suggests that the inclusion of baselines and Node2Vec embeddings can markedly improve the performance of the search algorithm.

To offer a more granular understanding of the influence of different search assumptions on the optimization process, we have included additional graphs for each n value. Figure 6 illustrates the line graphs for $n = 10$, 20, 30, 40, 50, and 100.

Examining these charts, we discern an interesting observation for smaller n values, such as 10, 20, and 30. For these cases, the benefit of integrating baselines or Node2Vec into the search space is less discernible.

For instance, with $n = 10$, all search methodologies produce similar outcomes, suggesting our optimization procedure is adept at identifying robust solutions irrespective of search assumptions.

When scrutinizing the outcomes for $n = 20$ and 30, we notice that the methods using Node2Vec do not consistently excel beyond other assumptions. For $n = 20$, the most optimal outcomes, on average, emerge when only baselines are integrated into the search space.

Likewise, for $n = 30$, no singular assumption consistently prevails across varying radii. This insinuates that, with smaller instances and an insufficient number of trials in the hyperparameters search, a more streamlined search space might occasionally offer superior outcomes. Yet, we posit that augmenting the number of trials in the

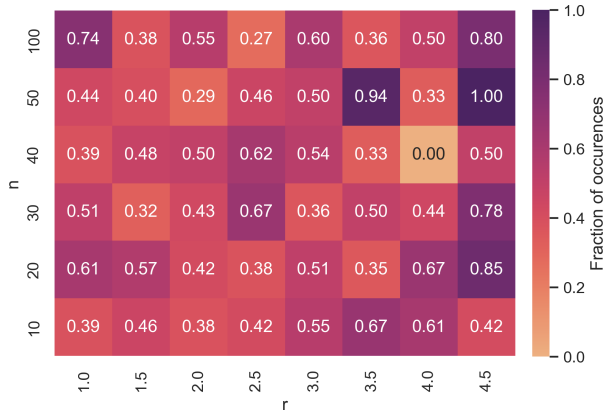

Fig. 10. Fraction of times the context was used in the best trials for the different values of n and r in the Node2Vec searches. Results are normalized by the fraction of times the Node2Vec representations were chosen

hyperparameters search could bolster results for both the baseline and Node2Vec assumptions.

Conversely, for greater n values, namely 40, 50, and 100, the merit of embedding baselines in the search space becomes evident. These visuals clearly depict an augmented fraction of satisfied clauses when baselines are incorporated, attesting to the efficacy of this approach for more extensive problem instances.

Moreover, the integration of Node2Vec embeddings into the search space results in general in even more pronounced improvements in satisfied clauses, emphasizing the advantages of incorporating instance-specific information into the optimization process. In light of the aforementioned analyses and observations, it becomes evident that search assumptions significantly influence MaxSAT solutions. These revelations accentuate the importance of employing baselines and leveraging Node2Vec embeddings, particularly when addressing problems of larger scales.

5.2 Which Configuration is Better?

To determine the most effective configurations, we counted the number of best trials as both n and r increased. Figure 7 demonstrates that for simpler instances—those with smaller n and r values—many trials achieve the sat number of clauses (i.e., the maximum number of satisfied clauses achieved by the model during a specific search).

This indicates consistent model performance even with randomized hyperparameters during the hyperparameters searches' warm-up phase. However, for more complex instances (larger n or r values), reaching the maximum number of satisfied clauses is challenging, and only specific configurations yield the best results.

Figure 8 provides insights into how frequently Transformer decoders, non-zero baselines, and Node2Vec representations appear in the best trials across the values of n, r, and search assumptions. For different search assumptions, sub-figures 8(a-c) show the prevalence of Transformer decoders.

With increasing n and r, the Transformer decoder becomes more favored over LSTM and GRU architectures, especially when the search includes baselines or variables' representation based on Node2Vec. Sub-figures 8(d-e) reveal a consistent preference for non-zero baselines across varying n and r values when available in the search space.

This suggests their pivotal role in optimizing MaxSAT solutions. Sub-figure 8(f) illustrates the preference for Node2Vec representations during Node2Vec searches. When available, these embeddings are often chosen, signaling the importance of instance-specific information, particularly for more intricate instances.

While simpler instances often exhibit a range of top-performing configurations, as revealed in fig. 7, complex ones with larger n or r values are more reliant on specific elements like Transformer decoders, non-zero baselines, and Node2Vec embeddings. This underscores the significance of these elements for handling intricate instances. Figure 9 offers a breakdown of how often different baselines feature in top trials.

It becomes clear that non-zero baselines play a substantial role in achieving optimal solutions. The pie chart highlights the dominance of the EMA approach in both searches, with greedy and sample baselines also being prominent. These data emphasize the merit of considering diverse baseline strategies in optimization processes.

During our experiments, models had the option to either utilize or ignore the context when Node2Vec variables were selected. Figure 10 portrays how often context was incorporated in top trials, normalized for instances when Node2Vec-type variables were selected. While its adoption slightly increases with n and r, context is not consistently present in top trials. We theorize that refining the context design might bolster the information from Node2Vec variables, potentially enhancing the solutions.

6 Conclusion and Future Work

We conducted a series of experiments across a diverse set of random 3-SAT instances to assess the potential of a sequential model for approximating solutions to the MaxSAT problem optimizing the parameters of the parametric model by maximizing the expected value of the number of satisfied clauses to predict a distribution over the possible solutions using the policy gradient method.

Our goal was to discern the architectural and procedural elements that enhance the optimization process to maximize the number of satisfied clauses.

The experiments presented demonstrated the potential advantages of using certain configurations and search assumptions, such as the inclusion of baselines and the Node2Vec embeddings, in improving the efficiency and accuracy of the model for MaxSAT solutions.

The clear influence of Transformer decoders, particularly for more complex instances, emphasized the utility of architectures that can capture long-range dependencies in the problem. This is pivotal for tasks such as MaxSAT where inter-variable relationships can have a significant impact on the overall satisfaction of clauses.

The introduction and evident advantage of non-zero baselines, especially the EMA approach, signaled the importance of stable and guided optimization. In the same way, Node2Vec embeddings benefit the model, especially as n and r increase. However, one surprising outcome was the inconsistent utilization of context with Node2Vec variables in top trials.

We postulate that an improved context design, which is a promising avenue for improvement, could supplement rich semantic information, potentially leading to enhanced solutions. Moving forward, several research directions emerge from the insights and limitations of our current approach:

- **Advanced Variables Representations**. Instead of relying solely on Node2Vec, we propose exploring more sophisticated graph embedding techniques, especially Graph Neural Networks, to better capture the structural intricacies of the MaxSAT instance.
- **Refining Context Usage**. The ambiguous outcomes from our usage of context with Node2Vec variables highlight the need for further investigation. A more refined integration approach, such as a time-dependent context updated at each time step using attention mechanisms, is worthy of exploration.

  Coupling this with alternative graph embedding methods may offer richer structural insights into the SAT formula.
- **Alternative Architectural Decoder Considerations**. The Transformer model's efficacy, especially for complex instances, leads us to believe there's potential in transformer-tailored architectures for combinatorial optimization tasks.

  For instance, a specialized decoder design that isn't reliant on consuming all previous inputs directly, and instead uses context more effectively, might yield better results.
- **Expanded Dataset Variability**. While our current experiments leaned on random 3-SAT instances, there's merit in diversifying our test cases. This includes exploring other MaxSAT problem types and integrating real-world instances to gain a comprehensive understanding of the model's capabilities.
- **Enhanced Exploration Techniques**. The logit temperature and clipping methods served us well in this study. Yet, integrating other exploration or regularization techniques, like the entropy bonus in computing the policy

gradient loss, might elevate the quality of the models outputs.

– **Improve Post-Processing**. Post-model solution refinement is a promising direction. Leveraging larger samplings, advanced techniques like beam search, or tailored search strategies could maximize solution quality further.

– **Contemplation of Variable Ordering**. Throughout our experiments, the variables were introduced to the model in ascending order based on their index. Investigating other types of variable ordering, especially those based on advanced strategies, may prove advantageous in the search for optimal assignments.

Acknowledgments

Authors would like to acknowledge the support provided by the Instituto Politécnico Nacional under projects: 20200651, 20210316, 20220002, 20230232, 20220798 and 20211096 to carry out this research. O. Gutiérrez thanks CONAHCYT for the scholarship granted towards pursuing his graduate studies.

References

1. **Akiba, T., Sano, S., Yanase, T., Ohta, T., Koyama, M. (2019).** Optuna: A next-generation hyperparameter optimization framework. Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 2623–2631. DOI: 10.1145/3292500.3330701.
2. **Avellaneda, F. (2020).** A short description of the solver EvalMaxSAT. MaxSAT Evaluation, Vol. 8.
3. **Bacchus, F. (2022).** MaxHS in the 2022 MaxSAT evaluation. MaxSAT Evaluation 2022, Vol. B-2022, pp. 17–18.
4. **Bello, I., Pham, H., Le, Q. V., Norouzi, M., Bengio, S. (2016).** Neural combinatorial optimization with reinforcement learning. 5th International Conference on Learning Representations. DOI: 10.48550/arXiv.1611.09940.
5. **Bengio, Y., Lodi, A., Prouvost, A. (2021).** Machine learning for combinatorial optimization: A methodological tour d'horizon. European Journal of Operational Research, Vol. 290, No. 2, pp. 405–421. DOI: 10.1016/j.ejor.2020.07.063.
6. **Berg, J., Demirović, E., Stuckey, P. J. (2019).** Core-boosted linear search for incomplete MaxSAT. In: Rousseau, L. M., Stergiou, K. (eds), Integration of Constraint Programming, Artificial Intelligence, and Operations Research. CPAIOR 2019. Lecture Notes in Computer Science, Vol. 11494, pp. 39–56. DOI: 10.1007/978-3-030-19212-9_3.
7. **Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B. (2011).** Algorithms for hyper-parameter optimization. Advances in Neural Information Processing Systems, Vol. 24.
8. **Cho, K., van-Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y. (2014).** Learning phrase representations using RNN encoder-decoder for statistical machine translation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). DOI: 10.48550/arXiv.1406.1078.
9. **Davies, J. (2013).** Solving MaxSAT by decoupling optimization and satisfaction. Ph.D. thesis, University of Toronto.
10. **Demirovic, E., Musliu, N. (2014).** Modeling high school timetabling as partial weighted maxSAT. LaSh 2014: The 4th Workshop on Logic and Search, pp. 1–39.
11. **Demirovic, E., Musliu, N., Winter, F. (2019).** Modeling and solving staff scheduling with partial weighted maxSAT. Annals of Operations Research, Vol. 275, pp. 79–99. DOI: 10.1007/s10479-017-2693-y.
12. **Graves, A. (2013).** Generating sequences with recurrent neural networks. arXiv preprint arXiv:1308.0850. DOI: 10.48550/arXiv.1308.0850.

13. **Grover, A., Leskovec, J. (2016).** node2vec: Scalable feature learning for networks. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge discovery and data mining, pp. 855–864. DOI: 10.1145/2939672.2939754.
14. **Hochreiter, S., Schmidhuber, J. (1997).** Long short-term memory. Neural computation, Vol. 9, No. 8, pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735.
15. **Hottung, A., Kwon, Y. D., Tierney, K. (2021).** Efficient active search for combinatorial optimization problems. 2022 The International Conference on Learning Representations. DOI: 10.48550/arXiv.2106.05126.
16. **Juma, F., Hsu, E. I., McIlraith, S. A. (2012).** Preference-based planning via MaxSAT. Advances in Artificial Intelligence: 25th Canadian Conference on Artificial Intelligence, Vol. 7310, pp. 109–120. DOI: 10.1007/978-3-642-30353-1_10.
17. **Khalil, E., Dai, H., Zhang, Y., Dilkina, B., Song, L. (2017).** Learning combinatorial optimization algorithms over graphs. Advances in neural information processing systems, Vol. 30.
18. **Kingma, D. P., Ba, J. (2014).** Adam: A method for stochastic optimization. International Conference on Learning Representations. DOI: 10.48550/arXiv.1412.6980.
19. **Kool, W., van-Hoof, H., Welling, M. (2018).** Attention, learn to solve routing problems! 2019 International Conference on Learning Representations. DOI: 10.48550/arXiv.1803.08475.
20. **Li, L., Jamieson, K., Rostamizadeh, A., Gonina, K., Hardt, M., Recht, B., Talwalkar, A. (2018).** Massively parallel hyperparameter tuning. ICLR 2018 Conference Acceptance Decision.
21. **Liaw, R., Liang, E., Nishihara, R., Moritz, P., Gonzalez, J. E., Stoica, I. (2018).** Tune: A research platform for distributed model selection and training. arXiv. DOI: 10.48550/arXiv.1807.05118.
22. **Martins, R., Manquinho, V., Lynce, I. (2014).** Open-WBO: A modular MaxSAT solver. Theory and Applications of Satisfiability Testing–SAT 2014: 17th International Conference, Held as Part of the Vienna Summer of Logic, VSL 2014, pp. 438–445. DOI: 10.1007/978-3-319-09284-3_33.
23. **Martins, R., Manthey, N., Terra-Neves, M., Manquinho, V., Lynce, I. (2023).** Open-WBO @ MaxSAT evaluation 2023. MaxSAT Evaluation 2023, pp. 18–19.
24. **Safarpour, S., Mangassarian, H., Veneris, A., Liffiton, M. H., Sakallah, K. A. (2007).** Improved design debugging using maximum satisfiability. Formal Methods in Computer Aided Design, pp. 13–19. DOI: 10.1109/FAMCAD.2007.26.
25. **Selsam, D., Lamm, M., Bünz, B., Liang, P., de-Moura, L., Dill, D. L. (2018).** Learning a SAT solver from single-bit supervision. arXiv. DOI: 10.48550/arXiv.1802.03685.
26. **Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., Polosukhin, I. (2017).** Attention is all you need. Advances in Neural Information Processing Systems, Vol. 30.
27. **Williams, R. J. (1992).** Simple statistical gradient-following algorithms for connectionist reinforcement learning. Machine learning, Vol. 8, pp. 229–256. DOI: 10.1007/BF00992696.

Article received on 26/10/2023; accepted on 24/11/2023.
* *Corresponding author is Ricardo Menchaca-Méndez.*