mathematics

MDPI

Article

MTS-PRO2SAT: Hybrid Mutation Tabu Search Algorithm in Optimizing Probabilistic 2 Satisfiability in Discrete Hopfield Neural Network

Ju Chen [1,2], Yuan Gao [2,3], Mohd Shareduwan Mohd Kasihmuddin [2,*], Chengfeng Zheng [2], Nurul Atiqah Romli [2], Mohd. Asyraf Mansor [4], Nur Ezlin Zamri [4] and Chuanbiao When [3,*]

1 School of Basic Medical Sciences, Chengdu University of Traditional Chinese Medicine, Chengdu 610000, China; chenju@cdutcm.edu.cn
2 School of Mathematical Sciences, Universiti Sains Malaysia, Penang 11800, Malaysia; gaoyuan@student.usm.my (Y.G.); chengfengzheng@student.usm.my (C.Z.); nurulatiqah_@student.usm.my (N.A.R.)
3 School of Medical Information Engineering, Chengdu University of Traditional Chinese Medicine, Chengdu 610000, China
4 School of Distance Education, Universiti Sains Malaysia, Penang 11800, Malaysia; asyrafman@usm.my (M.A.M.); ezlinzamri@student.usm.my (N.E.Z.)
* Correspondence: shareduwan@usm.my (M.S.M.K.); wcb@cdutcm.edu.cn (C.W.); Tel.: +60-4653-4769 (M.S.M.K.); +86-028-6180-0166 (C.W.)

Abstract: The primary objective of introducing metaheuristic algorithms into traditional systematic logic is to minimize the cost function. However, there is a lack of research on the impact of introducing metaheuristic algorithms on the cost function under different proportions of positive literals. In order to fill in this gap and improve the efficiency of the metaheuristic algorithm in systematic logic, we proposed a metaheuristic algorithm based on mutation tabu search and embedded it in probabilistic satisfiability logic in discrete Hopfield neural networks. Based on the traditional tabu search algorithm, the mutation operators of the genetic algorithm were combined to improve its global search ability during the learning phase and ensure that the cost function of the systematic logic converged to zero at different proportions of positive literals. Additionally, further optimization was carried out in the retrieval phase to enhance the diversity of solutions. Compared with nine other metaheuristic algorithms and exhaustive search algorithms, the proposed algorithm was superior to other algorithms in terms of time complexity and global convergence, and showed higher efficiency in the search solutions at the binary search space, consolidated the efficiency of systematic logic in the learning phase, and significantly improved the diversity of the global solution in the retrieval phase of systematic logic.

Keywords: artificial neural networks; mutation tabu search; satisfiability logic; metaheuristics

MSC: 68T07; 68T27; 68T20

Citation: Chen, J.; Gao, Y.; Kasihmuddin, M.S.M.; Zheng, C.; Romli, N.A.; Mansor, M.A.; Zamri, N.E.; When, C. MTS-PRO2SAT: Hybrid Mutation Tabu Search Algorithm in Optimizing Probabilistic 2 Satisfiability in Discrete Hopfield Neural Network. *Mathematics* **2024**, *12*, 721. https://doi.org/10.3390/math12050721

Academic Editors: Marko Đurasević and Domagoj Jakobović

Received: 12 January 2024
Revised: 17 February 2024
Accepted: 25 February 2024
Published: 29 February 2024

1. Introduction

In recent years, the rapid development of artificial intelligence has made artificial neural networks a focal point of interest within the academic community [1,2]. Artificial neural networks (ANNs) are an application of learning intelligence from biological brains and have gradually become the frontier in the field of artificial intelligence. In the early 1980s, J.J. Hopfield proposed the Hopfield neural network and made a first attempt to introduce the concept of "energy function" to analyze the stability of dynamic neural networks, and obtained the conditions for determining the stability of the system, which played a crucial role in the research and development of neural networks. Hopfield networks allow machines to store "memories". The network is similar to the way the

human brain processes information, and it is able to "recall" its complete information through incomplete information. Hopfield neural networks have been applied to many other disciplines, such as image processing [3,4], combinatorial optimization [5], and Boolean function optimization [6]. Discrete Hopfield neural networks (DHNNs) are a type of fully connected neural network, which means that each neuron is connected to every other neuron in the network, and the strength of the neuron connections is represented by synaptic weights. These synaptic weights are part of the content-addressable memory (CAM) in DHNNs. As the number of neurons increases, the DHNN suffers from more capacity issues with CAM. Therefore, an optimal symbolic instruction is required to ensure the appropriate connectivity of neurons without compromising the networks' behavior.

Satisfiability (SAT) was designed as a symbolic instruction for the neuronal connections of DHNNs. The logic rules were firstly introduced to DHNNs by Wan Abdullah [7]. In this paper, an artificial neural network (ANN) was formulated by leveraging the correlation between logical variables and the minimized final energy function. Through a comparative analysis between the cost function derived from logical rules and the final energy function, one can determine the synaptic weights that facilitate the modulation of the final neuron state's direction. The emergence of this study laid the key foundation for the further rapid development of systematic and nonsystematic logics of DHNNs. Typical Systematic Logical rules are available in 2 Satisfiability (2SAT), which requires each clause to contain only a fixed quantity of literals. Chen et al. [8] proposed Probabilistic 2 Satisfiability (PRO2SAT), in which probabilities were introduced to control the proportion of positive literals and the generation position, aiming to improve the satisfiability of systematic logical clauses. At present, PRO2SAT has successfully embedded the probability-based systematic logic into DHNNs. However, the learning efficacy of the PRO2SAT model remains unsatisfactory when addressing a large number of neurons. Additionally, research findings from reference [7,8] indicate that, as the number of neurons increases, the retrieval capability of systemic logic during the learning phase gradually approaches zero. Hence, there is a necessity to explore new methods for enhancing the learning efficiency of systemic logic.

Fueled by the effectiveness of the learning phase in showing direct contribution to that of the retrieval phase, the optimization of the learning phase plays a crucial part in DHNNs. To solve this problem, metaheuristics are embedded into systematic and non-systematic logic rules to minimize the valence function and obtain the optimal salient weights, thus enhancing the ability to obtain the uppermost final state of neurons. Metaheuristic algorithm refers to a method for the optimal or satisfactory solutions to complex optimization problems based on an intelligent computation mechanism [9,10], and it is also sometimes referred to as intelligent optimization algorithm, which is admired by more researchers for its accessibility to implement and provide feasible solutions to problems at a reasonable cost (computation time and space). Metaheuristic algorithms typically do not depend on the specific structure of the problem but adapt to the characteristics of various problems through a flexible iterative search process. Currently, more metaheuristic algorithms have been applied to systematic and non-systematic logics of DHNNs. As proposed by [11], the GA algorithm was imbedded in the logic phase and learning phase of DHNNs. During the logic phase, the GA algorithm assists in constructing the logical rules of r2SAT according to the desired ratio of negative literals. In the learning phase, the GA algorithm aids r2SAT in finding satisfied assignments of the proposed logic. According to [12], the EA algorithm was also embedded in the random maximum 2 satisfiability logic for minimizing the cost function. As defined in [13], the artificial bee colony algorithm (ABC) was used in the logic stage of r2SAT to control the distribution of negative literals in the logical structure and find the correct logical structure based on the initialization proportion of negative literals. The ant colony algorithm (ACO) was introduced in [14] for minimizing the cost function corresponding to the logic rules in DHNNs and finding the optimal path to the zero cost function. The gray wolf algorithm (GWO) was used in [15] in a rotor Hopfield neural network (RDHNN). Using the rotor Hopfield neural network and grey wolf optimization algorithm to identify solid oxide fuel cell models, the results show that this hybrid model

has advantages in terms of accuracy and computational efficiency. Overall, these studies demonstrated the auxiliary optimization function of metaheuristic algorithms in DHNNs, and they played a primary role in the logic phase and learning phase of DHNNs. Although there is a plethora of metaheuristic algorithms available, their application in both systematic and non-systematic logic is relatively limited. In order to obtain suitable metaheuristic algorithms in terms of solution quality and solution efficiency, more extensive explorations are needed to apply the metaheuristic algorithms to DHNNs. Therefore, a comprehensive survey of metaheuristic algorithms applied in DHNNs is necessary to identify the most suitable algorithms.

Tabu search (TS) is a metaheuristic algorithm starting from a single solution [16], which has been successfully applied to solve many combinatorial optimization problems [17]. TS conducts similar search activities as a human performs which are based on a global stepwise optimization algorithm for local domain search, and creates a stored list (tabu table) to record the search process. The data accessed in the tabu list are not accessed again in the next few iterations. In this way, repetitive search is avoided by the TS algorithm, which directs its search toward the local optimal solution. The selection of parameters and the design of neighborhood operation plays crucial roles in the TS algorithm [18]. An aspiration criterion will be used in TS during its use to release some excellent tabu solutions and avoid the loss of optimal solutions. The TS algorithm has the advantages of fast convergence and exemption from local minimal values. Since the TS algorithm was proposed, substantial optimization has been conducted for it by the researchers. Meeran et al. [19] proposed a hybrid TS algorithm which combines the TS algorithm with the evolutionary algorithm and the ant colony optimization (ACO) algorithm to solve the travelling salesman problem (TSP), providing the method and idea of the TS algorithm for the searching solution in binary search space. Žulj et al. [20] proposed a hybrid algorithm based on adaptive large neighborhood search or tabu search (ALNS/TS) to solve the order batching problem (OBP). Initially, multiple destruction and repair operators are employed for the solution search. As iterations proceed, the probabilities of selecting these operators are adaptively adjusted based on their performance in the search process to optimize the search. After a certain number of iterations, the TS algorithm is executed to explore the neighborhood of the current solution to find a better solution. In ALNS/MTS, the diversification capability of ALN was combined with the intensification capability of TS. Compared with state-of-the-art algorithms, ALNS/TS showed the most obvious advantages over larger instances. Lin et al. [21] proposed the hybrid binary particle swarm algorithm (HBPSO/TS) based on TS for solving the set-union knapsack problem (SUKP). A tabu-based mutation process was designed as part of this algorithm to guide the search into the new promising regions, and the MTS algorithm was employed to enhance the capabilities of local exploitation. In addition, a gain update strategy was used to reduce the solution time. Experiments have shown that HBPSO/TS performs much better than other algorithms in terms of solution quality. These two studies provided useful ideas for improving the quality and efficiency of the MTS algorithm in solving problems. Although these hybrid metaheuristic algorithms show potential in improving search efficiency and tackling complex problems, they typically involve combinations of multiple operators or operations, thereby resulting in higher algorithmic complexity and more pronounced challenges in parameter tuning.

In this paper, we introduced MTS to the latest systematic logic PRO2SAT [8] to extensively explore the application of metaheuristic arithmetic in systematic logic. The contributions of this paper are as follows:

(a) To propose a modified metaheuristic algorithm, namely the hybrid mutation tabu search algorithm, which integrates a mutation operator and segment operation as neighborhood operations. Through these newly constructed neighborhood operations, the tabu search algorithm has been successfully incorporated into the systematic logic for minimizing the cost function.

(b) To optimize the retrieval phase by introducing a mutation operator. The mutation operator randomly flips the literals in unsatisfied clauses, aiming to disrupt the bias in the final state of neurons and enhance the diversity of the global minimum.
(c) To propose the average similarity metric to evaluate the variability and diversity of solutions in the retrieval phase by calculating the similarity between non-repeated solutions and benchmark solutions.
(d) This study examined the effectiveness of the mutation tabu search algorithm through the analysis of different performance measures. The performance of the proposed approach was thoroughly evaluated against that of the leading logical rule in both the learning and retrieval phase.

The organization of this paper is as follows. Section 2 highlights the motivation of this study. Section 3 presents the general formula of PRO2SAT. Section 4 describes the logical structure of PRO2SAT in DHNNs, referred to as PRO2SAT. Section 5 provides the objective function of the metaheuristic algorithm in the learning phase. Section 6 details the metaheuristic algorithms used in the proposed model for the optimization process in the learning phase. Section 7 introduces the experimental setup and parameter settings as well as the performance evaluation indicators used in this paper. Section 8 discusses and analyzes the experimental results of the proposed model and the comparison model. It should be noted that the organization of this experiment is based on the listed contributions. This paper is not intended to replace the existing logical structure of neuron representation in DHNNs, but it will provide critical explorations on the different metaheuristics in the proposed logical phase with the improved DHNN model, which may contribute to the quality of the final neuron states retrieved.

2. Motivation

2.1. Inefficient Learning Phase of DHNNs

For DHNNs, effective learning phases can lead the network to optimal synaptic weight management. A satisfactory explanation of SAT logical rules ensures a zero-cost function for DHNNs, which, when compared with the Lyapunov energy function, yields the optimal synaptic weight. As the scale of neurons in the learning phase expands, most existing systematic and non-systematic logical rules employ metaheuristic algorithms as optimization training algorithms, iteratively finding consistent interpretation. The existing work of Zamri et al. [11] initially employs a genetic algorithm (GA) as the logic phase and learning phase algorithm for RAN2SAT in the DHNN model, known as DHNN-r2SAT. This model is compared with traditional exhaustive search (ES) and demonstrates superior retrieval performance. Unfortunately, the genetic algorithm mutation operator proposed in this paper easily falls into local optima when the number of neurons is large, meaning that the fitness of the neuron state before mutation may be better than after mutation. The study by Someetheram et al. [12] introduces a novel non-satisfiability logical rule for the first time and successfully embeds the election algorithm within it. This approach enhances the capability of minimizing the cost function through the election algorithm to seek consistent interpretations. The research findings demonstrate the significant superiority of the election algorithm in the training process of the discrete Hopfield neural network compared to genetic algorithms and exhaustive searches. Currently, existing optimization training algorithms typically rely on probability to explore the consistency solutions in the learning phase of both system and non-system logics, which indeed plays a role in improving learning efficiency. However, this probabilistic dependency can easily lead to being influenced by randomness and trapped in local optima, thus requiring more iterations and computational time. Therefore, from an innovative perspective, introducing a non-probabilistic approach to explore new solution methods is of significant importance. With this concept in mind, we propose a tabu search algorithm integrated with a mutation operator, which utilizes non-probabilistic operations such as a tabu list and aspiration rules to guide the search process, aiming to enhance the efficiency and robustness of learning for PRO2SAT.

2.2. Limited Solution Diversity in Retrieval Phase of DHNN

The optimal retrieval capability of the DHNN ensures that the final neuron states always converge to the global minimum energy. Currently, most systematic or non-systematic logics follow the retrieval phase training mechanism proposed by Abdullah [7], known as the Wan Abdullah (WA) method. For example, the PRO2SAT by Chen et al. [8], the 2SAT by Zamri et al. [11], and the RANkSAT by Someetheram et al. [12] all adopt this approach. These models update neuron states using the hyperbolic tangent activation function (HTAF) after local field computation. The combination of local field and HTAF results in good stability during the retrieval phase. However, unfortunately, the presence of positive literals in second or third-order clauses leads to bias in the final neuron states obtained during the retrieval phase, meaning that the solutions generated during the retrieval phase will be excessively similar. This bias becomes more pronounced when there are more positive literal clauses. To address this issue, this paper introduces a mutation strategy to improve the diversity of neurons after local field computation, thereby enhancing solution diversity.

3. Probabilistic 2 Satisfiability (PRO2SAT)

PRO2SAT belongs to systematic logic, which consists of variables, literals, probabilities, AND operators (conjunction, also denoted by), OR operators and non-operators. Compared with the traditional systematic logic 2SAT, PRO2SAT has a fixed number of positive literals, where the probability that is used to control the uniform distribution of positive literals in the clauses is introduced. Equation (1) shows the general formula of PRO2SAT, where C_i is a clause, n is the number of second-order clauses, and the composition of clauses is represented by Equation (2).

$$P_{PRO2SAT} = \wedge_{i=1}^{n} C_i \tag{1}$$

$$C_i = \vee_{l=2i-1}^{2i} p_l x_l \tag{2}$$

where x_l is a set of literals. The literal is either positive literal x_l or negative literal $\neg x_l$. The conjunction formula of the two literals is the clause. The literals are bipolar, i.e., they can take value $\{-1, 1\}$, which represents $\{FALSE, TRUE\}$, respectively. p_l represents the probability, which determines the positive and negative literals. For example, when $p_3 = 0.8$, the probability of $x_3 \in \{x_3\}$ is 0.8, whereas $x_3 \in \{\neg x_3\}$ has a probability of 0.2. The calculation p_l is shown in Equations (3)–(5):

$$p_i = \begin{cases} \xi, & if(\eta_{i-1} \geq \eta) \; or \; (i \mathrm{mod} 2 = 0 \; and \; x_{i-1}) \\ \max(\eta, 1-\eta), & otherwise \end{cases} \tag{3}$$

$$\eta_i = \frac{1}{i} \sum_{j=1}^{i} N_j \tag{4}$$

$$N_j = \begin{cases} 1, & x_j \\ 0, & \neg x_j \end{cases} \tag{5}$$

where l, i, and j are integers greater than zero. Variable N_j in Equation (5) is used to mark the positive literals, and variable η_i in Equation (4) is used to record the number of positive literals in the first i variables. Variable η in Equation (3) is the preset proportion of positive literals in PRO2SAT, and the value range is [0.1, 0.9]. It is worth mentioning that the step size of variable n is set to 0.1, ensuring that the number of positive literals is not duplicated. When the step size is less than 0.1 ($\Delta\eta \leq 0.1$), the number of positive literals ($N\eta$) will have a significant number of replicates, which will cause redundancy in the model's classification. When the step size is more than 0.1 ($\Delta\eta \geq 0.1$), fewer values will be assigned to the number of positive literals to reduce the number of classifications of the model. ξ is used to avoid too much negative literals and two positive literals in one clause, i.e., avoiding $x_i \vee x_{i+1}$. In addition, when p_l is assigned as $\max(\eta, 1-\eta)$, it means that the number of positive literals does not reach the preset value, and it is necessary to continue generating the positive literals. When the neuron is 8, the example of $P_{PRO2SAT}$ is shown in Table 1.

Table 1. Examples of PRO2SAT.

N, η	$N\eta$	$P_{PRO2SAT}$
8, 0.1	1	$(\neg x_1 \vee x_2) \wedge (\neg x_3 \vee \neg x_4) \wedge (\neg x_5 \vee \neg x_6) \wedge (\neg x_7 \vee \neg x_8)$
8, 0.3	2	$(x_1 \vee \neg x_2) \wedge (\neg x_3 \vee \neg x_4) \wedge (\neg x_5 \vee x_6) \wedge (\neg x_7 \vee \neg x_8)$
8, 0.5	4	$(x_1 \vee \neg x_2) \wedge (\neg x_3 \vee x_4) \wedge (\neg x_5 \vee x_6) \wedge (x_7 \vee \neg x_8)$
8, 0.7	6	$(x_1 \vee \neg x_2) \wedge (x_3 \vee x_4) \wedge (\neg x_5 \vee x_6) \wedge (x_7 \vee x_8)$
8, 0.9	7	$(x_1 \vee x_2) \wedge (\neg x_3 \vee x_4) \wedge (x_5 \vee x_6) \wedge (x_7 \vee x_8)$

4. $P_{PRO2SAT}$ in Discrete Hopfield Neural Network (DHNN)

A discrete Hopfield neural network (DHNN) is a single-layer symmetric full feedback network where each neuron has the same function and is interconnected with each other. The DHNN employs an asynchronous approach for adjusting network states, and it utilizes a symmetric matrix for the synaptic weight matrix. In this section, $P_{PRO2SAT}$ will be embedded in the DHNN, and the newly generated neural network is called PRO2SAT. The asynchronous update method for the neurons of PRO2SAT is shown in Equation (6):

$$S_i = \begin{cases} 1, \ if \sum\limits_j W_{ij} S_j \geq \theta \\ -1, \ otherwise \end{cases} \tag{6}$$

where S_i and S_j are two interconnected neurons, denoting the states of the i-th and j-th neurons, respectively. W_{ij} signifies the synaptic weight between neuron i and neuron j, with "θ" serving as the predetermined threshold value. Taking the value of 0 can effectively ensure the energy monotonic reduction in the network and allow the network to reach a stable state [11]. It is worth mentioning that the synaptic weight of PRO2SAT has no self-loop where $W_{ii} = W_{jj} = 0$, $W_{ij} = W_{ji}$. The synaptic weight will be stored as control-addressable memory (CAM). When the neuron is in the active state, it can be assigned as 1; otherwise, the neuron is in the inhibited state and can be assigned as -1. The cost function represents the "inconsistency" in the logic rule. Therefore, the inconsistency in the logic needs to be reduced by minimizing the cost function ($C_{PRO2SAT} = 0$). In PRO2SAT, every neuron is connected to one another through the following cost function:

$$C_{PRO2SAT} = \frac{1}{4} \sum_{i=1}^{n} \prod_{j=1}^{2} L_{ij} \tag{7}$$

where n represents the number of second-order clauses and L_{ij} is the inconsistency of $P_{PRO2SAT}$. It is calculated as shown in Equation (8).

$$L_{ij} = \begin{cases} 1 + S_j, & x_j \\ 1 - S_j, & \neg x_j \end{cases} \tag{8}$$

where S_j represents the state of the j-th neuron whose value is bipolar with $S_j \in \{-1, 1\}$. It is worth noting that $C_{PRO2SAT} = 0$ means that all clauses C_i in the logic rule $P_{PRO2SAT}$ are satisfied. $C_i = 1$, $C_{PRO2SAT} = y$, which means that y clauses in the logic rule are not satisfied. The cost function is crucial in the learning phase, and its minimization can assist PRO2SAT to find the optimal synaptic weights and generate a good energy profile in the retrieval phase. The minimized probability of $C_{PRO2SAT}$ for PRO2SAT is $(0.75)^n$, and it is small as the number of neurons and clause n becomes larger. PRO2SAT calculates the final state of the neuron by means of a local field, as shown in Equation (9):

$$h_i = \sum_{j=1, i \neq j}^{NN} W_{ij}^{(2)} S_j + W_i^{(1)} \tag{9}$$

In the retrieval phase, PRO2SAT uses the hyperbolic activation function (HTAF) to assist the local field to calculate the neuron final state, because the HTAF can help reduce the neuron oscillation tendency and assist PRO2SAT to converge to the global minimum energy. The final state of the neuron is shown in Equation (10) and the HTAF is shown in Equation (11):

$$S_i(t) = \begin{cases} 1, & if \tanh[h_i] > 0 \\ -1, & if \tanh[h_i] < 0 \end{cases} \tag{10}$$

$$tanh[h_i] = \frac{e^{h_i} - e^{-h_i}}{e^{h_i} + e^{-h_i}} \tag{11}$$

The final energy of PRO2SAT can be calculated using Equation (12), which is called Lyapunov energy function. It is worth mentioning that, according to the WA method, the cost function ($C_{PRO2SAT}$) is compared with the Lyapunov energy function ($E_{PRO2SAT}$) to compute the optimal synaptic weights $W_{ij}^{(2)}$ and $W_i^{(1)}$. Section 6 introduces the MTS metaheuristic algorithm, which will assist PRO2SAT in the learning phase to find the consistent interpretation when the neurons are large.

$$E_{PRO2SAT} = -\frac{1}{2}\sum_{i=1,i\neq j}^{NN}\sum_{j=1,i\neq j}^{NN} W_{ij}^{(2)} S_i S_j - \sum_{i=1}^{NN} W_i^{(1)} S_j \tag{12}$$

where NN represents the number of neurons. According to Equation (12), we can derive the calculation of the minimum energy, as shown in Equation (13). In the learning phase, the Lyapunov energy function obtains the minimum value when $C_{PRO2SAT} = 0$, where n represents the number of second-order clauses. According to Jamaluddin et al. [22], the energy of the DHNN always converges to near the global minimum energy in the search phase. Therefore, we need to focus on PRO2SAT to find the global minimum energy as much as possible.

$$E_{PRO2SAT}^{\min} = -\frac{n}{4} \tag{13}$$

The expression for the global minimum energy error of PRO2SAT is provided as Equation (14):

$$\left| E_{PRO2SAT} - E_{PRO2SAT}^{\min} \right| \leq Tol \tag{14}$$

Global minimum energy error is used to determine whether the correct final neuron state is obtained, where Tol is the tolerance value determined by the researcher, and the values is assigned as 0.001 [8].

5. Objective Function of PRO2SAT in Learning Phase

The main objective of the learning phase of PRO2SAT is to find a set of optimal synaptic weights. With the increase in the number of neurons, however, especially when it is greater than or equal to 60 neurons, it is difficult for PRO2SAT to find optimal synaptic weights anymore. Therefore, the metaheuristic algorithm is implemented into PRO2SAT to find the consistent interpretation. From this, the optimal synaptic weights can be obtained by comparing the cost function and the energy function. The objective function of the metaheuristic algorithm in PRO2SAT is shown in Equation (15):

$$Max[f_{P_{PRO2SAT}}(S_i)] \tag{15}$$

where S_i is the solution string; for example, when the number of neurons is 6, its value can be expressed as $S_i = (-1, 1, 1, 1, -1, 1)$. $f_{P_{PRO2SAT}}(S_i)$ represents the fitness function. S_i is used as input to $P_{PRO2SAT}$ obtain the fitness, and its specific calculation process is shown in Equation (16):

$$f_{P_{PRO2SAT}}(S_i) = \sum_{i=1}^{n} f_{P_{PRO2SAT}}(C_i) \tag{16}$$

where n is the number of second-order clauses, C_i represents the i-th second-order clauses, and $f_{P_{PRO2SAT}}(S_i)$ represents the fitness value of the i-th clause of PRO2SAT, which is calculated as shown in Equation (17):

$$f_{P_{PRO2SAT}}(C_i) = \begin{cases} 1, & if\ satisfied \\ -1, & if\ unsatisfied \end{cases} \tag{17}$$

In $P_{PRO2SAT}$, a C_i clause is considered to obtain the consistent interpretation when it is satisfied at an input value of S_i. Take 1 for the fitness value of MTS. Otherwise, the clause is not consistent and the fitness is -1. The purpose embedding PRO2SAT in the learning phase of the metaheuristic algorithm is to find the maximum fitness n, where n represents the number of second-order clauses.

6. Proposed Metaheuristics

Inspired by different natural phenomena, mathematical models are built in metaheuristic algorithms. Based on this, metaheuristic algorithms can be classified into three categories: evolutionary algorithms, physical mechanism algorithms, and swarm intelligence algorithms [23], where the primary role of evolutionary algorithms is to simulate the computation model of the biological evolution process by natural selection and genetic mechanism of Darwin's theory of biological evolution. Such algorithms are mainly meant to eliminate the individuals with poor fitness values and retain those with good genes through the continuous evolution of individuals. The evolutionary algorithms used in this paper involved GA, DE, and EDA. Physical mechanism algorithms are algorithms based on physical phenomena and physical laws, which optimize the answers to the problems mainly by simulating physical phenomena, such as temperature, energy, and force. The physical mechanism algorithm used in this paper encompassed the SA algorithm. Swarm intelligence algorithms are algorithms that simulate group behavior, which optimize the answers to the problems mainly by simulating group behavior. The swarm intelligence algorithms used in this paper included EA, GWO, ant algorithm, PSO, and ABC.

6.1. Proposed Mutation Tabu Search Algorithm (MTS)

The TS algorithm is an intelligent metaheuristic algorithm proposed by Glover, a famous American scholar, in the 1990s. The algorithm is a typical stochastic search algorithm [24], which mimics the human thought process and extends the scope of local domain search. In the search process, the TS algorithm first searches for some local optimal solutions, and records these objects by the tabu table, avoiding these local optimal solutions that have been searched for in the following iteration process, obtaining a variety of regions and improving the efficiency of the search. It is worth noting that a flouting rule is introduced into the MTS algorithm to release some excellent tabu solutions and avoid the loss of optimal solutions. The MTS algorithm has the advantages of fast convergence and the capability of avoiding local minima. In view of this, in this paper, the MTS algorithm is embedded into the systematic logic (PRO2SAT) for the first time, where mutation operators are embedded into MTS to assist in the faster convergence of the systematic logic PRO2SAT in the learning phase and to guarantee the optimal solution in the retrieval phase.

6.1.1. Initialization

Initialization of the initial solutions: the MTS algorithm starts from one initial solution. The convergence of this algorithm is closely related to the quality of the initial solution. A high-quality initial solution facilitates the rapid convergence of the algorithm during execution, while a lower-quality initial solution slows down the search speed. In order to better utilize the performance of the MTS algorithm, it is necessary to select one superior initial solution. In the PRO2SAT model, c solution strings are randomly generated in the learning phase, and each solution string is denoted by S_i, $S_i = \{s_1, s_2, s_3, \cdots, s_{NN}\}$, $s_i \in (-1, 1)$, $NN = 100$, where fitness $f_{P_{PRO2SAT}}(S_i)$, $i \in (1, 2, 3, \cdots, 100)$ is calculated for

each solution. Therefore, we selected the solution with the maximum fitness value as the initial solution, as shown in Equation (18):

$$S_{Initial} = S_i,\ if\ Max(f_{P_{PRO2SAT}}(S_i)) \tag{18}$$

Initialization of the local optimal solution and the global optimal solution: X_{best} and G_{best} are defined to represent the local optimal solution and the global optimal solution, respectively. X_{best} stores the solution with the maximum fitness among the neighborhood solutions under a certain cycle of the MTS algorithm, and G_{best} stores the solution with the maximum fitness among all iterations of the MTS algorithm. The initial values of the local optimal solution and the global optimal solution are the initial solutions, as shown in Equation (19).

$$X_{best} = G_{best} = S_{Initial} \tag{19}$$

By initializing the local and global optimal solutions with the initial solution, it ensures that the algorithm starts its search from a reasonable starting point and gradually optimizes to find better solutions.

6.1.2. Generation Strategy to Neighborhood Solution

A new solution can be obtained by performing certain operations on the initial solution, and these operations are called neighborhood operations. The option is critical in neighborhood operation, and the neighborhood operations vary from the results. Inspired by GA, we used mutation operations to generate neighborhood solutions. Firstly, the initial solution was divided into w segments equally by neurons, x segments were randomly selected among them, and y neurons were randomly selected in the selected segment for mutation operations, i.e., flipping the values of the selected neurons. For example, if the value of the selected neuron is -1, the mutation becomes 1. On the contrary, if the value of the selected neuron is 1, the mutation becomes -1. Figure 1 shows the generation process of the neighborhood solution when w takes the value of 5, x takes the value of 2, and y takes the value of 1. In this step, N replicates generate N neighborhood solutions, denoted by $NS_1, NS_2, NS_3, \cdots, NS_N$.

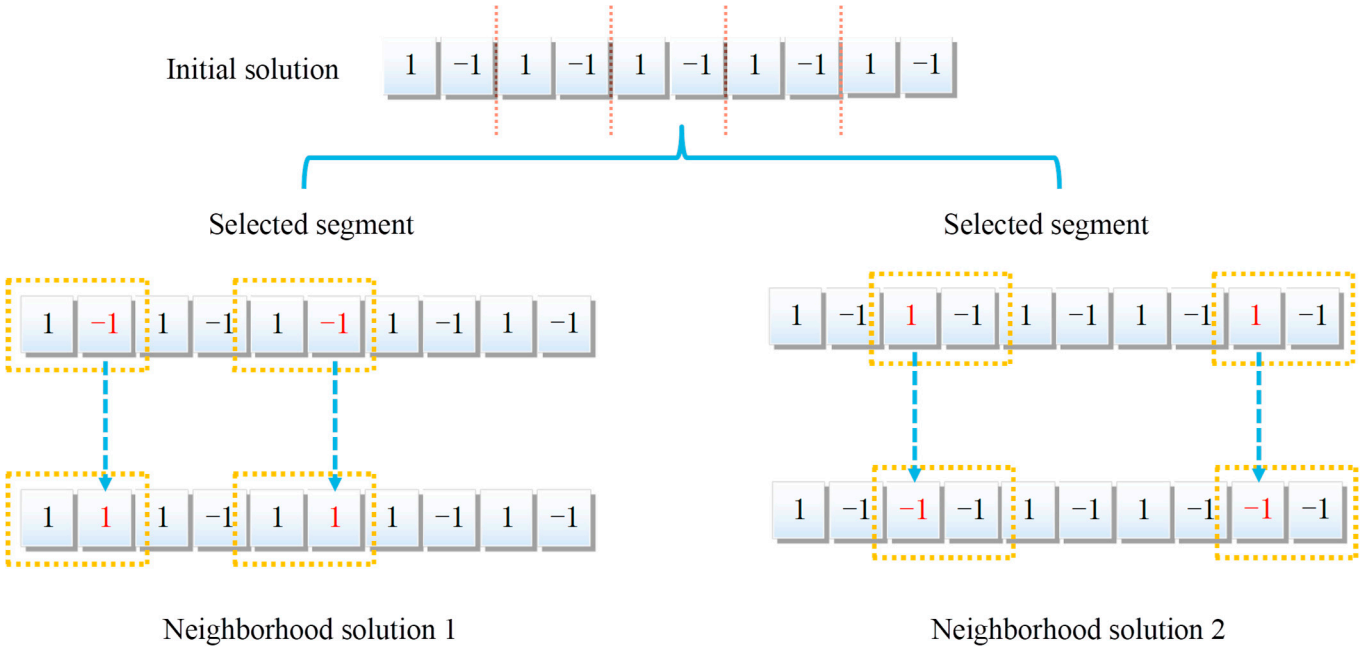


Figure 1. Generation process of neighborhood solution.

6.1.3. Generation Strategy to Candidate Solution

Candidate solution NS_*: the neighborhood solution with the maximum fitness is selected as the candidate solution, as defined in Equation (20).

$$NS_* = NS_i,\ if\ Max(f_{P_{PRO2SAT}}(NS_i)) \tag{20}$$

Tabu table (*TBT*): it refers to the storage structure used to store the tabu operations, where two main indicators are available: tabu operations (*TBS*) and tabu length (*L*). In this paper, the mutation segment number of the neighborhood solution was used as the tabu operation to avoid the repeated search of the obtained solution, expand the search area of the algorithm, and escape the local optimum. The tabu length is usually set to a positive integer; too short a tabu length tends to fall into the local optimal solution prematurely, and too long a tabu length leads to prolonged computation. As a result, all the neighborhood solutions are tabued, and it is impossible to continue computing. Therefore, according to [25], we set the tabu length to 3.

$$TBS_{NS_*} = (SN_1, \cdots, SN_{ns}) \tag{21}$$

$$TBT_i = TBT_{NS_j},\ if\ TBT_{NS_j} \neq\ all\ TBT_i \tag{22}$$

where i takes an integer and $i \in [1, L]$, ns takes an integer, and $ns \in [1, NN]$.

Update of the local optimal solution and the global optimal solution: it determines whether the mutation segment number of the candidate solution has been stored in *TBT* when implementing the neighborhood operation. If not, it is written to *TBT*. The local optimal solution is updated and the candidate solution values are assigned to the local optimal solution. Note that, if the new local optimal solution outperforms the global optimal solution, it is necessary to update the global solution, and assign the new local optimal solution value to the global optimal solution. If the mutation segment number of the candidate solution has been recorded in *TBT*, in order to expand the search domain of the MTS algorithm, the current candidate solution should be discarded and the solution with the maximum fitness among the remaining solutions in the neighborhood solution should be selected as the new candidate solution NS^*_{t+1}. The above process is repeated until the local optimal solution and the global optimal solution are generated.

$$X^{t+1}_{best} = NS^t_*,\ if\ TBT_{NS^t_*} \neq\ all\ TBT_i \tag{23}$$

Flouting rule: If the mutation segments number of the candidate solution is available in *TBT*, but the fitness is greater than the current global optimal solution, the tabu operations can be neglected. The mutation segment number of the candidate solution is written to *TBT*, and the local optimal solution and the global optimal solution are updated accordingly, as defined in Equations (24) and (25).

$$X^{t+1}_{best} = NS^t_*,\ if\ TBT_{NS^t_*} = TBT_i\ and\ f_{P_{PRO2SAT}}(NS^t_*) > f_{P_{PRO2SAT}}(G^t_{best}) \tag{24}$$

$$G^{t+1}_{best} = \begin{cases} X^{t+1}_{best},\ if\ TBT_{NS^t_*} = TBT_i\ and\ f_{P_{PRO2SAT}}(G^t_{best}) < f_{P_{PRO2SAT}}(X^{t+1}_{best}) \\ X^t_{best},\ otherwise \end{cases} \tag{25}$$

The flouting rule is typically executed after a certain number of iterations, aiming to revoke previously tabu neighborhood operations. It allows for the algorithm to tolerate neighborhood operations previously deemed prohibitive, provided that the fitness is sufficiently satisfactory, thus preventing undue constraints during the search process. This facilitates the algorithm's escape from local optimal solutions. The generation process of candidate solution is shown in Figure 2.

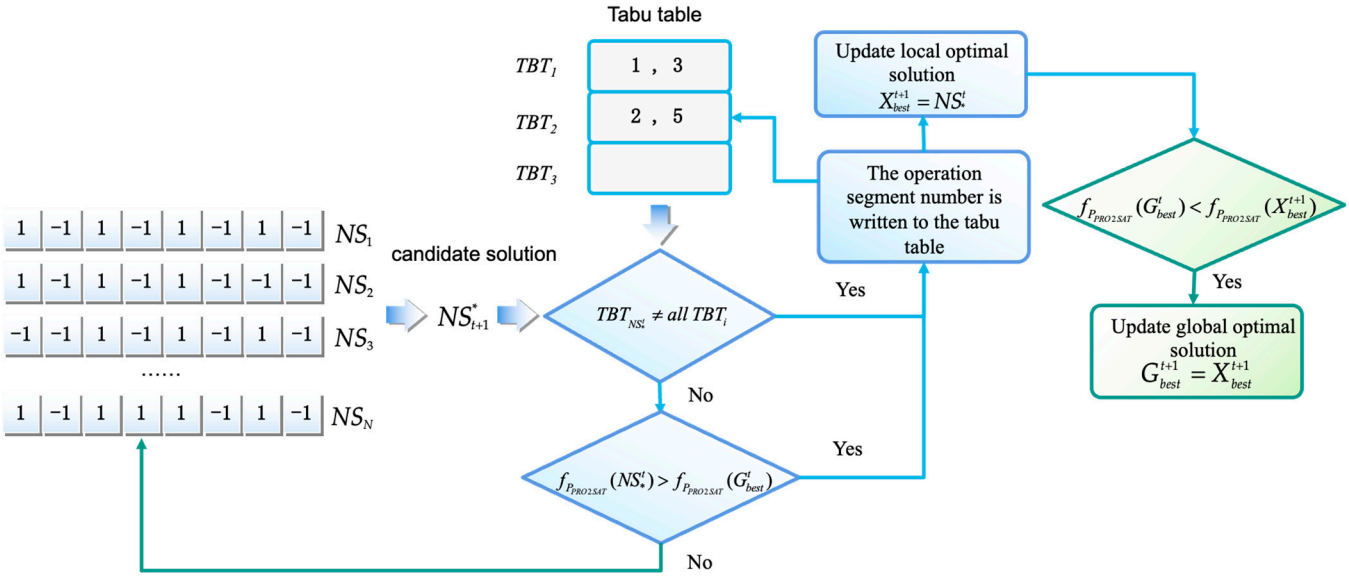


Figure 2. Generation strategy to candidate solution.

6.1.4. Fitness Assessment

The fitness value of G_{best} is calculated according to Equation (16). If the solution fails to reach the maximum fitness and the algorithm fails to reach the maximum number of iterations, return to 5.1.1 and continue to the next round.

6.1.5. Mutation

At the beginning of the test phase, 100 solution strings will be generated randomly in PRO2SAT in order to increase the diversity of the solutions in the retrieval phase. The MTS metaheuristic algorithm will flip the satisfied clauses of the global minimum solution by virtue of the mutation operators after calculating the local field in the retrieval phase as shown in Equations (26) and (27):

$$s_{2i-1} = -1 \cdot s_{2i-1},\ if\ C_i\ satisfied\ and\ rand(0,1) = 0 \tag{26}$$

$$s_{2i} = -1 \cdot s_{2i},\ if\ C_i\ satisfied\ and\ rand(0,1) = 1 \tag{27}$$

6.2. Baseline Model

The model proposed in this paper will be compared with other established existing algorithms. The baseline models in this paper include genetic algorithm (GA), election algorithm (EA), ant colony optimization (ACO), estimation of distribution algorithm (EDA), differential evolution algorithm (DE), grey wolf optimization algorithm (GWO), particle swarm optimization (PSO), simulated annealing algorithm (SA), and artificial bee colony algorithm (ABC). The following content summarizes their working principles.

(a) GA [11]: This study integrated the strengths of the genetic algorithm and Hopfield neural network to efficiently find solutions to the logic satisfiability problem. The genetic algorithm is a global search algorithm which usually uses a binary encoding technique for optimization problems [26]. At the time of solving, the initial solution to the actual problem is coded to form a gene string, i.e., a chromosome, which is selected, crossed, and mutated to form a new chromosome. The resulting chromosome will be retained if it is closer to the maximum fitness than the previous one.

(b) EA [12]: The EA algorithm is inspired by the election of a national president. The algorithm was proposed by Emami, H [27], which combined the features of evolutionary algorithms and SIA. In this algorithm [13], positive advertisement, negative advertisement, and coalition were used to implement intelligent search in a synergy

mechanism. All solutions will be considered as voters from whom candidates will be selected, and each candidate determines his own voters based on his social relationship to the voters, thus forming a political party. As the leader of this party, the candidate will positively influence his own supporters (voters) through positive advertisement and positively influence the supporters (voters) for the leaders of other parties through negative advertisement in order to expand the search space and increase their probability of being elected. The most popular candidate will ultimately receive the most votes.

(c) ACO [14]: Kho introduced the ACO algorithm in HDNNs (Kho, 2021). In this work, the ant colony algorithm is used to minimize the cost function of the corresponding logic rule in DHNNs. In the ACO algorithm, the pheromone density is used to find the optimal path, thus achieving a zero-cost function without consuming more learning iterations. In this study, the potential application of the ACO algorithm was fully demonstrated in optimization problems, including propositional logic.

(d) EDA [28]: This algorithm is a probability-based population evolution algorithm. By generating a new population through random sampling, the evolution of the population is achieved through iterations. The main function of MTS is to estimate and predict the distribution of the data. This algorithm is able to predict future data trends by inferring the distribution of the data through statistical analysis. The standard EDA has two important operations, namely the selection operations and the modeling of the probability distribution. The selection operation is the same as the selection strategy in GA. The probability distribution model can be a univariate marginal distribution algorithm (UMDA), which is calculated.

(e) DE [29]: A novel binary differential evolution algorithm based on Taper-shaped transfer functions (T-NBDE) is proposed to address the knapsack problem in [29]. The DE and GA algorithms are both evolutionary algorithms, which are adaptive global search algorithms first proposed by Price and Storn in the 1990s to solve the real number solution optimization problems. The DE algorithm has the main features of a simple structure, easy implementation, robustness and fast convergence, etc. In addition, the DE algorithm also has memory function, which can dynamically track the search situation, and the control parameters mainly include population size, variation operator, crossover operator, and selection operator. Unlike the GA algorithm, the variation operators of DE randomly select three individuals as parents for mutation operation to form new individuals.

(f) GWO [23]: The GWO algorithm has been successfully applied to RDHNNs in the work by Ba et al. [16]. Therefore, we used GWO in the learning phase of PRO2SAT for comparative analysis. It is a heuristic optimization algorithm based on the behavior of grey wolf packs in nature, which simulates the social hierarchy of grey wolves and divides the individuals within the pack into four classes: head wolf (X_α), subordinate wolf (X_β), common wolf (X_δ), and bottom wolf (X_ω). In the GWO algorithm, each grey wolf represents a potential solution, and each wolf has an adaptation value. The higher the adaptation value, the better the indicated solution. Starting from any position in the solution space, the individual with the best fitness is set as the leader wolf α, the one with the second fitness is set as the subordinate wolf β, the one with the third fitness is set as the ordinary wolf δ, and the rest are the bottom wolves ω. The leader wolf is responsible for guiding the behavior of the pack, the subordinate wolf assists the leader wolf in making decisions, the ordinary wolf obeys the leader wolf and the subordinate wolf, and dominates the bottom wolf to catch and hunt the target. The bottom wolf needs to obey the guidance of other wolves and follows other wolves to complete hunting, and mainly takes charge of the balance of intra-pack relationships.

(g) (PSO [30]: The PSO algorithm, proposed by Eberhart and Kenndy in 1995, is a type of SIA algorithm, which is widely used in the field of combinatorial optimization, and it can be applied to PRO2SAT. Inspired by the foraging behavior of a flock of birds,

this algorithm includes evolutionary theory. The idea of the PSO algorithm is that each individual searches for a better solution based on the optimal solution that has been found and compares the optimal solution currently found by the population to update its speed. The algorithm searches for the global optimal solution by constantly updating the position and velocity of the population, which is a process of movement from simple individuals to complex global solutions.

(h) SA [31]: The SA algorithm is a probability-based search algorithm proposed by Kirkpatrick, Gelatt, and Vecchi in 1983 to describe the physical annealing process of an object. It can be used to solve complex optimization problems. The basic idea is the process of finding the global optimal solution of the objective function randomly in the space of all local solutions starting from an initial solution (annealing point) in an isothermal process combined with the probabilistic sudden jump property, i.e., the ability to probabilistically jump out of each local solution and finally obtain the global optimal solution. The advantage of the SA algorithm is that it can find the global optimal solution and have a relatively large search space to find a better solution.

(i) ABC [13]: This study introduces the combination of the artificial bee colony algorithm with the Hopfield network to minimize or maximize the cost function of any combinatorial problem. With this literature approach, we processed the secondary values using the ABC algorithm. Each bee was assigned an initial nectar source, and the employed bee dances through the solution space to compute the new nectar source. This explores the solution space of consistent solutions during the learning phase of the Hopfield neural network and identifies potential solutions. The combination of the ABC algorithm and the Hopfield neural network demonstrates the superior performance of the artificial bee colony in solving the 2 Satisfiability problem.

7. Experimental Setup

In this experiment, we verified the validity of the MTS model proposed in this paper in the learning phase and the retrieval phase, and considered the metaheuristic algorithm to achieve a single objective function, i.e., the proposed model achieved the maximum fitness.The flowchart of MTS model is shown in Figure 3. To ensure the reproducibility and fairness of the experiment, our experimental setup is as follows.

7.1. Simulation Design

All simulations were conducted using the same set of features to prevent bias during the experiments, encompassing three specific characteristics. Experimental environment and programming language: We opted for PyCharm as our development tool, and all experimental models were coded using Python. The operating environment of the experiment was mac OS Monterey (12.1) with an apple M1 pro CPU, an apple M1 pro graphics card, and 16 GB of RAM. Note that the experimental models operated on the same devices to ensure fairness in the comparison of performance metrics. Setup for the number of neurons: The number of neurons is taken in the range of $10 \leq NN \leq 120$. PRO2SAT consists of the second-order logical clauses only, and this fixed structure is beneficial to the stability and fairness of the experimental results. It is worth mentioning that PRO2SAT specifies the proportion of the number of positive literals in the logic structure. Similarly, to ensure the fairness of the experiment, we divided PRO2SAT into nine types (The proportion of positive literals is taken in the range of [0.1, 0.9] and the step size is taken as 0.1). Each PRO2SAT was applied to 10 metaheuristics and exhaustive search algorithms proposed in this paper for comparison experiments. Maximum iteration limit: We set the maximum iteration limit to 100 for all metaheuristics.

7.2. Parameters Assignment

Table 2 lists the parameters used in the learning phase of MTS. It was found by a certain study [32] that an increased size of populations contributes to the chance of the algorithm to obtain the optimal solution. Therefore, in order to make the evaluation of each

algorithm more objective and convincing, we set the populations of all algorithms except ABC to 100.

Table 2. List of parameters for PRO2SAT-TS model.

Parameter	Parameter Value
Number of neurons (NN)	$10 \leq NN \leq 120$
Neuron combination (a)	100
Number of trials (b)	100
Number of learnings (c)	100
Tolerance value (Tol)	0.001
Activation function	HTAF
Synaptic weight method	Wan Abdullah method
Initialization of neuron states	Random
CPU computing time	24 h
Ratio of positive literal (η)	$\{0.1, 0.2, \cdots, 0.9\}$
Control probability (p_l)	$\{\xi$,max$(\eta, 1-\eta)\}$
Length of tabu table (L)	3
Number of segments (w)	8
Number of neighborhood operation segments	2
Number of mutated neurons	2
Neighborhood solution	14
Selection rate	1
Mutation rate	1
Maximum number of iterations	100

7.3. Performance Evaluation Metrics

In order to comprehensively evaluate the performance of PRO2SAT with the assistance of different metaheuristic algorithms, nine metrics were selected to analyze PRO2SAT. Since the metaheuristic algorithms acted in the learning phase of PRO2SAT and were used to assist in finding consistent explanations, the metrics were mainly meant to evaluate the number, quality, and efficiency of the solutions obtained by the proposed metaheuristic algorithm versus the baseline metaheuristic algorithm. The performance of all PRO2SAT models was evaluated based on different computational phases, each with the following two objectives:

Firstly, during the learning phase, we evaluate the efficiency and quality of the metaheuristic algorithm to minimize the cost function from macro (logic rules) and micro (clauses) perspectives. Secondly, an effective learning phase will help PRO2SAT to obtain better final neuron states and have better convergence in the retrieval phase. Therefore, we mainly evaluated the ability of the metaheuristic algorithm to facilitate PRO2SAT to obtain globally optimal solutions with small energy differences in the retrieval phase.

7.3.1. Learning Phase Metrics

(a) Mean absolute error of clause adaptation (MAE_{learn})

This metric is used to measure the ability of the metaheuristic algorithm to search for clause interpretations during the learning phase, i.e., the ability to find clause fitness, and to analyze PRO2SAT from a microscopic perspective. Good metaheuristic algorithms can rapidly improve the fitness of PRO2SAT clauses until they reach the maximum fitness (all clauses are satisfied).

$$MAE_{learn} = \frac{1}{a}\sum_{j=1}^{a}\sum_{i=1}^{n}\frac{|f_{\max} - f_i|}{n} \tag{28}$$

where $f_{\max}$ is the maximum fitness of a logic rule, and the value is equal to the number of second-order logic clauses. f_i is the actual fitness, and the value is equal to the number of second-order logical satisfiable clauses. a is the number of combinations, i.e., the number of times needed to generate $P_{PRO2SAT}$. n is the number of trials when $f_{\max} = f_i$.

(b) Mean absolute error of the adaptation of logic rules ($LRTR$)

This metric is used to measure the ability of the metaheuristic algorithm to search for a consistent interpretation of the logical rules in the learning phase. It analyzes PRO2SAT from a macro perspective, focused on the ability of the metaheuristic algorithm to obtain a solution.

$$LRTR = \frac{1}{a}\sum_{i=1}^{a} D_{satisfaction-learn} \tag{29}$$

where $D_{satisfaction-learn}$ represents whether the consistency explanation is found in 100 trials, and the value is 1 if found; otherwise, it is 0.

(c) Mean similarity of consistency interpretations (GLI_{avg})

$$S_i^{bs} = \begin{cases} 1, \ if \ \neg x \\ -1, \ if \ x_i \end{cases} \tag{30}$$

It can be seen that the benchmark state is only related to the positive and negative literals. The benchmark state is "1" for positive literals and "-1" for negative literals. For example, if the logic rule $P_{PRO2SAT}$ is $(\neg x_1 \vee x_2) \wedge (\neg x_3 \vee \neg x_4) \wedge (\neg x_5 \vee \neg x_6) \wedge (\neg x_7 \vee \neg x_8)$, the benchmark state is (-1,1,-1,-1,-1,-1). In this paper, we analyzed the diversity of solutions by comparing the consistency interpretation obtained by benchmark states and the metaheuristic algorithms, and the specific method for comparison is shown in Equation (44).

In order to analyze the quality of the solutions obtained by the metaheuristic algorithm in the learning phase, the similarity metrics were used to observe the diversity of solutions. It is worth mentioning that a certain logical rule $P_{PRO2SAT}$ has more than one consistent interpretation in the learning phase, but they have multiple benchmark states obtained by Equation (31):

$$C_{S_i^{bs},S_i} = \left\{ (S_i^{bs}, S_i); \ \ i = 1, 2, 3, \cdots m \right\} \tag{31}$$

where m represents the number of neurons and S_i represents the state value of the neuron when the metaheuristic algorithm obtains a consistent interpretation. The standard canonical variables in the comparison process can be determined by the following method:

l indicates the overall occurrence count of $(S_i^{bs} = 1, S_i = 1)$ in $C_{S_i^{bs},S_i}$;

m indicates the overall occurrence count of $(S_i^{bs} = 1, S_i = -1)$ in $C_{S_i^{bs},S_i}$;

n indicates the overall occurrence count of $(S_i^{bs} = -1, S_i = 1)$ in $C_{S_i^{bs},S_i}$;

o indicates the overall occurrence count of $(S_i^{bs} = -1, S_i = -1)$ in $C_{S_i^{bs},S_i}$.

According to the calculation of the standard specification variables above, the similarity metric GLI is calculated as shown in Equation (19), where the larger GLI represents a more diverse solution and the opposite represents a solution closer to the standard solution. From the point of view of the quality of the solution required, we hope to find the solution with a higher quality, i.e., the solution obtained is closer to the standard solution. This can be expressed by the average similarity metric in Equation (33).

$$GLI = \frac{lo}{l + 0.5(m+n) + o} \tag{32}$$

$$GLI_{avg} = \sum_{i=1}^{\lambda} GLI_i \tag{33}$$

where λ is the number of solutions.

(d) Computational Time (CT)

This metric is used to determine the effectiveness of the proposed model. It implies the capability and stability of the model. It is calculated as shown in Equation (34):

$$CT = \frac{1}{a}\sum_{i=1}^{a} MT_i \tag{34}$$

where MT_i is the average time for 100 trails performed by the metaheuristic algorithm for a given number of neurons. The time is expressed in seconds.

(e) Average iterations (AI)

This metric is used to demonstrate the speed of convergence of the algorithm and will be analyzed together with the metric CT to determine the efficiency of the metaheuristic algorithm. It is calculated as shown in Equation (35). It has been shown that the better the metaheuristic algorithm, the fewer iterations it will have.

$$AI = \frac{1}{a}\sum_{i=1}^{a} LP_i \tag{35}$$

where LP_i represents the number of iterations when the metaheuristic algorithm finds a one-time explanation under each combination. It is worth noting that we set the maximum number of iterations for the metaheuristic algorithm, and LP_i is counted as the maximum number of iterations when the maximum metaheuristic algorithm reaches the maximum number of iterations and still does not obtain a consistent explanation.

7.3.2. Retrieval Phase Metrics

(f) Global minimum proportion (ZM)

The global minimum solution is obtained by converging to the optimal final neuron state in a good retrieval phase. In this paper, we use this metric to demonstrate the ability of the neural network to obtain the global minimum solution. The general form of global minimum proportion is shown in Equation (36):

$$ZM = \frac{1}{ab}\sum_{i=1}^{ab} G_P \tag{36}$$

where G_P stands for the count of neuron states in the model achieving the global minimum energy, while a denotes the number of combinations, and b signifies the total number of trials.

(g) The average similarity of solutions ($TV_{similar}$)

In addition to considering the number of global minimum solutions obtained by the model in the retrieval phase, the average similarity metric of solutions was first proposed to calculate the similarity between the non-repeated solutions and the benchmark solutions in the retrieval phase. The larger the value of this metric indicates that the solution is more identical to the standard solution, and the worse the variability ability to the solutions and diversities is. On the contrary, the larger the difference between the solution and the standard solution, the better the variation ability and diversity of the solution.

$$TV_{similar} = \frac{\sum_{i=0}^{\lambda}(l_i + o_i)}{\lambda NN} \tag{37}$$

where λ is the number of non-repeating solutions in the retrieval phase, and l and o have the same meanings as (c).

(h) Mean Absolute Error of Logic Rule Energy (MAE_{test})

This metric is used to evaluate the energy error generated by the neuron, which will obtain the final energy after the final state of the nerve is obtained via the local field. The smaller energy error represents the better quality of the final state of the neuron. It is calculated according to Equation (38):

$$MAE_{test} = \sum_{i=1}^{ab} \frac{\left| E_{PRO2SAT}^{\min} - E_{PRO2SAT} \right|}{ab} \tag{38}$$

where $E_{PRO2SAT}^{\min}$ is the minimum energy and $E_{PRO2SAT}$ is the final energy.

(i) Friedman Statistical Analysis (F_d)

Friedman statistical analysis is a non-parametric test that determines whether there is a significant difference in the effectiveness of the metaheuristics by counting and rating the results of the metrics in the learning and retrieval phases of each metaheuristic algorithm. The Friedman test is represented in Equation (39) in its general form.

$$F_d = \frac{12N}{K(K+1)} \left(\sum_j R_j^2 - \frac{K(K+1)^2}{4} \right) \tag{39}$$

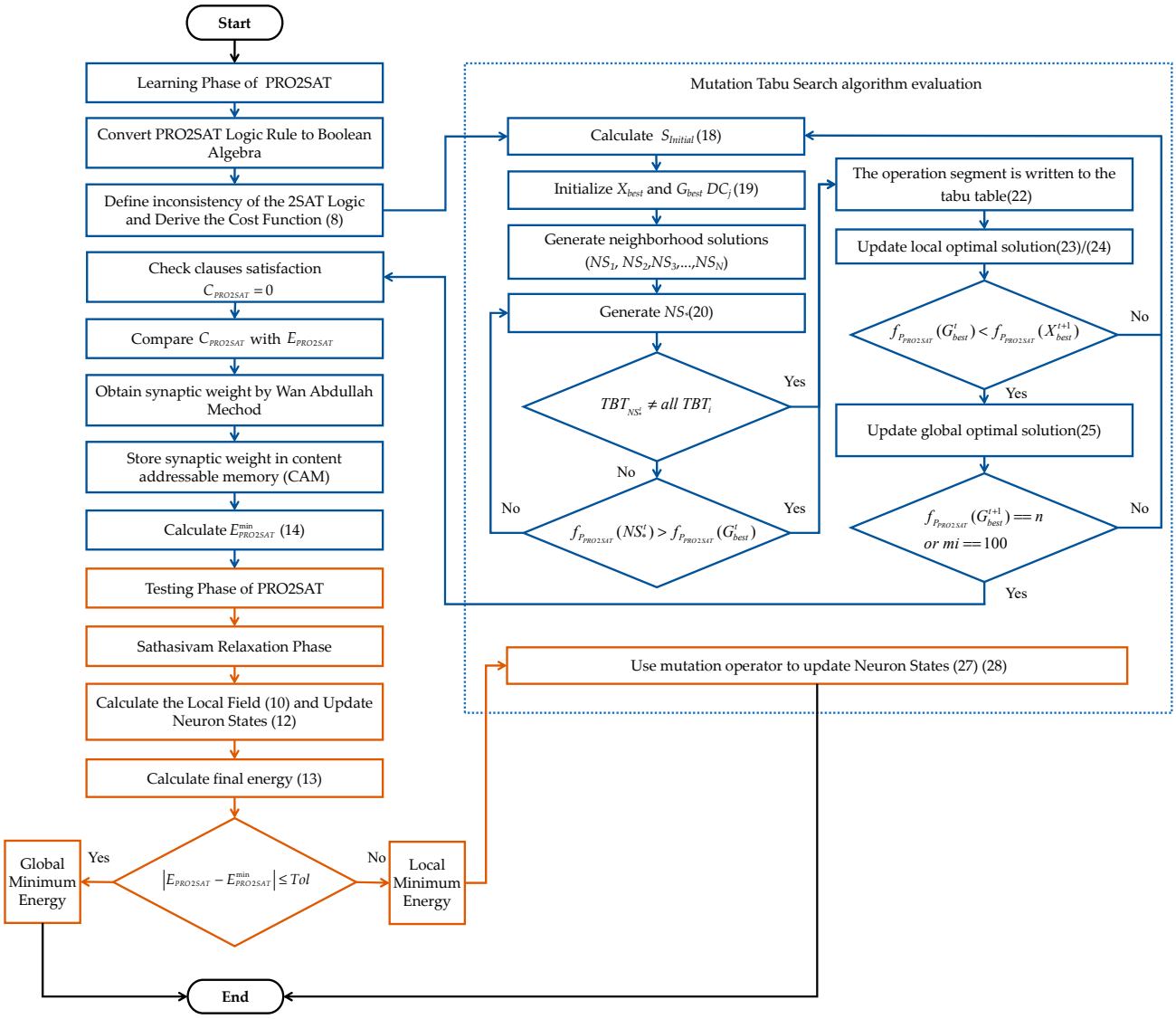


Figure 3. Flowchart of MTS.

7.4. Simulation Dataset

In this paper, we conducted the experiments by using simulated data in which the initial values of the neurons are bipolar and randomly generated. The simulated dataset is typically used to simulate and evaluate the performance of a newly proposed logic programming models with 100 randomly generated initial data in both the learning phase and the retrieval phase. This simulated dataset was employed following the works of Zamri et al. [11] and Someetheram et al. [12], allowing the learning phase metaheuristic algorithm to have an unbiased start, so that the final state of neurons in the retrieval phase can be determined by the optimal synaptic weights. The metaheuristic algorithm acts upon the learning phase of PRO2SAT, where each iteration is aimed to minimize the cost function. Since different proportions of positive literals were proposed in the PRO2SAT model, the randomized simulated experiment data make each simulation of the model independent and do not cause the experiment to converge early or fail to converge due to too many positive or negative literals.

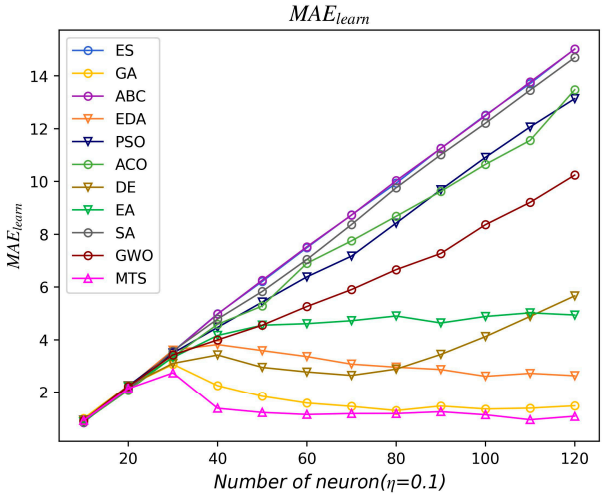
8. Result and Discussions

In this section, we will discuss the performance of the metaheuristic algorithm in the learning phase of PRO2SAT and the impact on the retrieval phase. MTS algorithms were first embedded into the systematic logic PRO2SAT, and nine typical meta-inspired algorithms embedded into PRO2SAT were compared and analyzed as baseline models. Our experiments were conducted to understand the effectiveness of the proposed MTS metaheuristic algorithm by limiting the number of neurons ($10 \leq NN \leq 120$). In this section, the abbreviations of the metaheuristic algorithms were used directly to represent the experimental models, e.g., MTS for PRO2SAT-MTS and GA for PRO2SAT-GA. In order to assess the algorithms' average performance and stability, we conducted five runs of each set of experiments, and the final result was determined as the average. It is important to note that the data used in these experiments were derived from a simulated dataset with a space size of 2^{NN}.

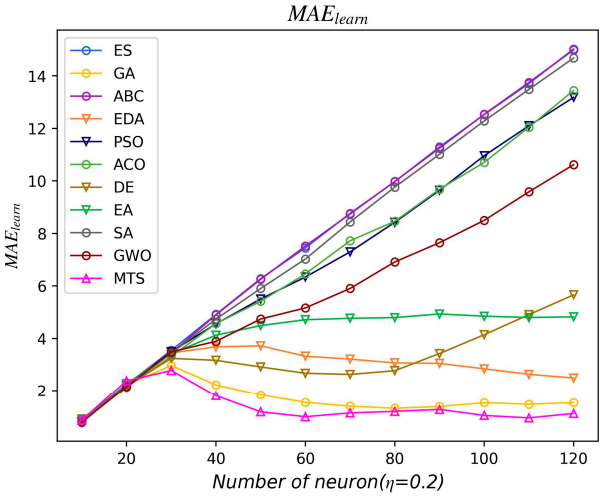
8.1. Learning Phase

The metaheuristic algorithm was embedded into the learning phase of PRO2SAT and was used to improve the ability of PRO2SAT to find consistent interpretation. We used MAE_{learn} performance metrics to analyze the variation in the fitness of PRO2SAT embedded in the metaheuristic algorithm for different proportions of positive literals in clauses, with the smaller MAE_{learn} representing the higher satisfied interpretation.

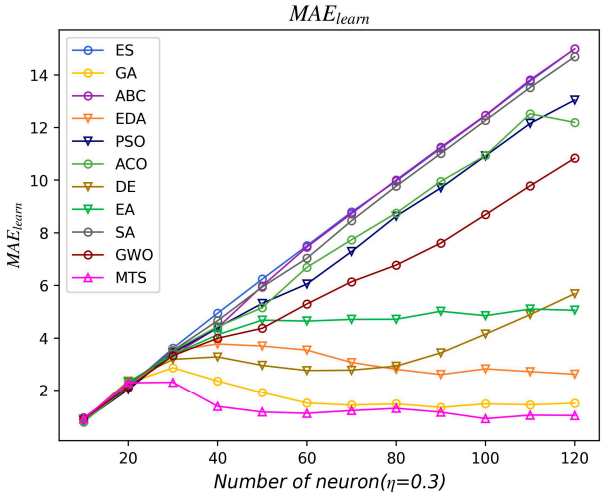
We used a line graph to show all the experimental models of MAE_{learn}, as shown in Figure 4. Since MAE_{learn} was used to describe the change in fitness, it was also referred to as fitness error. From the figure, we can see that, among them, the ES model has the maximum fitness error, which is due to the fact that an exhaustive search method is used in this model to generate the 100 sets of initial neuron states. No metaheuristic algorithm was used to explore the solution space, but it would be used as the most dominant comparison model. It is worth mentioning that, at $NN \leq 30$, all experimental models have no fundamental difference in the fitness error for different proportions of positive literal values; this is due to the stipulation in our code that the 10 metaheuristic algorithms of the experiment are implemented only if a consistent interpretation cannot be found in the set of 100 sets of initial neuronal states. Therefore, the metaheuristic algorithm was substantially not implemented in the experimental model at $NN \leq 30$.



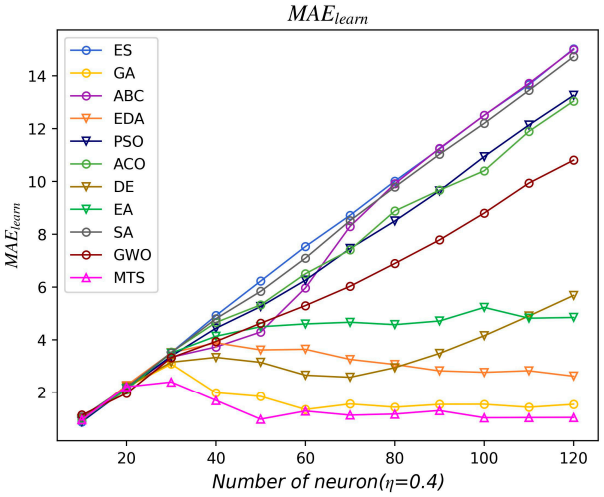
(**a**) MAE_{learn} evaluation when $\eta = 0.1$.



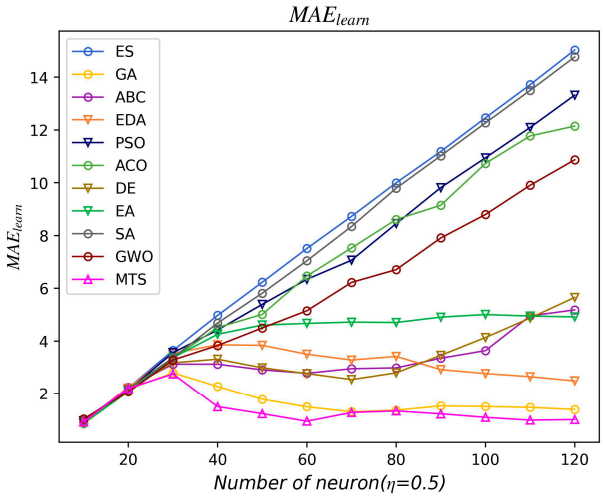
(**b**) MAE_{learn} evaluation when $\eta = 0.2$.



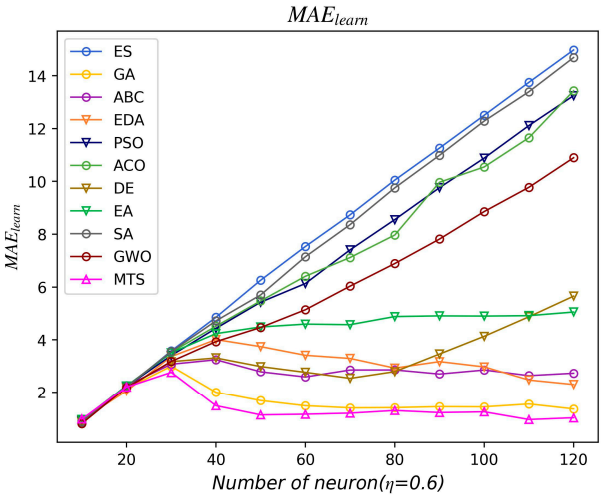
(**c**) MAE_{learn} evaluation when $\eta = 0.3$.



(**d**) MAE_{learn} evaluation when $\eta = 0.4$.

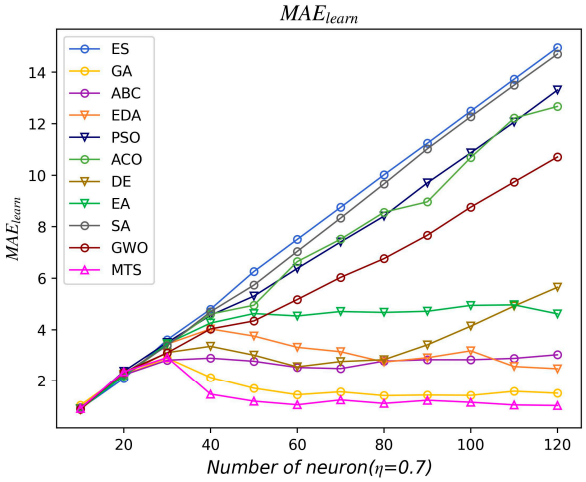


(**e**) MAE_{learn} evaluation when $\eta = 0.5$.

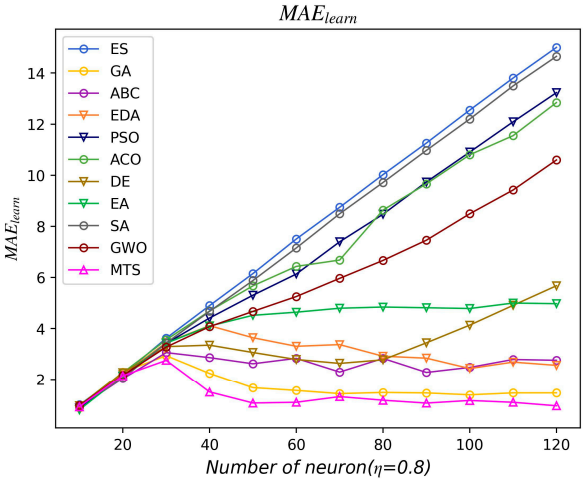


(**f**) MAE_{learn} evaluation when $\eta = 0.6$.

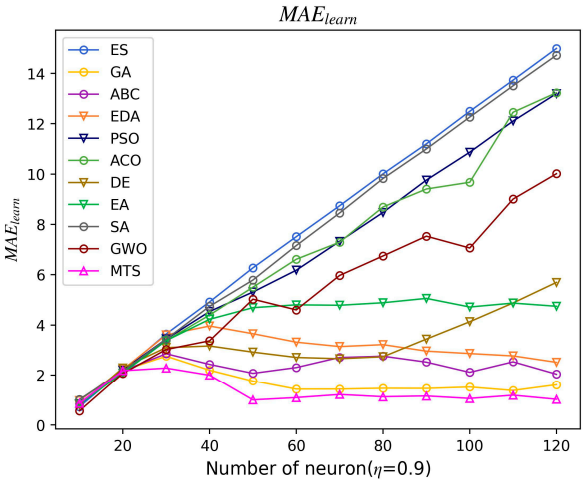
Figure 4. *Cont.*



(**g**) MAE_{learn} evaluation when $\eta = 0.7$.



(**h**) MAE_{learn} evaluation when $\eta = 0.8$.



(**i**) MAE_{learn} evaluation when $\eta = 0.9$.

Figure 4. MAE_{learn} evaluation for all experimental models.

Five models spanning PSO, ACO, SA, GWO, and DE show a linear increase in fitness error as the number of neurons increases. This indicates that the clauses of these algorithms are affected by the number of neurons. What is more interesting is that, in the ABC model, the fitness error is larger when $\eta \leq 0.4$, but smaller and stable when $\eta \geq 0.5$. This is because the ABC algorithm bees dance to find the new honey source operator $v_{ij} = x_{ij} \vee [\phi_{ij} \otimes (x_{ij} \wedge x_{kj})]$, where the $\vee$ operator represents OR. It can be seen that, if the old honey source x_{ij} is positive, the new honey source v_{ij} must be positive. The neuron state will change toward +1 as the iterations progress, which directly leads to poor satisfiability of the clause when $0.4 \geq \eta \geq 0.1$ and good satisfiability of the clause when $0.9 \geq \eta \geq 0.5$. In the four models involving EDA, EA, GA, and MTS, their fitness errors are presented as a horizontal line, indicating that the algorithm is better stabilized and does not iteratively swing with the increase in the number of neurons, and still has good clause satisfiability. Among these four models, the MTS model proposed in this paper has the best performance, attributed to the fact that the MTS metaheuristic algorithm begins the operation by selecting the optimal solution from 100 initial solutions. In contrast, other metaheuristic algorithms in this paper involve multiple solutions in each iteration, extensively exploring potential high-quality solutions across the solution space. This characteristic endows the MTS metaheuristic algorithm with higher retrieval efficiency. Additionally, leveraging the characteristics of metaheuristic algorithms [31], conducting local search based on high-

quality solutions enables the rapid discovery of consistent interpretations. Considering the structural characteristics of second-order logical clauses, reducing the MAE_{learn} value to find consistent interpretations necessitates increasing the number of satisfiable clauses. To increase the number of satisfiable clauses, flipping any literal in unsatisfiable clauses suffices. Based on this property, the mutation operator in the MTS algorithm is set to have a mutation probability of 1 for flipping clause literals. Simultaneously, segment operations are employed to ensure that the mutation operator operates on different clauses, directing the exploration of the local solution space. Consequently, the MTS algorithm exhibits superior stability and performance in terms of the MAE_{learn} indicator compared to other algorithms.

Tables 3 and 4 show the results of the experimental model with different proportions of positive literals (η) and number of neurons for $LRTR$. The values of the number of neurons were taken in consistency with the values of MAE_{learn}, and the optimal value of $LRTR$ was 1. In the table, Max, Min, and Mean represent the maximum, minimum, and intermediate values of $LRTR$ in the model, respectively. The larger these three values, the stronger the ability of the experimental model to find consistent explanations. It is worth noting that MAE_{learn} measures the ability of the experimental model to find consistent explanations in terms of clauses, and $LRTR$ measures the ability of the experimental model to find consistent explanations in terms of logic ($P_{PRO2SAT}$), which is more important than MAE_{learn}, because it maps the ability of the model to obtain the optimal salient weights. Std refers to the standard deviation, which reflects the degree of dispersion of the $LRTR$ dataset. A larger Std represents a larger difference between most of the L values and their means; a smaller Std means that these $LRTR$ values are closer to the mean, and the better the stability of the model. Avg Rank represents the average ranking of the Friedman test of the model with the same proportion of positive literals and number of neurons highlighted.

Table 3. The average $LRTR$ value of all proposed algorithms in the learning phase for $\eta = \{0.1, 0.2, 0.3, 0.4, 0.5\}$. The bold values indicate the superior result among the mentioned metrics.

η	Measure	ES	GA	ABC	EDA	PSO	ACO	DE	EA	SA	GWO	MTS
0.1	Max	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
	Min	0.000	**1.000**	0.000	0.690	0.000	0.850	0.000	**1.000**	0.000	0.220	**1.000**
	Mean	0.263	**1.000**	0.262	0.937	0.620	0.954	0.609	**1.000**	0.536	0.850	**1.000**
	Std	0.389	**0.000**	0.406	0.093	0.434	0.055	0.443	**0.000**	0.455	0.261	**0.000**
	Avg Rank	9.375	**3.667**	9.625	5.333	6.458	4.958	6.917	**3.667**	7.458	4.875	**3.667**
0.2	Max	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
	Min	0.000	**1.000**	0.000	0.780	0.000	0.800	0.000	**1.000**	0.000	0.040	**1.000**
	Mean	0.256	**1.000**	0.294	0.943	0.613	0.953	0.603	**1.000**	0.524	0.790	**1.000**
	Std	0.386	**0.000**	0.423	0.069	0.438	0.062	0.445	**0.000**	0.465	0.341	**0.000**
	Avg Rank	9.583	**3.625**	9.500	5.042	6.667	5.000	6.708	**3.625**	7.417	5.208	**3.625**
0.3	Max	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
	Min	0.000	**1.000**	0.000	0.780	0.100	0.870	0.000	**1.000**	0.000	0.050	**1.000**
	Mean	0.253	**1.000**	0.360	0.953	0.615	0.971	0.608	**1.000**	0.523	0.774	**1.000**
	Std	0.383	**0.000**	0.448	0.069	0.432	0.044	0.443	**0.000**	0.457	0.349	**0.000**
	Avg Rank	9.792	**3.625**	9.000	5.208	6.250	5.042	7.000	**3.625**	7.635	5.208	**3.625**
0.4	Max	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
	Min	0.000	**1.000**	0.000	0.74	0.000	0.81	0.000	**1.000**	0.000	0.040	**1.000**
	Mean	0.263	**1.000**	0.483	0.939	0.620	0.956	0.599	**1.000**	0.520	0.754	**1.000**
	Std	0.394	**0.000**	0.459	0.078	0.435	0.065	0.445	**0.000**	0.462	0.370	**0.000**
	Avg Rank	9.792	**3.583**	8.333	5.417	6.833	5.083	7.000	**3.583**	7.583	5.208	**3.583**
0.5	Max	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
	Min	0.000	**1.000**	0.820	0.830	0.000	0.880	0.000	**1.000**	0.000	0.030	**1.000**
	Mean	0.266	**1.000**	0.960	0.957	0.612	0.963	0.614	**1.000**	0.525	0.750	**1.000**
	Std	0.395	**0.000**	0.064	0.064	0.438	0.045	0.439	**0.000**	0.463	0.375	**0.000**
	Avg Rank	9.917	**3.833**	5.167	5.167	7.5	5.583	7.042	**3.833**	8.208	5.917	**3.833**

Table 4. The average *LRTR* value of all proposed algorithms in the learning phase for $\eta = \{0.6, 0.7, 0.8, 0.9\}$. The bold values indicate the superior result among the mentioned metrics.

η	Measure	ES	GA	ABC	EDA	PSO	ACO	DE	EA	SA	GWO	MTS
0.6	Max	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
	Min	0.000	**1.000**	**1.000**	0.770	0.010	0.820	0.000	**1.000**	0.000	0.050	**1.000**
	Mean	0.300	**1.000**	**1.000**	0.935	0.265	0.920	0.225	**1.000**	0.150	0.420	**1.000**
	Std	0.391	**0.000**	**0.000**	0.084	0.436	0.057	0.445	**0.000**	0.458	0.367	**0.000**
	Avg Rank	10.417	**4.042**	**4.042**	5.625	6.917	5.542	7.250	**4.042**	8.167	5.917	**4.042**
0.7	Max	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
	Min	0.000	**1.000**	**1.000**	0.740	0.000	0.83	0.000	**1.000**	0.000	0.070	**1.000**
	Mean	0.510	**1.000**	**1.000**	0.950	0.140	0.965	0.150	**1.000**	0.255	0.475	**1.000**
	Std	0.393	**0.000**	**0.000**	0.080	0.434	0.061	0.439	**0.000**	0.463	0.355	**0.000**
	Avg Rank	10.375	**4.083**	**4.083**	5.375	7.000	5.708	7.167	**4.083**	8.208	5.833	**4.083**
0.8	Max	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
	Min	0.000	**1.000**	**1.000**	0.740	0.000	0.800	0.000	**1.000**	0.000	0.160	**1.000**
	Mean	0.150	**1.000**	**1.000**	0.930	0.100	0.930	0.350	**1.000**	0.575	0.705	**1.000**
	Std	0.395	**0.000**	**0.000**	0.090	0.443	0.071	0.447	**0.000**	0.457	0.298	**0.000**
	Avg Rank	9.917	**4.000**	**4.000**	5.625	6.833	5.625	7.625	**4.000**	8.667	5.708	**4.000**
0.9	Max	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**
	Min	0.000	**1.000**	**1.000**	0.740	0.000	0.700	0.000	**1.000**	0.000	0.000	**1.000**
	Mean	0.170	**1.000**	**1.000**	0.895	0.300	0.945	0.200	**1.000**	0.225	0.400	**1.000**
	Std	0.388	**0.000**	**0.000**	0.086	0.430	0.082	0.445	**0.000**	0.462	0.341	**0.000**
	Avg Rank	9.875	**4.167**	**4.167**	5.500	7.333	5.792	7.208	**4.167**	8.167	5.458	**4.167**

(a) Experimental model for $\eta = \{0.1, 0.2, 0.3, 0.4, 0.5\}$

The optimal data in Table 3 are identified in bold. As described in the table, we know that the three models MTS, GA, and EA have the same performance and are the best among all models. The values of Max, Min, and Mean are 1, which means that *LRTR* is optimal for all values of neuron number, and the value of Std is 0, meaning that the models are absolutely stable in finding consistent explanations. Meanwhile, the values of Avg Rank for these three models are 3.667, 3.625, 3.583, and 3.833, respectively, which are the minimum among the experimental models under the corresponding values of positive literal occupancy. The results validate the advantages of the MTS, GA, and EA models in finding consistent interpretations.

(b) Experimental model for $\eta = \{0.6, 0.7, 0.8, 0.9\}$

The optimal data in Table 4 are identified in bold. As described in the table, MTS, GA, ABC, and EA perform well and are the most satisfactory among all models. The optimal value of *LRTR* is achieved for all values of the number of neurons. All of the values Max, Min, and Mean have a value of 1. The Std value of 0 means that the model is absolutely stable in finding consistent explanations. Similarly, the values of Avg Rank for these four models are 4.042, 4.083, 4.000, and 4.167, respectively, which are the minimum among the experimental models under the corresponding values of positive literals. The results validate the advantages of the MTS, GA, and EA models in finding consistent interpretations.

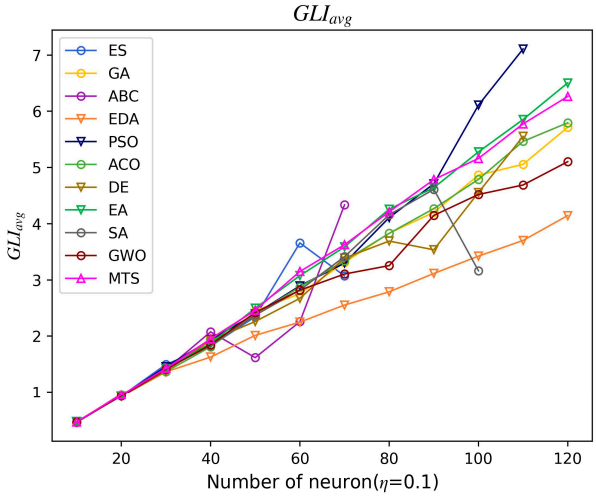
Combined with the results of (a) and (b), it can be observed that the ABC model performs poorly at $\eta = \{0.1, 0.2, 0.3, 0.4, 0.5\}$ and well at $\eta = \{0.6, 0.7, 0.8, 0.9\}$. The reason for this phenomenon is still the OR operator used in the generation of new nectar sources, and we have explained in the MAE_{learn} analysis section. In addition, the MTS, GA, and EA models optimize *LRTR* values for all positive literal proportion. The key to achieving the optimal value of *LRTR* lies in the ability of metaheuristic algorithms to escape the local optimum solution. The MTS, GA, and EA algorithms all possess good capabilities for escaping the local optimum solution. The MTS metaheuristic algorithm achieves this goal by utilizing the mutation operator, while the GA metaheuristic algorithm employs crossover and mutation operators to avoid premature convergence to local solutions. It

is worth noting that, due to the mutation probability of 0.01 in the GA metaheuristic algorithm, the crossover operator plays a major role. In contrast, the EA metaheuristic algorithm uses an advertising strategy to confine the search scope of solutions around the optimal solution and alternates between positive and negative advertisements to avoid premature convergence to the local optimum solution. Therefore, compared to other metaheuristic algorithms, these three metaheuristic algorithms all have useful strategies to avoid the local optimum solution, allowing them to explore unknown areas as much as possible. Even with a large number of neurons, they still possess good capabilities for finding consistent interpretations. However, acquiring this capability typically requires more iterations. Regarding the number of iterations, we will discuss this further in the *AI* indicator section.

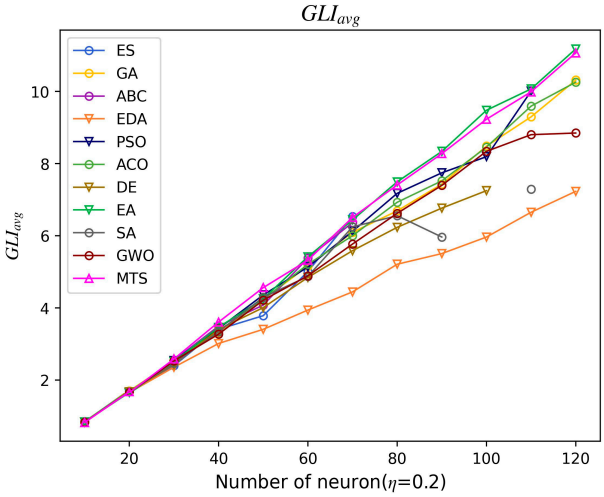
Figure 5 depicts the GLI_{avg} values obtained for the different experimental models. *GLI* measures the similarity of the solutions with respect to the benchmark state. For this case, the higher the GLI_{avg} value is, the more favorable it is to study the similarity index, indicating that the smaller the difference between the solution found by the metaheuristic algorithm and the standard solution, the better the ability of the model to obtain high-quality solutions. In our analysis, it is easier to obtain solutions for all experimental models when $NN \leq 60$. On the contrary, when $NN \geq 60$, the six models ES, ABC, SA, DE, PSO, and GWO are unstable in their solving ability. These models will have null values of GLI_{avg} when the models are trapped in the local optimal solution. In addition, the average similarity index values of the solutions of the five models including MTS, GA, EDA, ACO, and EA do not have null values and increase linearly with the number of neurons, indicating that these models can search for solutions in the learning phase and the solution diversity is greater with the increase in the number of neurons. The diversity of solutions is greater as the number of neurons increases. Among them, the EDA model has the lowest solution quality, which is due to the EDA algorithm showing the selection rate of 0.1, with the statistical fitness of the largest 10 solutions to obtain the probability of the neuron state taking values. This method to obtain the neuron state is guided by these 10 maximum fitness solutions, indicating that the arithmetic exploration ability is weaker. It can also be seen that the MTS and EA models are the best performers among all the models. To observe the results of these two models more clearly, we used the matrix diagram to show their specific results, as shown in Tables 5 and 6.

Table 5. Matrix diagram of GLI_{avg} for MTS and EA models. The bold value indicates the superior result among the mentioned metrics.

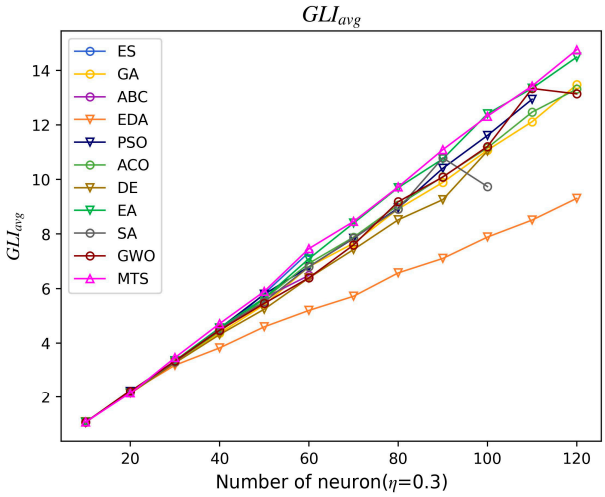
η	Algorithm	10	20	30	40	50	60	70	80	90	100	110	120
0.1	MTS	0.457	**0.946**	**1.426**	**1.949**	2.462	**3.150**	**3.626**	4.212	**4.785**	5.161	5.776	6.242
	EA	0.466	0.919	1.422	1.873	2.498	3.076	3.588	4.259	4.641	5.278	5.855	6.505
0.2	MTS	0.800	**1.652**	**2.586**	**3.610**	**4.564**	5.324	**6.510**	7.409	8.278	9.232	9.991	11.074
	EA	0.814	1.629	2.514	3.465	4.257	5.418	6.439	7.497	8.339	9.481	10.072	11.181
0.3	MTS	**1.072**	2.146	**3.464**	**4.713**	**5.894**	**7.462**	**8.457**	**9.725**	**11.091**	12.319	**13.426**	**14.755**
	EA	1.062	2.175	3.352	4.556	5.648	7.089	8.397	9.698	10.752	12.402	13.340	14.484
0.4	MTS	**1.198**	2.447	**3.876**	**5.284**	**6.736**	**8.285**	**9.791**	**11.122**	12.610	13.891	**15.477**	16.744
	EA	1.197	2.512	3.834	5.201	6.688	8.112	9.697	10.942	12.663	13.986	15.338	16.892
0.5	MTS	**1.252**	**2.590**	**4.020**	**5.611**	**7.098**	**8.682**	**10.061**	**11.571**	**13.080**	**14.564**	15.870	17.312
	EA	1.241	2.561	3.902	5.211	6.944	8.421	9.791	11.501	12.936	14.555	16.169	17.662
0.6	MTS	**1.198**	**2.506**	**3.857**	**5.316**	**6.754**	**8.175**	**9.725**	**11.160**	**12.761**	**13.992**	15.323	16.634
	EA	1.184	2.443	3.841	5.249	6.667	8.058	9.690	11.122	12.723	13.981	15.630	16.839
0.7	MTS	**1.062**	**2.191**	**3.400**	**4.520**	**6.032**	**7.248**	8.420	**9.775**	11.084	**12.265**	**13.502**	14.629
	EA	1.056	2.163	3.328	4.463	5.692	7.057	8.533	9.633	11.107	12.203	13.448	14.723
0.8	MTS	**0.816**	**1.691**	**2.586**	**3.518**	4.432	**5.670**	**6.519**	**7.476**	8.221	9.145	**10.194**	11.138
	EA	0.808	1.686	2.471	3.496	4.451	5.379	6.319	7.407	8.237	9.250	10.140	11.228
0.9	MTS	**0.458**	0.918	1.390	**2.068**	**2.721**	**3.325**	**3.759**	**4.228**	**4.708**	**5.339**	5.661	6.138
	EA	0.455	0.957	1.423	1.862	2.401	2.926	3.649	4.218	4.554	5.244	5.814	6.249



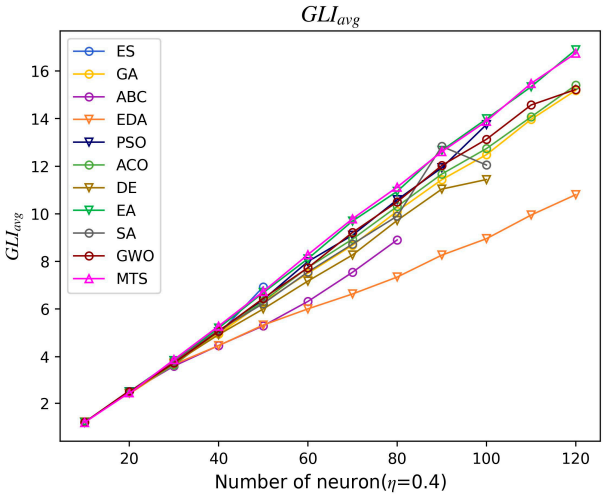
(**a**) GLI_{avg} evaluation when $\eta = 0.1$



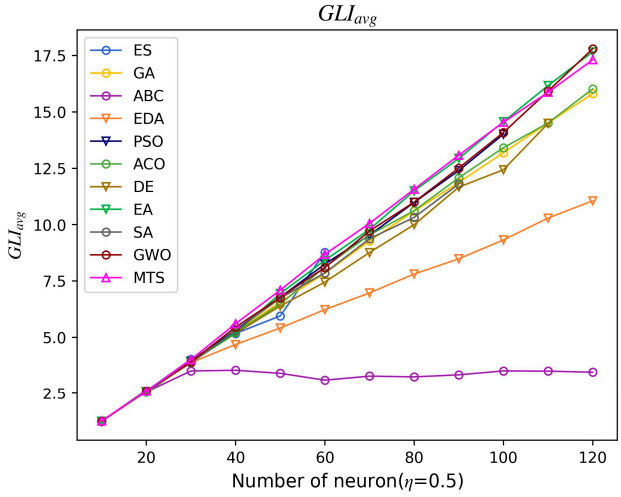
(**b**) GLI_{avg} evaluation when $\eta = 0.2$



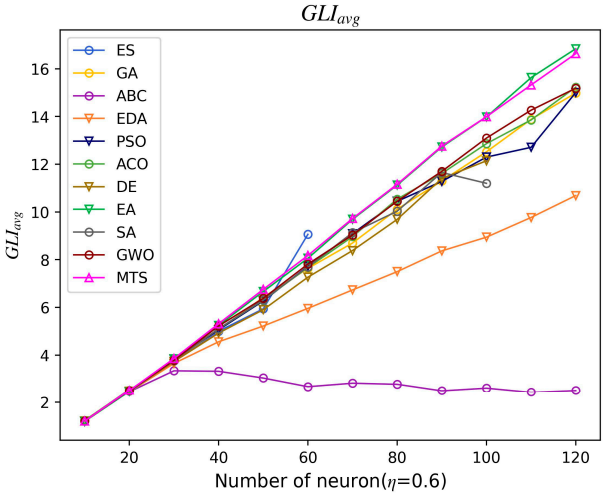
(**c**) GLI_{avg} evaluation when $\eta = 0.3$



(**d**) GLI_{avg} evaluation when $\eta = 0.4$

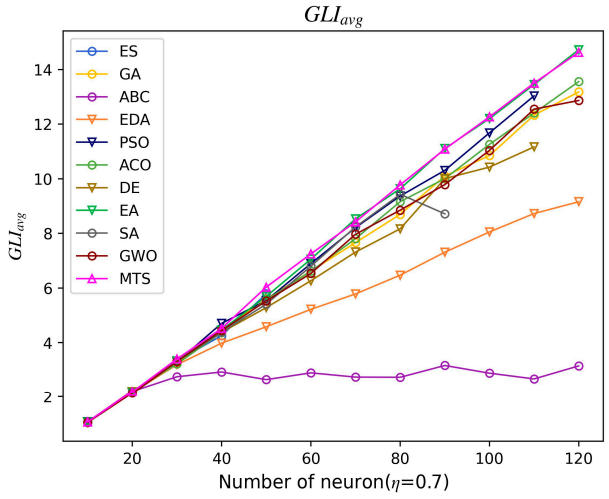


(**e**) GLI_{avg} evaluation when $\eta = 0.5$

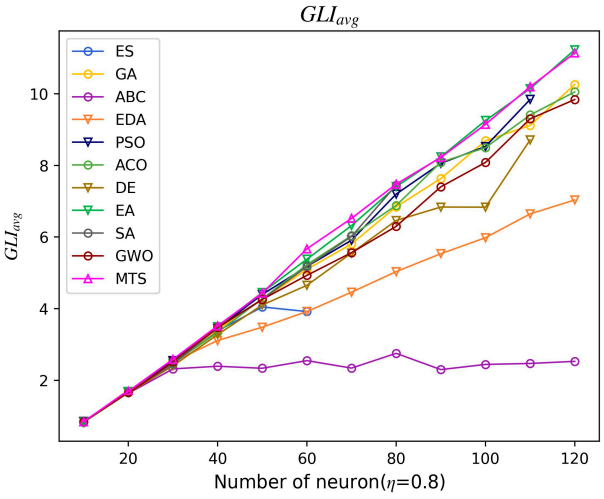


(**f**) GLI_{avg} evaluation when $\eta = 0.6$

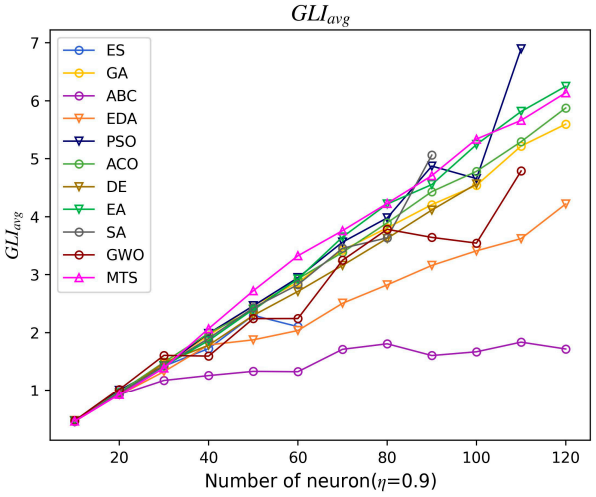
Figure 5. *Cont.*



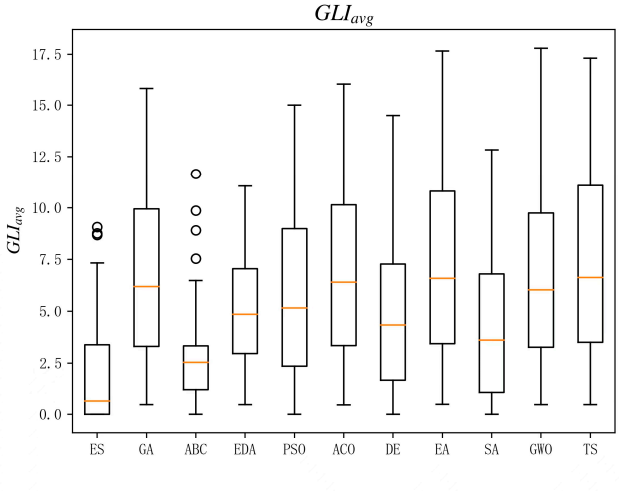
(**g**) GLI_{avg} evaluation when $\eta = 0.7$



(**h**) GLI_{avg} evaluation when $\eta = 0.8$



(**i**) GLI_{avg} evaluation when $\eta = 0.9$



(**j**) Box diagram of GLI_{avg}

Figure 5. GLI_{avg} evaluation for all experimental models.

Table 6. Matrix representing the frequency of the maximum GLI_{avg}. The bold value indicates the superior result among the mentioned metrics.

Model	10	20	30	40	50	60	70	80	90	100	110	120	Probability
MTS	7	6	8	9	7	8	8	7	5	4	4	1	**69%**
EA	2	3	1	0	2	1	1	2	4	5	5	8	31%

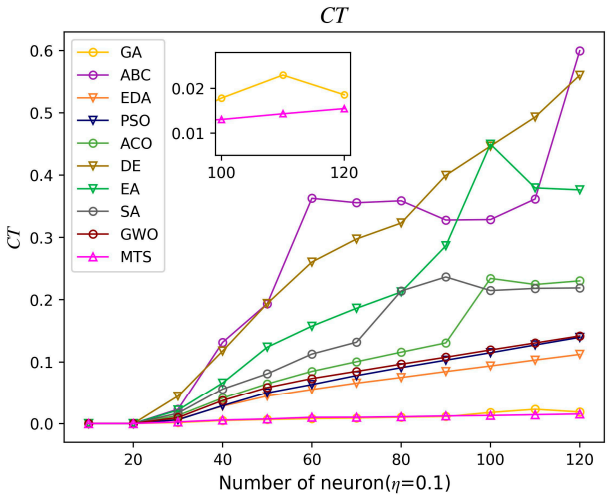
We plotted the box plots summarizing the GLI_{avg} data for each model $\eta = \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9\}$, as shown in (j) in Figure 5, from which we can see that the box of the MTS model is longer, indicating that the data are more concentrated, and the median of the MTS model is the highest compared to all experimental models, indicating that the model solution has the best quality. In additionally, the ES model and ABC model GLI_{avg} values are biased and the data are more discrete, indicating that the model-solving ability is not stable.

Table 5 depicts the GLI_{avg} values obtained for the MTS and EA models with different positive literal proportions. The best values are expressed in bold following the comparison between MTS and EA. It can be seen that the MTS metaheuristic algorithm has more

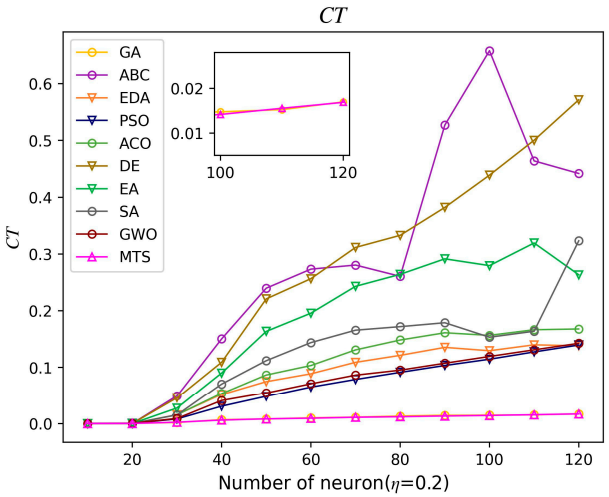
obvious advantages when $NN \leq 110$, and the EA algorithm performs even better when the neuron value is equal to 120. Note that Table 6 counts the number of times that the model obtains the maximum GLI_{avg} value, and the chance that MTS obtains the optimal GLI_{avg} value is 69%. Thus, it can be seen that the quality of solutions obtained by the MTS metaheuristic algorithm is overall higher than that of the EA metaheuristic algorithm. This is because, in the EA algorithm, each voter may be influenced by the assimilation of the party leader they support (positive advertisement) or by the assimilation of other party leaders (negative advertisement), leading to frequent superposition operations. In the worst-case scenario, a voter may be subject to assimilation operations from all party leaders, resulting in unstable changes in party membership direction and consequently causing larger differences between generated solutions and standard solutions. In contrast, the MTS algorithm's process of finding the optimal solution is more direct and explicit. Firstly, by using segment operations and the mutation operator to flip the states of four neurons, new solutions are directly generated, avoiding repetitive superposition operations and ensuring the stability of solution generation. Secondly, the use of a tabu table as short-term "memory" avoids using operation segments stored in the tabu table in the next iterations, thereby avoiding repetitive searches in the short term and providing exploration directions for solution generation, thus yielding high-quality solutions. In summary, the MTS metaheuristic algorithm demonstrates superior solution quality among all experimental models.

In this paper, we used two metrics, *CT* and *AI*, to measure the efficiency of all experimental models in the retrieval phase. An efficient and stable model can be run in a short time, i.e., the value of both metrics is small. *CT* is the average time to execute the metaheuristic algorithm in 100 logical combination ($P_{PRO2SAT}$). The values of these two metrics are demonstrated for the 10 experimental models. It is worth mentioning that the absence of data for the ES model in Figure 6 is due to the fact that the ES algorithm does not run a metaheuristic algorithm.

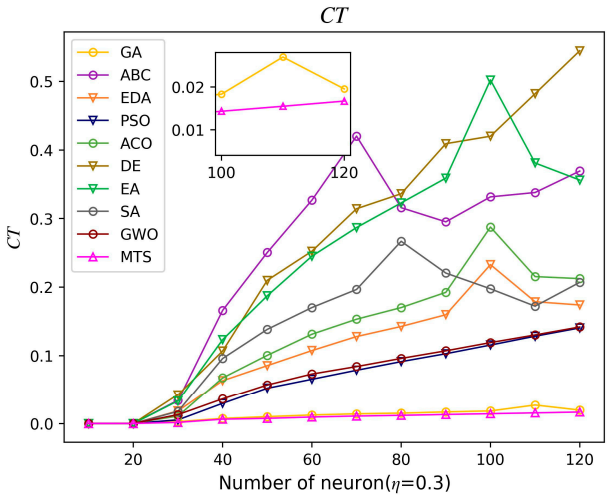
From Figure 6, it can be known that all experimental models have a gradual increase in the *CT* value with the increase in the number of neurons. Among them, the DE model takes the most time among all experimental models for one iteration and changes unstably as the number of neurons increases, because the optimal solution in the learning phase is not a definite combination of neurons. The DE algorithm randomly selects three fathers to generate the new generation solution, and the quality of the new generation solution depends on the quality of the parent fitness. Random fathers leads the DE algorithm to easily stay in the local optimal solution and be unable to effectively find the global optimal solution [33]. Furthermore, the DE algorithm requires a longer computation time per iteration [34]. This is because it is the only model among all others that performs mutation and crossover calculations for each neuron of every generation of solutions. Therefore, the DE algorithm necessitates more iterations and computation time for solving. Secondly, we can see from the figure that the *CT* values of the models GA and MTS are the smallest and basically overlap. As the number of neurons increases, their *CT* values basically remain as a straight line with a very small increase, indicating that these two algorithms have good computational scalability. Similarly, as the number of neurons increases, the computation time of the algorithms can remain basically constant without much impact on the computation time. After zooming on the result locally, we can clearly observe that the *CT* value of MTS is the smallest among the metaheuristics, and the algorithm time has a much lower complexity.



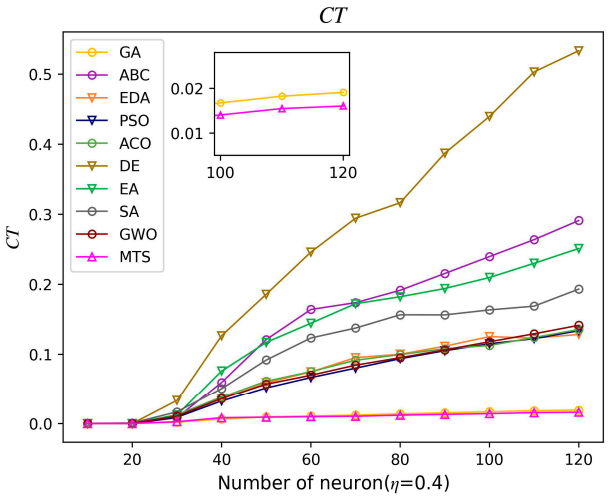
(**a**) CT evaluation when $\eta = 0.1$



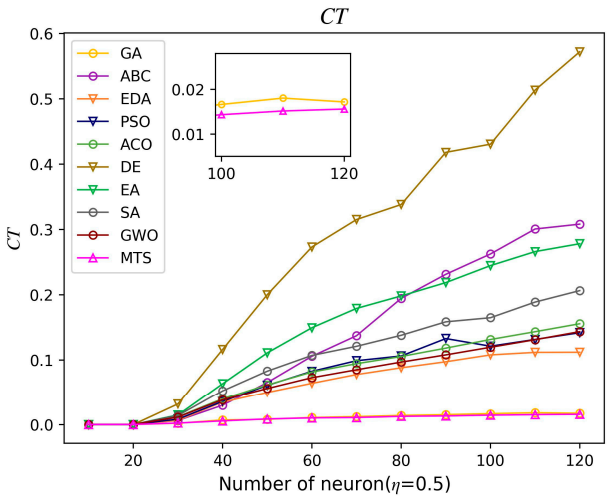
(**b**) CT evaluation when $\eta = 0.2$



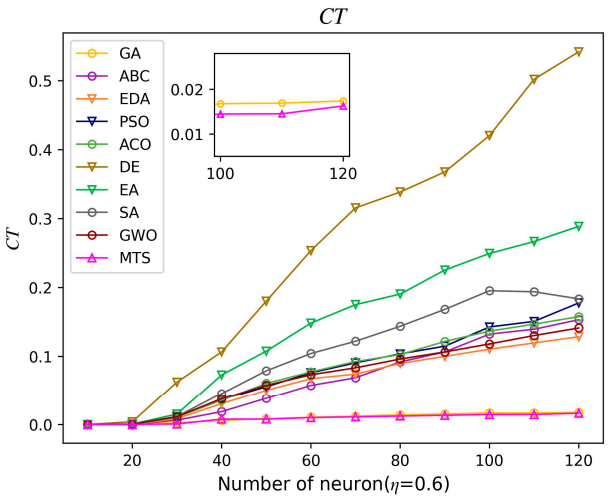
(**c**) CT evaluation when $\eta = 0.3$



(**d**) CT evaluation when $\eta = 0.4$

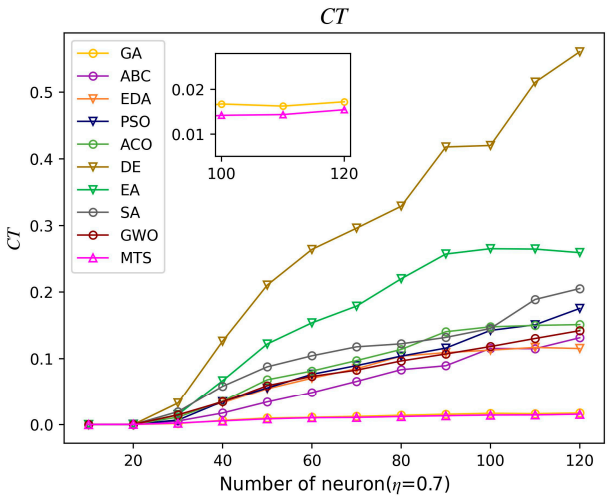


(**e**) CT evaluation when $\eta = 0.5$

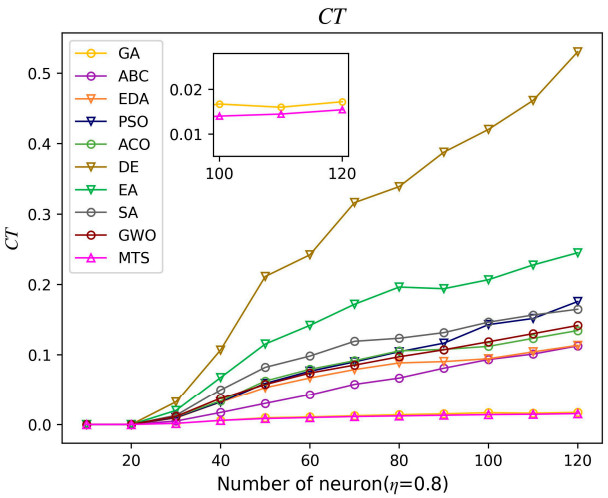


(**f**) CT evaluation when $\eta = 0.6$

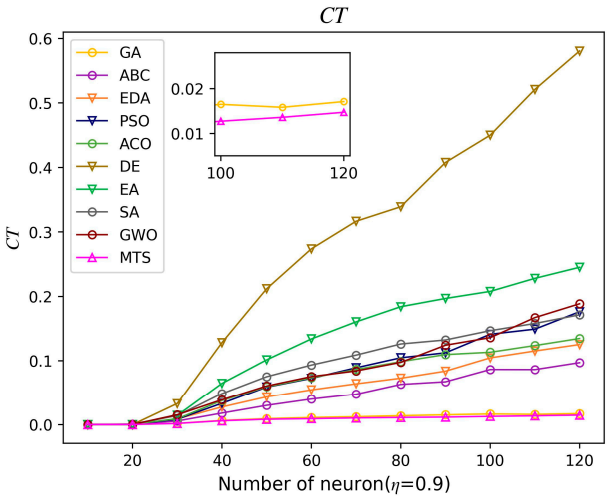
Figure 6. *Cont.*



(**g**) CT evaluation when $\eta = 0.7$



(**h**) CT evaluation when $\eta = 0.8$



(**i**) CT evaluation when $\eta = 0.9$

Figure 6. CT evaluation for all PRO2SAT embedded in metaheuristic algorithms.

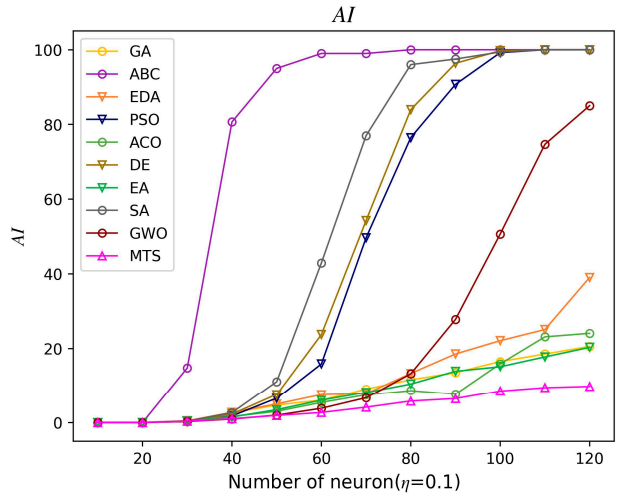
We will further compare the time complexity of GA and MTS algorithms to more intuitively assess their performance. The calculation mainly considers the time complexity of the major steps in one iteration of the algorithm under the worst-case scenario, where c represents the number for learning, n represents the number of offspring chromosomes, and m represents the number of excellent parent chromosomes in each iteration process. In this experiment, the values are set to 50 [11]. NN represents the number of neurons, and w represents the number of domain solutions, with a value of 14 as introduced in Table 2. The time complexity of the GA and MTS algorithms is represented as GA_{TC} and MTS_{TC}, respectively. For a more intuitive comparison of algorithm time complexity, we set n and w to the same value. By comparing GA_{TC} and MTS_{TC}, it can be observed that MTS_{TC} has a smaller time complexity. The advantage of the MTS algorithm mainly manifests in the aspect of the fitness evaluation. This advantage stems from the feedback correction mechanism implemented in the MTS algorithm during computation, specifically the guided search for the highest-quality solution using a tabu list. In contrast, the GA algorithm generates multiple offspring chromosomes in each iteration, leading to increased computational complexity in evaluating the fitness of the next generation, thus consuming more time. Additionally, the mutation operation in the GA algorithm requires traversing every gene in all chromosomes, directly resulting in a more significant time gap compared

to the MTS algorithm. Therefore, the MTS algorithm exhibits higher search efficiency and performance, especially in terms of fitness evaluation.

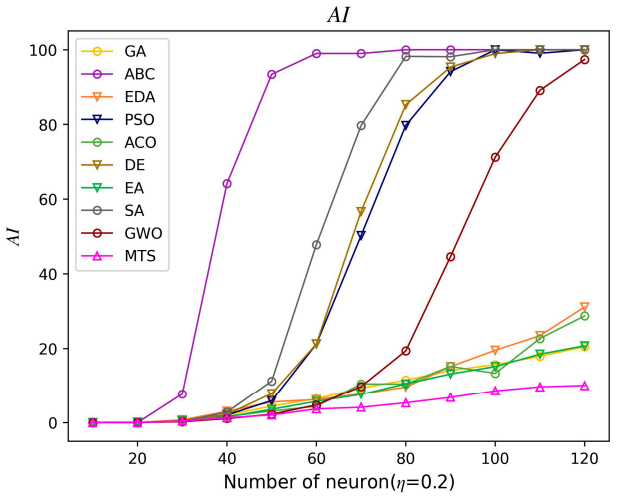
At the same time, we can also see that the *CT* value visibly varies as the ABC model changes with the proportion value of the positive literal. The *CT* value is larger when $\eta \leq 0.4$. However, as the positive literal share increases $\eta \geq 0.5$, it becomes smaller, but it still larger than the GA and MTS models. This is because, when the proportion of the positive literals proportion is small, the ABC metaheuristic algorithm requires many iterations and even reaches the maximum iteration limit. When the proportion of positive literals is large, the ABC algorithm usually converges in one to two iterations. From this, we know that the MTS model in this paper is more efficient and stable, and the value remains stable as the number of neurons increases.

In this experiment, Figure 7a–i show the convergence trend to all experimental models, and the number of iterations required for convergence of the experimental models mostly tends to increase as the number of neurons increases. It is worth noting that, when $NN \leq 30$, $C_{PRO2SAT} = 0$ can be satisfied among the 100 initial solutions of PRO2SAT. At this point, no metaheuristic algorithm is executed and the *AI* value of the experimental model is mostly 0. From the figure, we can divide the experimental models into two categories. The first category "parameter-sensitive models", which are influenced by the number of neurons, including SA, DE, PSO, GWO. These four models are influenced by the proportion of positive literals. The second category of "stable models", where the number of iterations is less influenced by the number of neurons and the proportion of positive literals, including EDA, ACO, EA, GA, and MTS. The part of the parameter-sensitive model that is influenced by the number of neurons falls into a local solution after a certain number of iterations and is unable to jump out of the local solution. The convergence of the ABC model also significantly fluctuates depending on the proportion of positive literals. This convergence bias of the ABC metaheuristic algorithm is not suitable for PRO2SAT, because PRO2SAT includes nine positive literal proportion settings, and the desired metaheuristic algorithm is meant to satisfy all occupancy proportions of the positive literals of PRO2SAT.

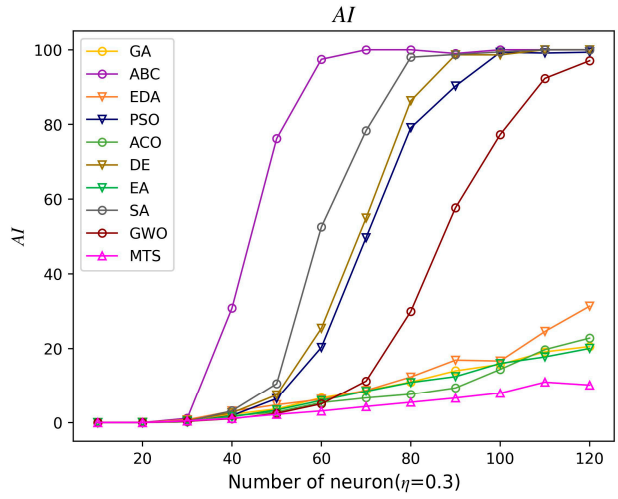
Stable metaheuristic algorithms are required, in which the most outstanding performance is the MTS algorithm proposed in this paper, which has the fastest convergence speed and is equally stable. When $NN = 120$, $9.8 \leq AI \leq 11.26$, the consistency interpretation can be found by calculating an average of 10.48 iterations of the MTS algorithm for a neuron value of 120. This is because the MTS algorithm constructs a search feedback strategy, which is specifically manifested as short-term "memory" and long-term "memory". Firstly, based on the tabu table to achieve short-term "memory", the tabu table is used to record the recently accessed segment numbers, enabling adaptive adjustment of the search direction when selecting segments, actively choosing segments that have not been operated on recently to facilitate a faster solution search. Secondly, based on G_{best} to achieve long-term "memory", G_{best} stores the solutions with the highest fitness in each iteration; combined with the flouting rule, when the local solution is greater than the global solution ($X_{best} > G_{best}$), the excellent domain operations banned by the tabu are pardoned, allowing repeated access to segment numbers that were recently accessed, thereby ensuring diverse and effective exploration and avoiding missing better solutions. Short-term "memory" functions within each iteration, while long-term "memory" functions between iterations, enabling the MTS algorithm to possess intelligent search capabilities throughout the entire process. In contrast, other comparative models only calculate fitness between each iteration to determine if a solution has been found, lacking effective search guidance. Thus, the MTS metaheuristic algorithm achieves a more balanced exploration and exploitation of the solution space, exhibiting efficient and stable characteristics.



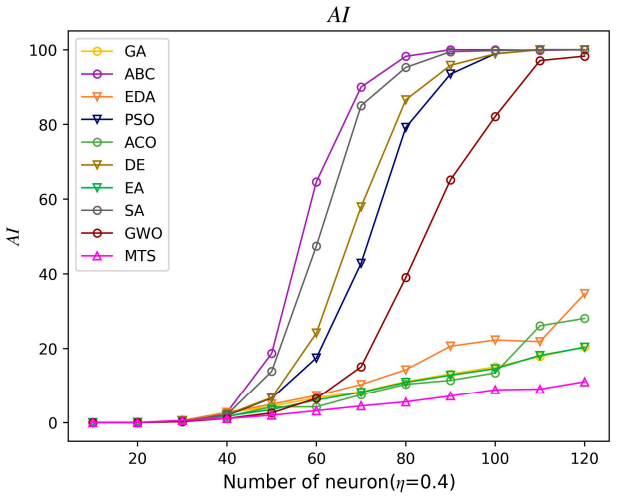
(**a**) AI evaluation when $\eta = 0.1$



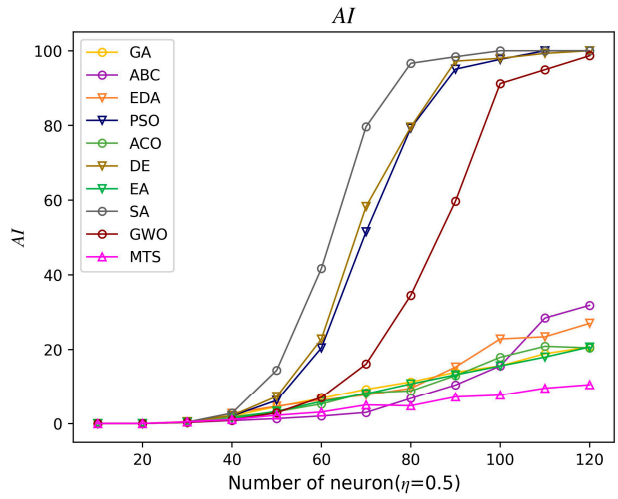
(**b**) AI evaluation when $\eta = 0.2$



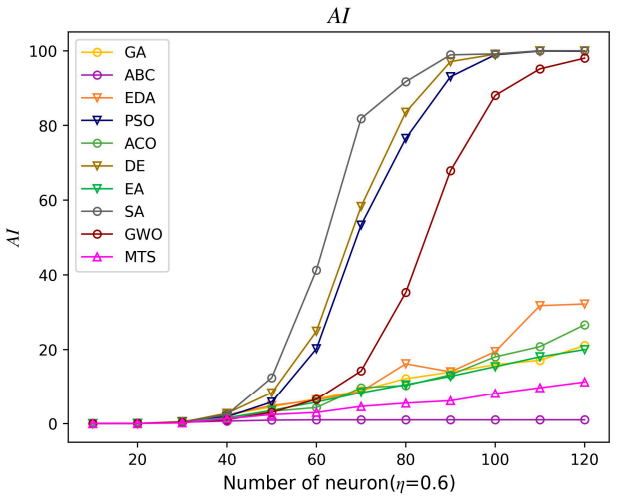
(**c**) AI evaluation when $\eta = 0.3$



(**d**) AI evaluation when $\eta = 0.4$

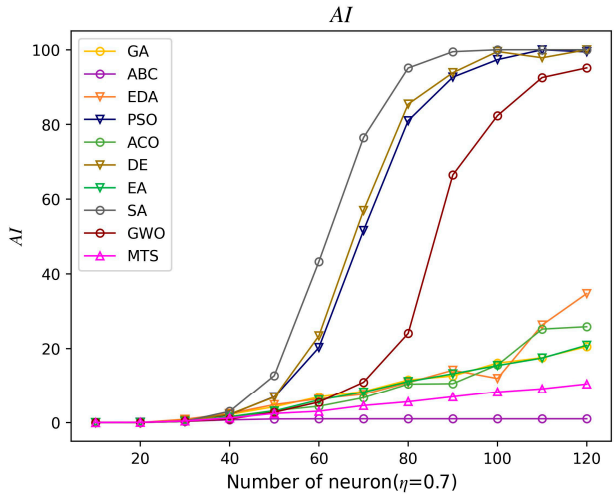


(**e**) AI evaluation when $\eta = 0.5$

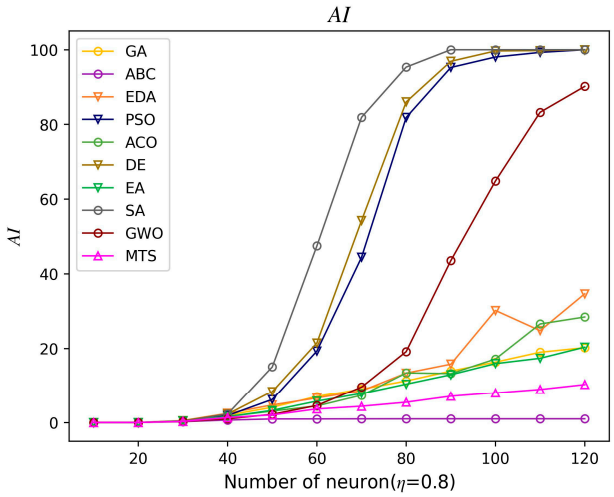


(**f**) AI evaluation when $\eta = 0.6$

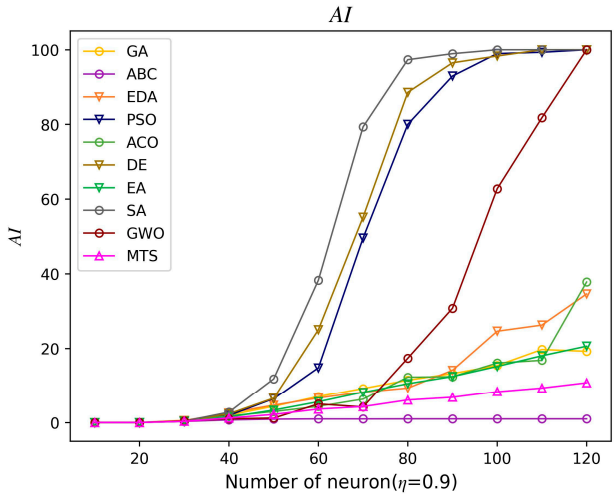
Figure 7. *Cont.*



(g) AI evaluation when $\eta = 0.7$



(h) AI evaluation when $\eta = 0.8$



(i) AI evaluation when $\eta = 0.9$

Figure 7. AI evaluation for all PRO2SAT embedded in metaheuristic algorithms.

In summary, for the performance analysis of all experimental models in the learning phase, the MTS model proposed in this paper has three advantages in the learning phase. Firstly, it has the strongest ability to find consistent explanations. Secondly, the solutions found have high quality and the highest diversity. Finally, it has the highest efficiency in finding consistent explanations, that is, the iteration time and the number of iterations is minimized. Therefore, the MTS model is the best among all experimental models when examined comprehensively.

8.2. Retrieval Phase

ZM is the core metric of the experimental model in the retrieval phase. Tables 7 and 8 show the results of the ZM metric for 11 experimental models with different proportions of positive literal (η) and number of neurons. The ideal value for ZM is 1. A higher value for this metric indicates a greater presence of global minimum solutions within the solution set and a reduced number of local minimum solutions. Note that the data following the '/' symbol in the table represent the number of neurons at the time the value was obtained. It is worth noting that the experimental models in this paper start the metaheuristic algorithm only when they cannot find consistent explanations within the default learning (a). All models will not start the metaheuristic algorithm for PRO2SAT at $NN = \{10, 20\}$.

Table 7. The time complexity of the GA and MTS algorithms.

Index	GA	Time Complexity	MTS	Time Complexity
1	Selection operation	$O(clogc)$	Initializing the initial solutions	$O(clogc)$
2	Crossover operation	$O(m^2 + nlogn)$	Generation strategy to neighborhood solution	$O(w \times NN/2 + wlogw + w)$
3	Mutation operation	$O(n \times NN)$	Generation strategy to candidate solution	$O(2)$
4	Fitness evaluation	$O(n \times NN)$	Fitness evaluation	$O(NN/2)$
collect	$GA_{TC} = O(clogc + w \times NN/2 + wlogw + m^2 + w \times NN)$		$MTS_{TC} = O(clogc + w \times NN/2 + wlogw + w + 2 + NN/2)$	

Table 8. The average ZM value of all proposed algorithms during the retrieval phase at $\eta = \{0.1, 0.2, 0.3, 0.4, 0.5\}$. The bold value indicates the superior result among the mentioned metrics.

η	Measure	ES	GA	ABC	EDA	PSO	ACO	DE	EA	SA	GWO	MTS
0.1	Max	1.000 /20	**1.000**	1.000 /20	1.000 /70	1.000 /60	1.000 /60	1.000 /60	**1.000**	1.000 /50	1.000 /70	**1.000**
	Min	0.000 /70	**1.000**	0.000 /80	0.690 /120	0.000 /120	0.780 /120	0.000 /120	**1.000**	0.000 /110	0.180 /120	**1.000**
	Mean	0.262	**1.000**	0.262	0.937	0.620	0.954	0.609	**1.000**	0.536	0.850	**1.000**
	Std	0.405	**0.000**	0.424	0.097	0.445	0.075	0.467	**0.000**	0.479	0.282	**0.000**
	Avg Rank	9.540	**3.670**	9.620	5.330	6.370	4.830	6.540	**3.670**	7.580	5.170	**3.670**
0.2	Max	1.000 /20	**1.000**	1.000 /20	1.000 /60	1.000 /60	1.000 /60	1.000 /60	**1.000**	1.000 /50	1.000 /70	**1.000**
	Min	0.000 /80	**1.000**	0.000 /80	0.780 /120	0.000 /120	0.800 /120	0.000 /110	**1.000**	0.000 /120	0.040 /120	**1.000**
	Mean	0.256	**1.000**	0.294	0.943	0.613	0.953	0.603	**1.000**	0.524	0.790	**1.000**
	Std	0.403	**0.000**	0.442	0.087	0.458	0.065	0.465	**0.000**	0.485	0.356	**0.000**
	Avg Rank	9.580	**3.620**	9.500	5.000	6.710	5.000	6.750	**3.620**	7.460	5.120	**3.620**
0.3	Max	1.000 /20	**1.000**	1.000 /30	1.000 /60	1.000 /60	1.000 /50	1.000 /50	**1.000**	1.000 /50	1.000 /70	**1.000**
	Min	0.000 /90	**1.000**	0.000 /100	0.780 /120	0.010 /120	0.870 /120	0.000 /110	**1.000**	0.000 /110	0.050 /120	**1.000**
	Mean	0.255	**1.000**	0.360	0.953	0.615	0.971	0.608	**1.000**	0.523	0.774	**1.000**
	Std	0.401	**0.000**	0.468	0.719	0.451	0.046	0.462	**0.000**	0.477	0.365	**0.000**
	Avg Rank	9.750	**3.620**	9.040	5.210	6.250	5.040	7.000	**3.620**	7.620	5.210	**3.620**
0.4	Max	1.000 /20	**1.000**	1.000 /40	1.000 /50	1.000 /50	1.000 /60	1.000 /50	**1.000**	1.000 /50	1.000 /70	**1.000**
	Min	0.000 /80	**1.000**	0.000 /120	0.740 /120	0.000 /110	0.810 /120	0.000 /110	**1.000**	0.000 /110	0.040 /120	**1.000**
	Mean	0.264	**1.000**	0.483	0.939	0.538	0.956	0.517	**1.000**	0.520	0.754	**1.000**
	Std	0.412	**0.000**	0.479	0.082	0.471	0.068	0.477	**0.000**	0.483	0.386	**0.000**
	Avg Rank	9.710	**3.580**	8.170	5.330	7.080	5.080	7.250	**3.580**	7.420	5.210	**3.580**
0.5	Max	1.000 /20	**1.000**	1.000 /70	1.000 /70	1.000 /50	1.000 /50	1.000 /60	**1.000**	1.000 /50	1.000 /70	**1.000**
	Min	0.000 /70	**1.000**	0.820 /120	0.830 /120	0.000 /110	0.890 /120	0.000 /120	**1.000**	0.000 /100	0.030 /120	**1.000**
	Mean	0.267	**1.000**	0.960	0.957	0.612	0.963	0.614	**1.000**	0.525	0.750	**1.000**
	Std	0.413	**0.000**	0.665	0.669	0.458	0.047	0.459	**0.000**	0.483	0.392	**0.000**
	Avg Rank	9.920	**3.830**	5.170	5.170	7.500	5.580	7.040	**3.830**	8.210	5.920	**3.830**

The optimal data in Table 8 are marked in bold. We analyzed and classified the experimental models into three different classes. The first rank models include GA, EA, and MTS, which reaches the optimal value of ZM under all neuron takes, i.e., the value of ZM is always 1. In the table, their Max, Min, and Mean values are all 1, which means that these three models can always obtain the global minimum solution. In addition, their standard deviation values are all 0, indicating that these three models have absolute stability and are not affected by the change in the number of neurons in the experiments. Further observing the average rank values of the first rank models, it can be found that their Ava

Rank values are equal and the minimum among the results under the condition of the same proportion of positive literals. All the results show the superiority of these three models in *ZM* metrics. The second-level models include ACO, EDA, and GWO. The second-level models have a slightly inferior performance compared to first-level models, which is mainly reflected in their inability to obtain the optimal value of ZM as the number of neurons increases. The third-level models include ES, ABC, PSO, DE, and SA, which have poorer performance, and the *ZM* value will gradually take 0 as the number of neurons increases. Notably, it is important to highlight that the results presented in this paper differ from the conclusions reported in the literature [8]. In the literature [8], researchers have found that the *ZM* value of the PRO2SAT model increases as the proportion of positive literals increases. This is because traditional activation functions tend to lead the final neuron states toward suboptimal states, which may likely converge to local minimum energy, But such a phenomenon is not observed in our experiment. This difference can be attributed to the new activation function used in this paper. When the value of Equation (11) is taken as 0, the final state of the neuron in Equation (12) remains the same as the initial state. The new activation function leads to the correct update of the final neuron states in clauses containing negated literals, thereby converging to the global minimum energy, resulting in higher *ZM* values.

Based on the optimal data identified in bold in Table 9, we performed a comprehensive analysis according to the same model hierarchical classification method as in Table 7. Notably, The ABC model is elevated from the third to the first rank, while the *ZM* metrics data of the other models present the same performance as in Table 7, with no change in rank attribution. Despite the ABC model's stronger performance seen in Table 8, this merely highlights its superiority in situations wherein there is a high proportion of positive literals. Nevertheless, with a low proportion of positive literals, the model's *ZM* performance deteriorates, indicating its instability. Overall, only three metaheuristic algorithms, namely GA, EA, and MTS, with LRTR taking the max value of 1 for different proportion values of positive literals and number of neurons, were able to provide optimal synaptic weights for the retrieval phase. Under the guidance of the optimal synaptic weights and the new activation function, the three models GA, EA, and MTS show excellent performance in obtaining the global minimum solution. Consequently, we analyzed the quality of the global minimum solution obtained by the models using the new metrics $TV_{similar}$.

Figure 8 illustrates the $TV_{similar}$ metric, which is used to analyze the quality of the solutions generated during the retrieval phase. Accordingly, we calculated the similarity between the non-repeated solution and the benchmark solution. If the solution is less different from the benchmark solution (the larger the $TV_{similar}$ value is), the resultant solution has poorer variability and diversity, and vice versa, the resultant solution has better variability and diversity. Firstly, the linear trend of $TV_{similar}$ values for all experimental models is due to the logical stable clause structure of the system. Furthermore, the ES, ABC, PSO, DE, and SA models take the $TV_{similar}$ value as null in the case of neurons that are available. We zoomed in on these plots and observed that the line plots of these models are suddenly interrupted. This is because the value of *ZM* affects the $TV_{similar}$ value obtained by the final neuron states, as only the final neuron states reaching the global minimum solution are used to calculate $TV_{similar}$. Finally, the distribution of the $TV_{similar}$ value in the MTS model is around 0.58, and the distribution of this value of all the remaining comparison models is around 0.75, meaning that the MTS model outperforms among all the experimental models in terms of solving variability and solution diversity due to the fact that the MTS model has embedded a mutation operator when generating the solution in the test phase. Otherwise, under the influence of the new activation function, the final neuron states of all models may tend to converge to the benchmark solution. The mutation operator during the retrieval phase is responsible for searching for clauses that match the benchmark solution after the local field computation of neuron strings and randomly flipping neuron states. This operation enhances the diversity of model solutions while maintaining the same number of global minimum solutions.

Table 9. The average ZM value of all proposed algorithms during the retrieval phase at $\eta = \{0.6, 0.7, 0.8, 0.9\}$. The bold value indicates the superior result among the mentioned metrics.

η	Measure	ES	GA	ABC	EDA	PSO	ACO	DE	EA	SA	GWO	MTS
0.6	Max	1.000 /10	1.000	**1.000**	1.000 /60	1.000 /60	1.000 /60	1.000 /60	**1.000**	1.000 /50	1.000 /50	**1.000**
	Min	0.000 /70	1.000	**1.000**	0.770 /120	0.010 /120	0.820 /120	0.000 /100	**1.000**	0.000 /110	0.050 /120	**1.000**
	Mean	0.258	1.000	**1.000**	0.942	0.611	0.955	0.607	**1.000**	0.529	0.746	**1.000**
	Std	0.409	0.000	**0.000**	0.874	0.456	0.060	0.465	**0.000**	0.479	0.383	**0.000**
	Avg Rank	10.420	4.040	**4.040**	5.620	6.920	5.540	7.080	**4.040**	8.170	5.920	**4.040**
0.7	Max	1.000 /10	**1.000**	**1.000**	1.000 /70	1.000 /60	1.000 /60	1.000 /60	**1.000**	1.000 /50	1.000 /70	**1.000**
	Min	0.000 /60	**1.000**	**1.000**	0.740 /120	0.020 /120	0.830 /120	0.000 /120	**1.000**	0.000 /100	0.070 /100	**1.000**
	Mean	0.254	**1.000**	**1.000**	0.957	0.619	0.959	0.608	**1.000**	0.536	0.761	**1.000**
	Std	0.411	**0.000**	**0.000**	0.084	0.453	0.064	0.458	**0.000**	0.483	0.371	**0.000**
	Avg Rank	10.370	**4.080**	**4.080**	5.370	7.000	5.710	7.170	**4.080**	8.210	5.830	**4.080**
0.8	Max	1.000 /20	**1.000**	**1.000**	1.000 /60	1.000 /60	1.000 /60	1.000 /50	**1.000**	1.000 /40	1.000 /70	**1.000**
	Min	0.000 /70	**1.000**	**1.000**	0.740 /120	0.000 /120	0.800 /120	0.000 /120	**1.000**	0.000 /90	0.160 /120	**1.000**
	Mean	0.259	**1.000**	**1.000**	0.937	0.618	0.948	0.603	**1.000**	0.520	0.818	**1.000**
	Std	0.413	**0.000**	**0.000**	0.094	0.463	0.074	0.467	**0.000**	0.478	0.311	**0.000**
	Avg Rank	9.920	**4.000**	**4.000**	5.620	6.830	5.620	7.620	**4.000**	8.670	5.710	**4.000**
0.9	Max	1.000 /20	**1.000**	**1.000**	1.000 /70	1.000 /50	1.000 /60	1.000 /50	**1.000**	1.000 /50	1.000 /90	**1.000**
	Min	0.000 /70	**1.000**	**1.000**	0.740 /120	0.000 /120	0.700 /120	0.000 /110	**1.000**	0.000 /100	0.000 /120	**1.000**
	Mean	0.257	**1.000**	**1.000**	0.946	0.617	0.952	0.599	**1.000**	0.530	0.817	**1.000**
	Std	0.405	**0.000**	**0.000**	0.090	0.450	0.086	0.463	**0.000**	0.482	0.356	**0.000**
	Avg Rank	9.870	**4.120**	**4.120**	5.460	7.250	5.750	7.580	**4.120**	8.170	5.420	**4.120**

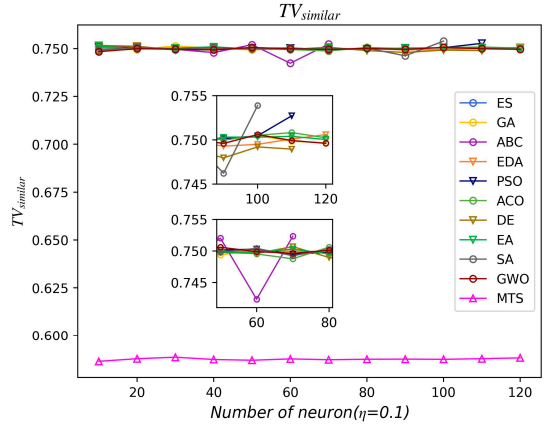
Tables 10 and 11 show the energy error MAE_{test} for 11 experimental models taking different proportions of positive literal (η) and number of neurons. This comparison is vital for assessing the model's convergence during the retrieval phase, with lower metric values signifying improved model convergence.

Table 10. The average MAE_{test} value of all proposed algorithms during the retrieval phase at $\eta = \{0.1, 0.2, 0.3, 0.4, 0.5\}$. The bold values indicate the superior result among the mentioned metrics.

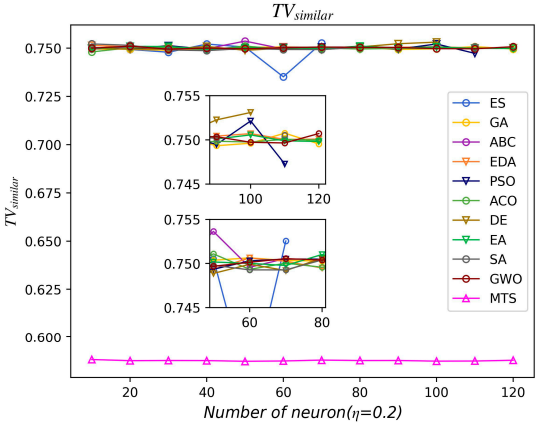
η	Measure	ES	GA	ABC	EDA	PSO	ACO	DE	EA	SA	GWO	MTS
0.1	Max	14.604 /120	**0.000**	16.077 /120	2.668 /120	15.498 /120	3.160 /120	14.857 /120	**0.000**	14.944 /120	12.505 /120	**0.000**
	Min	0.000 /20	**0.000**	0.000 /20	0.000 /50	0.000 /60	0.000 /60	0.000 /60	**0.000**	0.000 /50	0.000 /70	**0.000**
	Mean	7.388	**0.000**	7.452	0.659	4.925	0.589	4.955	**0.000**	5.587	2.112	**0.000**
	Std	5.287	**0.000**	5.854	1.013	6.200	1.021	6.230	**0.000**	6.148	4.151	**0.000**
	Avg Rank	9.333	**3.625**	9.500	5.417	6.750	4.917	6.667	**3.625**	7.417	5.125	**3.625**
0.2	Max	15.559 /120	**0.000**	17.358 /120	3.559 /120	14.524 /120	2.077 /120	15.431 /120	**0.000**	14.747 /120	13.961 /120	**0.000**
	Min	0.000 /20	**0.000**	0.000 /20	0.000 /60	0.000 /50	0.000 /60	0.000 /60	**0.000**	0.000 /50	0.000 /70	**0.000**
	Mean	7.409	**0.000**	7.228	0.762	4.760	0.489	5.106	**0.000**	5.541	2.775	**0.000**
	Std	5.422	**0.000**	5.740	1.252	6.049	0.756	6.286	**0.000**	6.050	4.963	**0.000**
	Avg Rank	9.667	**3.625**	9.417	5.167	6.750	4.833	6.833	**3.625**	7.333	5.125	**3.625**

Table 10. *Cont.*

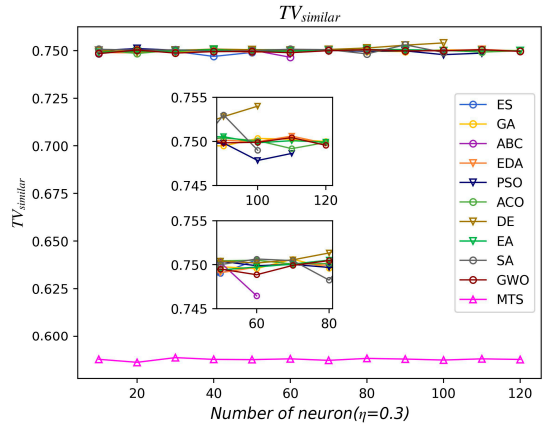
η	Measure	ES	GA	ABC	EDA	PSO	ACO	DE	EA	SA	GWO	MTS
0.3	Max	15.045 /120	**0.000**	14.590 /120	3.271 /120	14.255 /120	2.920 /120	14.752 /120	**0.000**	15.040 /120	14.117 /120	**0.000**
	Min	0.000 /20	**0.000**	0.000 /30	0.000 /60	0.000 /60	0.000 /50	0.000 /50	**0.000**	0.000 /50	0.000 /70	**0.000**
	Mean	7.374	**0.000**	6.935	0.739	4.843	0.604	4.881	**0.000**	5.633	3.190	**0.000**
	Std	5.346	**0.000**	5.493	1.197	6.089	1.019	6.082	**0.000**	6.143	5.226	**0.000**
	Avg Rank	9.917	**3.625**	8.792	5.167	6.583	4.958	6.792	**3.625**	7.625	5.292	**3.625**
0.4	Max	15.272 /120	**0.000**	17.675 /120	5.091 /120	15.229 /120	3.289 /120	14.988 /120	**0.000**	14.791 /120	14.167 /120	**0.000**
	Min	0.000 /20	**0.000**	0.000 /40	0.000 /50	0.000 /60	0.000 /60	0.000 /60	**0.000**	0.000 /50	0.000 /70	**0.000**
	Mean	7.420	**0.000**	5.929	0.834	4.863	0.607	5.021	**0.000**	5.631	3.399	**0.000**
	Std	5.392	**0.000**	6.541	1.465	6.149	1.000	6.207	**0.000**	6.185	5.306	**0.000**
	Avg Rank	9.833	**3.667**	8.167	5.458	6.542	5.250	6.792	**3.667**	7.625	5.333	**3.667**
0.5	Max	15.067 /120	**0.000**	5.601 /120	3.354 /120	14.711 /120	2.880 /120	14.581 /120	**0.000**	14.902 /120	14.661 /120	**0.000**
	Min	0.000 /20	**0.000**	0.000 /70	0.000 /70	0.000 /50	0.000 /50	0.000 /60	**0.000**	0.000 /50	0.000 /70	**0.000**
	Mean	7.396	**0.000**	0.990	0.666	5.094	0.498	4.993	**0.000**	5.641	3.392	**0.000**
	Std	5.332	**0.000**	1.715	1.120	6.140	0.869	6.092	**0.000**	6.142	5.371	**0.000**
	Avg Rank	9.833	**3.833**	5.333	5.083	7.542	5.542	6.875	**3.833**	8.292	6.000	**3.833**



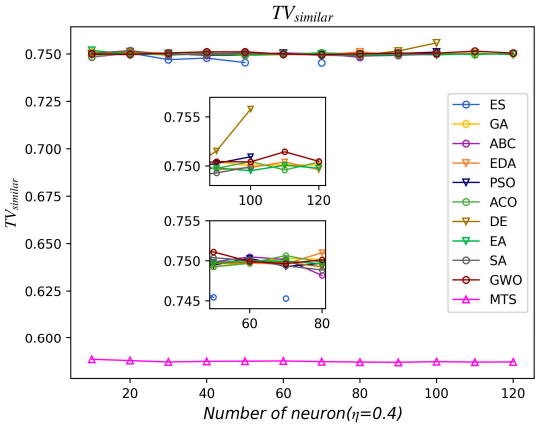
(**a**) $TV_{similar}$ evaluation when $\eta = 0.1$



(**b**) $TV_{similar}$ evaluation when $\eta = 0.2$

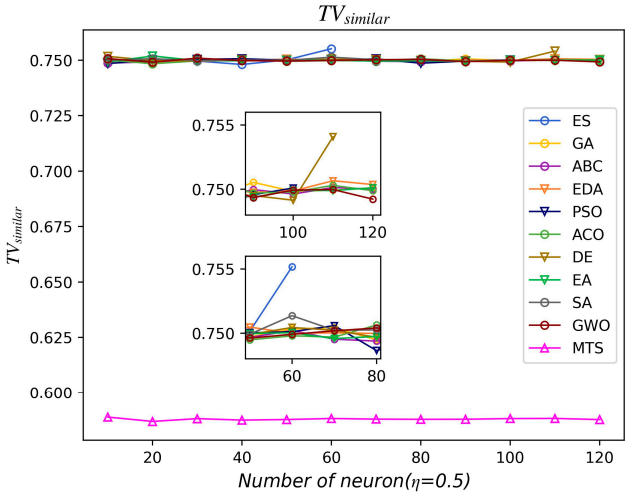


(**c**) $TV_{similar}$ evaluation when $\eta = 0.3$

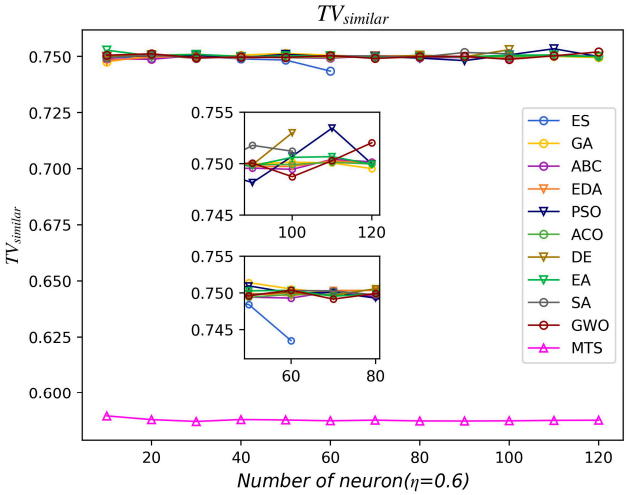


(**d**) $TV_{similar}$ evaluation when $\eta = 0.4$

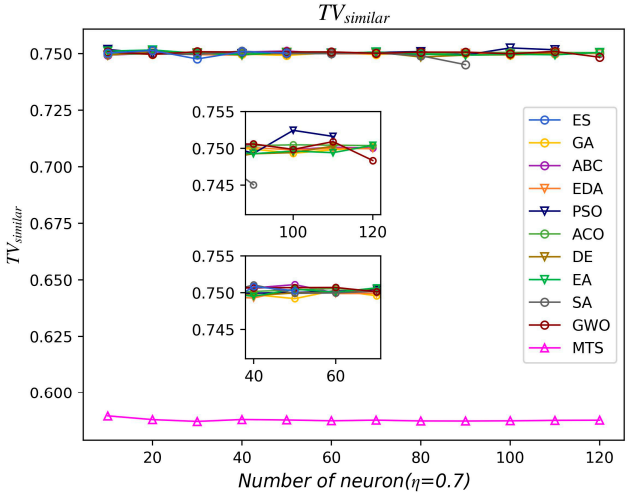
Figure 8. *Cont.*



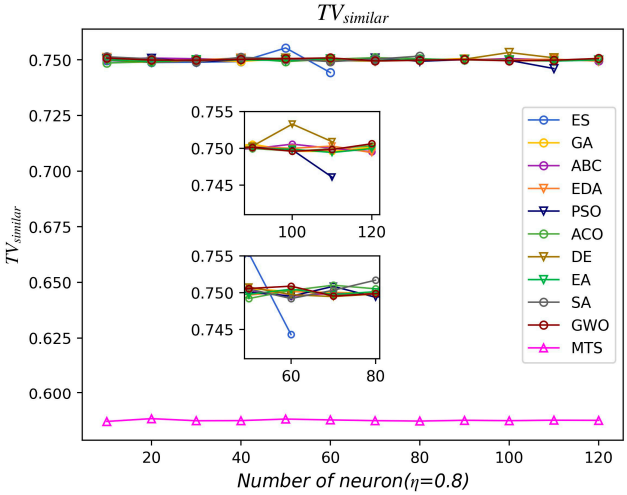
(**e**) $TV_{similar}$ evaluation when $\eta = 0.5$



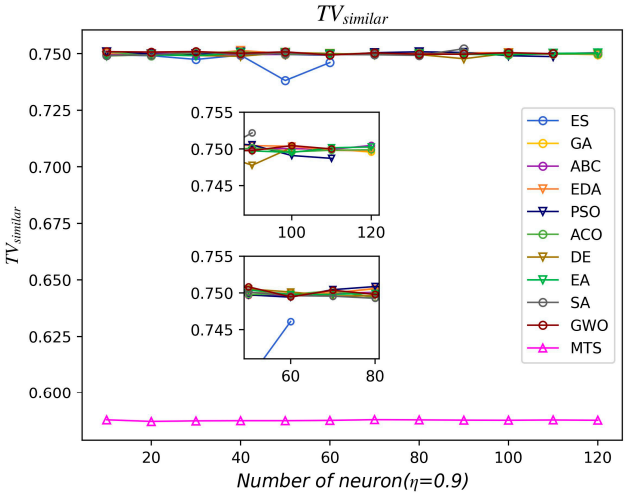
(**f**) $TV_{similar}$ evaluation when $\eta = 0.6$



(**g**) $TV_{similar}$ evaluation when $\eta = 0.7$



(**h**) $TV_{similar}$ evaluation when $\eta = 0.8$



(**i**) $TV_{similar}$ evaluation when $\eta = 0.9$

Figure 8. $TV_{similar}$ evaluation for all experimental models.

Table 11. The average MAE_{test} value of all proposed algorithms during the retrieval phase at $\eta = \{0.6, 0.7, 0.8, 0.9\}$. The bold values indicate the superior result among the mentioned metrics.

η	Measure	ES	GA	ABC	EDA	PSO	ACO	DE	EA	SA	GWO	MTS
0.6	Max	14.723 /120	**0.000**	**0.000**	3.678 /120	15.109 /120	1.692 /120	14.908 /120	**0.000**	14.852 /120	14.030 /120	**0.000**
	Min	0.000 /10	**0.000**	**0.000**	0.000 /60	0.000 /60	0.000 /60	0.0000 /60	**0.000**	0.000 /50	0.000 /70	**0.000**
	Mean	7.383	**0.000**	**0.000**	0.696	4.803	0.474	4.853	**0.000**	5.555	3.399	**0.000**
	Std	5.311	**0.000**	**0.000**	1.142	6.063	0.695	6.068	**0.000**	6.091	5.386	**0.000**
	Avg Rank	10.083	**4.042**	**4.042**	5.667	7.250	5.500	7.500	**4.042**	7.917	5.917	**4.042**
0.7	Max	14.888 /120	**0.000**	**0.000**	2.596 /120	14.221 /120	1.691 /120	14.912 /120	**0.000**	14.865 /120	14.975 /120	**0.000**
	Min	0.000 /10	**0.000**	**0.000**	0.000 /70	0.000 /60	0.000 /60	0.000 /60	**0.000**	0.000 /50	0.000 /70	**0.000**
	Mean	7.384	**0.000**	**0.000**	0.625	4.694	0.457	4.947	**0.000**	5.677	3.085	**0.000**
	Std	5.357	**0.000**	**0.000**	0.944	6.009	0.704	6.190	**0.000**	6.209	5.346	**0.000**
	Avg Rank	10.083	**4.083**	**4.083**	5.458	6.833	5.500	7.583	**4.083**	7.917	6.292	**4.083**
0.8	Max	14.930 /120	**0.000**	**0.000**	2.892 /120	15.01 /120	2.513 /120	14.982 /120	**0.000**	14.813 /120	12.733 /120	**0.000 /120**
	Min	0.000 /20	**0.000**	**0.000**	0.000 /60	0.000 /60	0.000 /60	0.000 /50	**0.000**	0.000 /40	0.000 /70	**0.000 /10**
	Mean	7.551	**0.000**	**0.000**	0.633	4.949	0.539	4.909	**0.000**	5.620	2.597	**0.000**
	Std	5.421	**0.000**	**0.000**	0.962	6.114	0.932	6.063	**0.000**	6.093	4.531	**0.000**
	Avg Rank	9.917	**4.000**	**4.000**	5.708	7.375	5.458	7.500	**4.000**	8.250	5.792	**4.000**
0.9	Max	14.821 /120	**0.000**	**0.000**	2.955 /120	14.389 /120	2.222 /120	15.276 /120	**0.000**	15.069 /120	11.345 /120	**0.000**
	Min	0.000 /20	**0.000**	**0.000**	0.000 /70	0.000 /50	0.000 /60	0.000 /50	**0.000**	0.000 /50	0.000 /90	**0.000**
	Mean	8.841	**0.000**	**0.000**	0.997	8.143	0.515	5.186	**0.000**	9.681	4.500	**0.000**
	Std	5.304	**0.000**	**0.000**	0.979	5.835	0.822	6.216	**0.000**	6.197	3.805	**0.000**
	Avg Rank	9.667	**4.125**	**4.125**	5.583	7.125	5.625	7.958	**4.125**	8.292	5.250	**4.125**

The optimal data in Tables 10 and 11 are identified in bold. As described in the table, we can see that the models GA, EA, and MTS perform more prominently than the other models in terms of energy error, which consistently takes the value of 0 for different numbers of neurons. From $\left|E_{PRO2SAT} - E_{PRO2SAT}^{\min}\right| = C_{PRO2SAT}$, the energy error value is equivalent to the cost function value, and the value of $C_{PRO2SAT}$ is equal to the sum of unsatisfiable clauses in the range of [0, n]. n represents the number of second-order clauses. As the clause satisfaction rate of PRO2SAT increases during the retrieval phase, the energy error value decreases. When the optimal prominence weights are obtained, all types of clauses will be satisfied. This shows that the model that performs well in the learning phase will have a smaller energy error in the retrieval phase. With the analysis of the retrieval phase, we learn that the GA, EA, and MTS models all have better performance, thanks to the excellent performance of these models in the learning phase. While ABC also performs well in terms of energy error, it has a biased final state of neurons and is not suitable for all PRO2SAT models under positive literal occupancy.

8.3. Friedman Test

We conducted the Friedman test for all metric values with different η values in the learning and retrieval phases. It can be seen that the Chi-square value takes the interval [64.303, 77.455] for MAE_{learn}, [71.525, 78.046] for $LRTR$, [40.920, 63.445] for GLI_{avg}, [87.320, 100.993] for CT, [71.021, 81.908] for AI, [71.689, 80.273] for ZM, [27.707, 52.180] for $TV_{similar}$, and [68.753, 78.643] for MAE_{test}. The degree of freedom of these indicators is 10. Most importantly, the p-value values for all metrics are less than 0.05 ($p < 0.05$), as shown in Table 12, representing that the null hypothesis of similar performance between MTS and

all compared models is rejected, that is, the results of all models are significantly different across metrics.

Table 12. Performing a Friedman test on MTS using various comparison algorithms.

Model	η	Chi-Square Value, χ^2	p-Value	Accept(A)/Reject(R), H_0	Model	Chi-Square Value, χ^2	p-Value	Accept/Reject, H_0
	0.1	67.727	1.215×10^{-10}	R H_0		79.697	5.7561×10^{-13}	R H_0
	0.2	70.182	4.089×10^{-11}	R H_0		80.273	4.4381×10^{-13}	R H_0
	0.3	69.546	5.426×10^{-11}	R H_0		77.994	1.2399×10^{-12}	R H_0
	0.4	64.303	5.509×10^{-10}	R H_0		73.484	9.3768×10^{-12}	R H_0
	0.5	75.046	4.662×10^{-12}	R H_0		71.689	2.0903×10^{-11}	R H_0
	0.6	77.455	1.581×10^{-12}	R H_0		78.046	1.2112×10^{-12}	R H_0
	0.7	72.167	1.689×10^{-12}	R H_0		77.552	1.5129×10^{-12}	R H_0
	0.8	81.742	2.285×10^{-13}	R H_0		76.690	2.229×10^{-12}	R H_0
	0.9	71.621	2.154×10^{-11}	R H_0		71.525	2.249×10^{-11}	R H_0
	0.1	41.028	1.116×10^{-5}	R H_0		98.527	1.074×10^{-16}	R H_0
	0.2	40.920	1.166×10^{-5}	R H_0		87.320	1.818×10^{-14}	R H_0
	0.3	54.616	3.725×10^{-8}	R H_0		97.046	2.122×10^{-16}	R H_0
	0.4	53.175	6.909×10^{-8}	R H_0		96.696	2.493×10^{-16}	R H_0
GLI_{avg}	0.5	54.191	4.472×10^{-8}	R H_0	CT	93.146	1.271×10^{-15}	R H_0
	0.6	58.906	5.835×10^{-9}	R H_0		100.993	3.448×10^{-17}	R H_0
	0.7	55.253	2.833×10^{-8}	R H_0		90.348	4.569×10^{-15}	R H_0
	0.8	63.445	8.032×10^{-10}	R H_0		93.743	9.667×10^{-16}	R H_0
	0.9	57.113	1.270×10^{-8}	R H_0		93.733	9.711×10^{-16}	R H_0
	0.1	76.419	2.517×10^{-12}	R H_0		79.650	5.878×10^{-13}	R H_0
	0.2	72.226	1.645×10^{-11}	R H_0		80.273	4.438×10^{-13}	R H_0
	0.3	76.282	2.677×10^{-12}	R H_0		77.843	1.327×10^{-12}	R H_0
	0.4	74.333	6.416×10^{-12}	R H_0		72.248	1.629×10^{-11}	R H_0
AI	0.5	75.121	4.507×10^{-12}	R H_0	ZM	71.689	2.090×10^{-11}	R H_0
	0.6	81.908	2.120×10^{-13}	R H_0		78.046	1.211×10^{-12}	R H_0
	0.7	71.021	2.816×10^{-11}	R H_0		77.552	1.513×10^{-12}	R H_0
	0.8	76.477	2.452×10^{-12}	R H_0		76.690	2.229×10^{-12}	R H_0
	0.9	74.019	7.383×10^{-12}	R H_0		73.164	1.082×10^{-11}	R H_0
	0.1	31.566	4.729×10^{-4}	R H_0		74.835	5.124×10^{-12}	R H_0
	0.2	28.559	1.467×10^{-3}	R H_0		78.643	9.257×10^{-13}	R H_0
	0.3	40.200	1.563×10^{-5}	R H_0		76.486	2.443×10^{-11}	R H_0
	0.4	40.246	1.53×10^{-5}	R H_0		68.753	7.714×10^{-11}	R H_0
$TV_{similar}$	0.5	27.707	2.011×10^{-3}	R H_0	MAE_{test}	69.862	4.714×10^{-11}	R H_0
	0.6	40.627	1.313×10^{-3}	R H_0		73.134	1.097×10^{-11}	R H_0
	0.7	43.478	4.084×10^{-6}	R H_0		71.994	1.825×10^{-12}	R H_0
	0.8	40.727	1.261×10^{-5}	R H_0		73.922	7.790×10^{-12}	R H_0
	0.9	52.180	1.057×10^{-7}	R H_0		72.254	1.625×10^{-11}	R H_0

8.4. Conclusions

In this paper, the MTS metaheuristic algorithm was introduced in systematic logic for the first time. Taking the representative systematic logic programming PRO2SAT as an example, we introduced the MTS metaheuristic algorithm and nine advanced metaheuristics (GA, EA, ACO, ED, DE, GWO, PSO, SA, and ABC). What is more, the ES algorithm was added to carry out experiments and comparisons from two dimensions in both the learning and retrieval phases, giving all the experimental model calculations and quantitatively analyzing the results. Finally, we used the Friedman test to test the significant difference between the experimental results in these two dimensions.

Based on the experimental results presented in Sections 8.1–8.3, the MTS model demonstrated superior performance over other models in both the learning and retrieval phases, particularly surpassing the exhaustive search approach. In the learning phase, the MTS metaheuristic algorithm exhibits higher retrieval efficiency by starting from a high-quality solution. The neighborhood solution generation strategy makes the process of solution generation simple, direct, and adaptive, resulting in high-quality solutions during the learning phase. Due to the short-term "memory" and long-term "memory" of the MTS algorithm, it can balance efficiency and stability. The overall algorithm structure

of the MTS metaheuristic algorithm was simple and with low computational complexity, providing faster search speed compared with other metaheuristic algorithms and finding the optimal solution faster. These characteristics were reflected in the $LRTR$, GLI_{avg}, CT, AI, and $TV_{similar}$ metrics, due to the fact that the active role of the MTS algorithm in the learning phase provided advantages to the retrieval phase of the MTS model.

A successful new metaheuristic algorithm introduced into logic programming, the MTS algorithm has achieved excellent performance in systematic logic and is also expected to be introduced into non-systematic logic to enhance the performance of non-systematic logic. The enhancement of metaheuristic algorithms for DHNN-SAT models is not limited to finding consistent interpretations, but can also be used to enhance solution diversity, logic construction regularity, etc. For future work, metaheuristic algorithms can be used to enhance logic programming from more dimensions. PRO2SAT's powerful architecture provides a new perspective on the application of knowledge in real life. For the future work, the proposed PRO2SAT can be embedded in logic mining to extract the best logic rules to classify and predict palm oil.

Author Contributions: Conceptualization, Project Administration, Writing—Original Draft, J.C.; Visualization, Y.G.; Resources, M.S.M.K.; Formal Analysis, Investigation, C.Z.; Methodology, Writing—Review & Editing, N.A.R.; Validation, M.A.M. and N.E.Z.; Funding Acquisition, C.W. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by Xinglin Scholar, Chengdu University of Traditional Chinese Medicine with Project number (QNXZ2018042).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data are contained within the article.

Acknowledgments: The authors would like to express special thanks to all researchers in the AI Research Development Group for their continued support.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Beam, A.L.; Kohane, I.S. Big Data and Machine Learning in Health Care. *JAMA* **2018**, *319*, 1317–1318. [CrossRef] [PubMed]
2. Vigilante, K.; Escaravage, S.; McConnell, M. Big Data and the Intelligence Community-Lessons for Health Care. *N. Engl. J. Med.* **2019**, *380*, 1888–1890. [CrossRef] [PubMed]
3. Egmont-Petersen, M.; de Ridder, D.; Handels, H. Image processing with neural networks—A review. *Pattern Recognit.* **2002**, *35*, 2279–2301. [CrossRef]
4. Dang, X.; Tang, X.; Hao Ren, J. Discrete Hopfield neural network based indoor Wi-Fi localization using CSI. *EURASIP J. Wirel. Commun. Netw.* **2020**, *2020*, 76. [CrossRef]
5. Mérida-Casermeiro, E.; Galán-Marín, G.; Muoz-Pérez, J. An Efficient Multivalued Hopfield Network for the Traveling Salesman Problem. *Neural Process. Lett.* **2001**, *14*, 203–216. [CrossRef]
6. Chu, P.P. Applying Hopfield network to find the minimum cost coverage of a Boolean function. In Proceedings of the First Great Lakes Symposium on VLSI, Kalamazoo, MI, USA, 1–2 March 1991; IEEE Computer Society: Washington, DC, USA, 1991; pp. 182–183.
7. Abdullah, W.A.T.W. Logic programming on a neural network. *Int. J. Intell. Syst.* **1992**, *7*, 513–519. [CrossRef]
8. Chen, J.; Kasihmuddin, M.S.M.; Gao, Y.; Mansor, M.A.; Romli, N.A.; Chen, W.; Zheng, C. PRO2SAT: Systematic Probabilistic Satisfiability logic in Discrete Hopfield Neural Network. *Adv. Eng. Softw.* **2023**, *175*, 103355. [CrossRef]
9. Hussain, K.; Mohd Salleh, M.N.; Cheng, S.; Shi, Y. Metaheuristic research: A comprehensive survey. *Artif. Intell. Rev.* **2019**, *52*, 2191–2233. [CrossRef]
10. Abdel-Basset, M.; Abdel-Fatah, L.; Sangaiah, A.K. Metaheuristic algorithms: A comprehensive review. In *Computational Intelligence for Multimedia Big Data on the Cloud with Engineering Applications*; Elsevier: Amsterdam, The Netherlands; Academic Press: Cambridge, MA, USA, 2018; pp. 185–231.
11. Zamri, N.E.; Azhar, S.A.; Mansor, M.A.; Alway, A.; Kasihmuddin MS, M. Weighted Random k Satisfiability for k = 1, 2 (r2SAT) in Discrete Hopfield Neural Network. *Appl. Soft Comput.* **2022**, *126*, 109312. [CrossRef]

12. Someetheram, V.; Marsani, M.F.; Mohd Kasihmuddin, M.S.; Zamri, N.E.; Muhammad Sidik, S.S.; Mohd Jamaludin, S.Z.; Mansor, M.A. Random Maximum 2 Satisfiability Logic in Discrete Hopfield Neural Network Incorporating Improved Election Algorithm. *Mathematics* **2022**, *10*, 4734. [CrossRef]
13. Muhammad Sidik, S.S.; Zamri, N.E.; Mohd Kasihmuddin, M.S.; Wahab, H.A.; Guo, Y.; Mansor, M.A. Non-Systematic Weighted Satisfiability in Discrete Hopfield Neural Network Using Binary Artificial Bee Colony Optimization. *Mathematics* **2022**, *10*, 1129. [CrossRef]
14. Kho, L.C.; Kasihmuddin, M.S.M.; Mansor, M.A.; Sathasivam, S. Propositional Satisfiability Logic via Ant Colony Optimization in Hopfield Neural Network. *Malays. J. Math. Sci.* **2022**, *16*, 37–53.
15. Ba, S.; Xia, D.; Gibbons, E.M. Model identification and strategy application for Solid Oxide Fuel Cell using Rotor Hopfield Neural Network based on a novel optimization method. *Int. J. Hydrogen Energy* **2020**, *45*, 27694–27704. [CrossRef]
16. Glover, F. Tabu search: A tutorial. *Interfaces* **1990**, *20*, 74–94. [CrossRef]
17. Glover, F.; Laguna, M. Tabu search. In *Handbook of Combinatorial Optimization*; Springer: Berlin/Heidelberg, Germany, 1998; pp. 2093–2229.
18. Gopalakrishnan, M.; Mohan, S.; He, Z. A tabu search heuristic for preventive maintenance scheduling. *Comput. Ind. Eng.* **2001**, *40*, 149–160. [CrossRef]
19. Meeran, S.; Morshed, M.S. A hybrid genetic tabu search algorithm for solving job shop scheduling problems: A case study. *J. Intell. Manuf.* **2012**, *23*, 1063–1078. [CrossRef]
20. Žulj, I.; Kramer, S.; Schneider, M. A hybrid of adaptive large neighborhood search and tabu search for the order-batching problem. *Eur. J. Opeproportionnal Res.* **2018**, *264*, 653–664. [CrossRef]
21. Lin, G.; Guan, J.; Li, Z.; Feng, H. A hybrid binary particle swarm optimization with tabu search for the set-union knapsack problem. *Expert Syst. Appl.* **2019**, *135*, 201–211. [CrossRef]
22. Mohd Jamaludin, S.Z.; Mohd Kasihmuddin, M.S.; Md Ismail, A.I.; Mansor, M.A.; Md Basir, M.F. Energy based logic mining analysis with hopfield neural network for recruitment evaluation. *Entropy* **2021**, *23*, 40. [CrossRef]
23. Mirjalili, S.; Mirjalili, S.M.; Lewis, A. Grey wolf optimizer. *Adv. Eng. Softw.* **2014**, *69*, 46–61. [CrossRef]
24. Gendreau, M.; Iori, M.; Laporte, G.; Martello, S. A Tabu search heuristic for the vehicle routing problem with two-dimensional loading constraints. *Int. J.* **2008**, *51*, 4–18. [CrossRef]
25. Misevicius, A. A tabu search algorithm for the quadratic assignment problem. *Comput. Optim. Appl.* **2005**, *30*, 95–111. [CrossRef]
26. Holland, J.H. Genetic algorithms and the optimal allocation of trials. *SIAM J. Comput.* **1973**, *2*, 88–105. [CrossRef]
27. Emami, H.; Derakhshan, F. Election algorithm: A new socio-politically inspired strategy. *AI Commun.* **2015**, *28*, 591–603. [CrossRef]
28. Pelikan, M.; Sastry, K.; Goldberg, D.E. Multiobjective estimation of distribution algorithms. In *Scalable Optimization via Probabilistic Modeling*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 223–248.
29. He, Y.; Zhang, F.; Mirjalili, S.; Zhang, T. Novel binary differential evolution algorithm based on taper-shaped transfer functions for binary optimization problems. *Swarm Evol. Comput.* **2022**, *69*, 101022. [CrossRef]
30. Poli, R.; Kennedy, J.; Blackwell, T. Particle swarm optimization. *Swarm Intell.* **2007**, *1*, 33–57. [CrossRef]
31. Kirkpatrick, S.; Gelatt, C.D., Jr.; Vecchi, M.P. Optimization by Simulated Annealing. *Science* **1983**, *220*, 671–680. [CrossRef] [PubMed]
32. Eiben, A.E.; Smith, J.E. *Introduction to Evolutionary Computation*; Springer: Berlin/Heidelberg, Germany, 2015.
33. Deng, W.; Shang, S.; Cai, X.; Zhao, H.; Song, Y.; Xu, J. An improved differential evolution algorithm and its application in optimization problem. *Soft Comput.* **2021**, *25*, 5277–5298. [CrossRef]
34. Qin, A.K.; Huang, V.L.; Suganthan, P.N. Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE Trans. Evol. Comput.* **2008**, *13*, 398–417. [CrossRef]