Ant Colony Optimization Architecture for the CASH Problem in the Context of Machine Learning Classification

Diego D. Fernandes
State University of Ceará
Fortaleza-CE, Brazil
diego.dias@aluno.uece.br

Thalyson G. N. da Silva
State University of Ceará
Fortaleza-CE, Brazil
thalyson.uece@gmail.com

Gustavo A. L. de Campos
State University of Ceará
Fortaleza-CE, Brazil
gustavo@larces.uece.br

Ana Luiza B. de Paula Barros
State University of Ceará
Fortaleza-CE, Brazil
analuiza.barros@uece.br

Felipe G. Marajo
State University of Ceará
Fortaleza-CE, Brazil
felipe.marajo@aluno.uece.br

***Abstract*—This paper presents ACOMV-CASH, an automated machine learning (AutoML) approach based on Ant Colony Optimization (ACO) to select a classifier from K-Nearest Neighbors (KNN), Multi-Layer Perceptron (MLP), Decision Tree, Random Forest Classifier (RFC), and Support Vector Machine (SVM), while also optimizing model-specific hyperparameters. The framework incorporates a preprocessing pipeline that automatically prepares the dataset. Finally, the proposed algorithm brings the traditional ACO architecture back into the spotlight by structuring the search space as a directed graph (digraph) and incorporating novel strategies, such as the generation of continuous variables and hierarchical modeling. Evaluation of eight UCI datasets demonstrates competitive performance compared to existing AutoML frameworks, highlighting the robustness of ACOMV-CASH.**

AutoML Ant Colony Optimization Classification.

I. Introduction

Although the relevance of Machine Learning (ML) has increased drastically in recent years, and there is a growing demand for the application of ML techniques to generate models that solve various types of Artificial Intelligence (AI) problems in socially relevant areas, the number of specialists in the field remains insufficient and is concentrated in a few high-demand projects. The presence of specialists is essential because generating a model to solve a specific problem using an ML technique requires knowledge of the hyperparameters associated with each technique and how to select appropriate values for these hyperparameters in order to maximize the model's performance, considering one or more specific datasets related to the problem.

This reliance on a variety of specialized decisions poses a barrier for new users of machine learning (ML) techniques. The field of automated machine learning (AutoML) seeks to reduce these barriers by automating the specialized design choices required to generate high-performance models. By using an AutoML system, users only need to provide the data related to their problem, and the system will automatically determine the most suitable ML techniques and the corresponding hyperparameter values to produce an effective model. In this context, AutoML systems have contributed to the democratization of ML by breaking down barriers and making the technology more accessible to a wider audience [1].

Currently, there are several mature AutoML systems, some of which are better than most human experts. For example, Auto-WEKA [2] is one of the first AutoML frameworks, based on WEKA [2], a machine learning platform written in Java. The Auto-Sklearn framework [3] is widely used due to its integration with Scikit-learn and its ease of use. AutoKeras [4] is an AutoML framework designed to work on top of Keras and TensorFlow, providing an easy-to-use interface for automatically building deep learning [5] models. [6] reviews some of those approaches and more.

Different types of problems, however, are associated with these frameworks. Some of them, for example, may face difficulties when dealing with very large datasets. Others may be limited in terms of support for new algorithms that may become available in the future. Users may not have the necessary computational resources to perform an intensive hyperparam-

eter search process, which can take a significant amount of time to train models, especially on large datasets. Despite being automated, some frameworks still require an advanced understanding of ML itself or the underlying libraries, which can be a barrier for new users.

Many of these limitations arise from the fact that these frameworks are designed to generate models to solve all generic AI problems, such as pattern recognition, classification, diagnosis, monitoring, control, planning, among others. More focused AutoML approaches [7]–[11], designed to solve just one of these problems within a specific domain, can generate models with superior performance compared to those generated by most general-purpose frameworks [2]–[4]. There are many approaches developed for the automatic generation of models to solve each of these generic problems in specific domains. Most of these approaches focus on models to solve classification problems, since other problems in the context of human-centric AI, such as the diagnosis problem, can be formally defined as a classification problem.

The problem of generating models to solve specific classification problems, in different domains, has been approached as an optimization problem. The solving system seeks to find one or more ML techniques and the values of their associated hyperparameters that maximize the performance of the generated classification model. Within the context of AutoML approaches to solving specific problems, there are problem-solving approaches, defined in high-dimensional spaces and low computational cost, that employ Bayesian Optimization [12], Grid Search [13], Random Search [14], Genetic Algorithms [7]–[9], Swarm Intelligence [10], [11], among other meta-heuristics.

The main limitations of AutoML approaches that treat model generation as an optimization problem include the complexity and high computational cost associated with high-dimensional state spaces. Methods such as Bayesian Optimization, although efficient, can be limited by the assumption that the search space is smooth and continuous. Grid Search and Random Search are simple but inefficient for problems with many hyperparameters, as they may require many evaluations to find an optimal configuration.

This work consists of an approach that integrates a Pre-processing pipeline and a system based on Ant Colony Optimization (ACO) [15] to generate a high performance classification model by selecting a machine learning model and its associated hyperparameter values. The approach is named Combined Algorithm and Hyper-parameter Selection (CASH). Several adaptations were made to the original ACO meta-heuristic, aiming to master the complexity associated with the optimization problem without compromising the performance of the generated models. Among these adaptations, it is worth highlighting a structure in the form of a special graph to define the state space of the problem to be searched by the ants, some strategies to deal with the mixed-variable nature of the problem, and a new formulation for pheromone distribution.

The remaining sections of this paper are organized as follows: Section 2 presents a literature review and contextualizes recent research in the areas of Automated Machine Learning (AutoML), metaheuristics (MH), and the CASH approach. Section 3 describes the pipeline steps of this project, including data preprocessing, the proposed **ACOMV-CASH** structure, and its evaluation. Section 4 discusses the experimental results and compares the proposed method with other approaches, while Section 5 concludes the paper and outlines directions for future work.

II. Related Works

The problem of searching for hyperparameters in Machine Learning models has been studied using a combination of techniques due to its inherent complexity. Some approaches include Bayesian Optimization [12]; Grid Search, where a set of hyperparameters are selected by a scientist and an exhaustive search of all combinations are tested, which present good results for the optimization of few hyper parameters but can be impracticable due to higher dimensional spaces; Random Search, in which serves better for higher dimensional spaces then Grid Search and can be optimized, for example [14]; Genetic [7]–[9] and Swarm Intelligence such as the Particle Swarm [16] and Ant Colony Optimization, where the study of this paper lies on.

Meta heuristics such as Genetic Algorithms [8] and Swarm Intelligence Optimizers (SIO) [9], [10] have been massively studied aiming mainly two interconnected points: Improved results and less computational cost for higher dimensional spaces then the popular Grid Search (GS) and Random Search (RS) methods. This issue has been raised due to more complex models such as Deep Learning strategies. The number of hyper parameters of many models could make GS and RS methods impracticable. On the other hand, GA and SIO have real world based strategies that try to converge to the global minima in a guided manner.

The second part of the problem this paper focus is the CASH problem, or Full Model Selection, FMS, that incorporates the choice of a classifier on top of its hyper parameters. Famous Auto-ML Frameworks that resolve this problem are Auto-Sklearn and Auto-WEKA, but, recently more studies are being invested on this field, despite not many, being that by Bayesian optimization [17] or Swarm Intelligence [11]. The main issue with this problem is the vast search space that are independent, that is, in single classifier hyper parameter optimizations, every variable is related to the other ones so the model is built. For

this, we will be using a novel approach of the Ant Colony Optimization, ACO.

ACO emerged as a solution to the Traveling Salesman problem [18] where ants walk through the edges of a graph and are guided by pheromones left by previous ants; therefore, to resolve problems from categorical domains, but since more studies around ACOs were developed, an ACO for continuous domains is presented in [19]. To this point we had an ACO for the continuous domain and another one for categorical domain, but the hyper-parameter optimization for the Machine Learning problem often needs both simultaneously, in this context a **ACOMV** proposed by [20], dealt with the so-called Mixed Variable problem and presented many more improvements to the existing method.

III. ACOMV-CASH AUTOML

This section introduces the proposed **ACOMV-CASH** system, which has been developed to perform model selection and hyperparameter optimization within a complete AutoML pipeline. Section III-A provides an overview of the AutoML process, highlighting the role of **ACOMV-CASH** and detailing the sequential steps of the pipeline. Section III-B describes the internal architecture of the proposed architecture, including the graph-based modeling of the search space, strategies for handling mixed-variable types, construction mechanisms, pheromone update rules, decision processes, and the hierarchical structure of the search spaces.

A. AutoML *ACOMV-CASH* Flowchart

Figure 1 illustrates the flowchart of the proposed AutoML framework incorporating the **ACOMV-CASH** system. The process begins with data preprocessing, followed by automated model selection and hyperparameter optimization, which are carried out by the **ACOMV-CASH** module, ultimately leading to the generation of high-performance classification models. The following paragraphs describe each pipeline stage to provide a comprehensive overview before delving into the internal structure of the **ACOMV-CASH** system.

The process begins with the input dataset, which undergoes a preprocessing phase. This step involves essential transformations, such as data cleaning, normalization, encoding of categorical variables, and handling of missing values, ensuring the data is suitable for subsequent modeling tasks.

Following preprocessing, the pipeline proceeds to model selection and hyperparameter tuning, both managed by the **ACOMV-CASH** module. In this stage, multiple machine learning algorithms and their respective hyperparameters are explored simultaneously. The exploration is guided by an Ant Colony Optimization (ACO) strategy, where artificial ants construct solutions by selecting models and hyperparameter configurations based on pheromone trails and heuristic information.

After each exploration, candidate solutions are evaluated on a validation set. The system checks whether the maximum number of ants predefined for the search process has been reached. If not, the search continues with further iterations; otherwise, the best-performing configuration found is selected. Finally, the selected model and hyperparameters are applied to the reserved test split to assess generalization performance, completing the AutoML process.

The next section provides a detailed description of the internal architecture of the **ACOMV-CASH** system, including its graph-based structure, construction mechanisms, pheromone update strategies, and the hierarchical organization of the search space.

B. *ACOMV-CASH* Architecture

The internal architecture of **ACOMV-CASH** enables the efficient exploration of model and hyperparameter search spaces through an Ant Colony Optimization (ACO) framework. This section presents the structural elements of the system, including its graph-based modeling, handling of discrete and continuous variables, solution construction, pheromone update rules, initialization procedures, and hierarchical organization of the search space.

The graph-based representation adopted in **ACOMV-CASH** departs from recent ACO-based approaches applied to machine learning [20], [21]. No archive or population of candidate solutions is maintained. Instead, the search space is structured as a directed graph (digraph), where each movement from one node to another can represent the selection of an option for discrete variables, a step in the construction of a value for continuous variables, or a hierarchical delimitation indicating which nodes are reachable. Each movement is probabilistic: the probability P_j of moving through edge j depends on the pheromone amount ω_j on that edge and the sum of pheromones on all outgoing edges, following the expression:

$$P_j = \frac{(\omega_j)^{\alpha}}{\sum_{i=1}^{k} (\omega_i)^{\alpha}} \tag{1}$$

where α is an ant-specific parameter, chosen randomly from 0.8, 1, 1.2, to modulate exploration and exploitation behaviors. Figure 2 illustrates the structure of the **ACOMV-CASH** digraph.

Regarding the construction of solutions for categorical variables, categorical options are modeled by a set of neighboring nodes $\{Q_0, Q_1, ..., Q_n\}$ connected to a common node Q. When an ant moves from Q to one of its neighbors, the corresponding

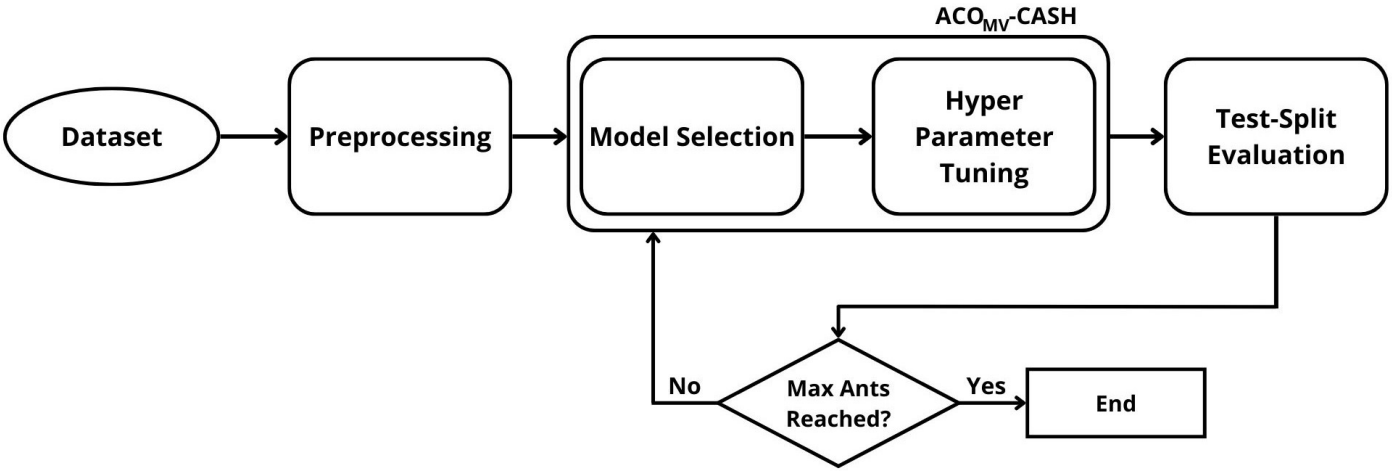

Fig. 1. Auto-ML ACOMV-CASH Flowchart

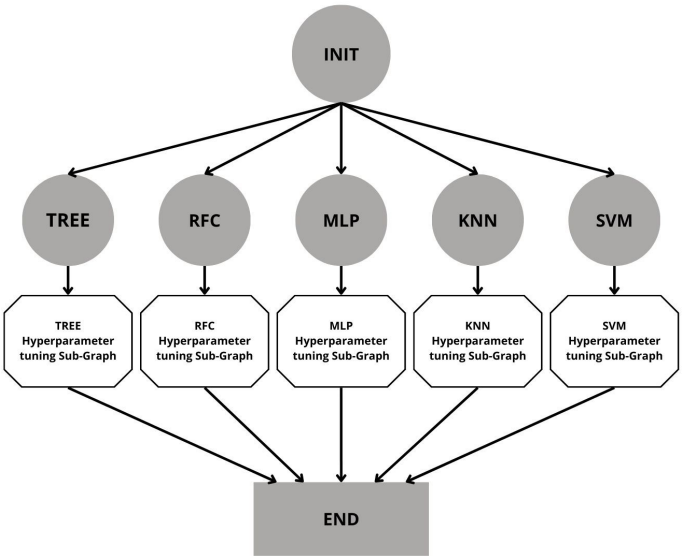

Fig. 2. **ACOMV-CASH** Digraph

categorical value is assigned to the model configuration under construction.

The solution construction for continuous variables uses a different mechanism. When a continuous parameter is required, a group of five nodes represented by $\{C_0, C_1, C_2, C_3, C_4\}$ is generated. The process starts at node C_0 with an initial value $p = \frac{p_{\max}}{2}$, where $p_{\max}$ is the upper bound of the search space. Movements among nodes C_1 to C_4 adjust the value of p incrementally or decrementally by predefined fractions of $p_{\max}$. After a series of up to five movements, the final value of the continuous variable is determined.

In the evaluation and pheromone update phase, once an ant completes its solution path, the resulting model is trained using 80% of the dataset and validated on the remaining 20%. The test accuracy serves as the optimization target. The amount of pheromone ω deposited on the traversed edges is computed by:

$$\omega(f, i) = \left(\frac{1}{1 + e^{-f+0.45}} + 0.55\right)^{20} + 3.5 \times \log_{10} i \qquad (2)$$

where f is the validation accuracy and i is the ant's index. After deposition, pheromone evaporation reduces the pheromone concentration on affected edges by 10%, balancing reinforcement and exploration.

The initialization phase precedes the main optimization process. Before regular ant traversal begins, five ants are deployed for each classifier type to explore the hyperparameter tuning subgraphs individually. Pheromones are deposited based on their performance, providing initial guidance for the full optimization cycle.

Finally, the hierarchization and search space mechanisms structure the digraph in a layered manner. Each solution must include a classifier selection variable, which defines the accessible subset of hyperparameters to be explored. This hierarchical organization reduces the effective dimensionality of the search space, improving the convergence and efficiency of the ACO process.

With the proposed architecture fully defined, the next section presents the experimental setup, evaluation methodology, and performance results obtained across multiple benchmark datasets.

IV. Experiments and Results

A. Environment

The experiments were conducted on a machine with an AMD Ryzen 7 5800H processor (16 CPUs), using Python 3.11 [22] for the ACO implementation. The implementation relied on open-source libraries such as Keras (Tensorflow 2.15) [23] and Scikit-learn 1.5.1 [24] for model construction. Finally, to manipulate the datasets and perform mathematical calculations, NumPy [25] and Pandas [26] were used.

B. ACOmv-CASH Settings

The stopping criterion for the proposed architecture is the maximum number of ants, or models evaluated, which was set to 100 for the experiments presented here. The initialization step includes five models for each classifier type, resulting in 25 evaluations before the online phase begins.

The manually designed search space for each classifier was built around the most popular and meaningful hyperparameter choices available for scikit-learn and Keras models (see Table I).

TABLE I
Detailed search spaces for each classifier

Classifier	Search Space
Multilayer Perceptron (MLP)	Number of Layers: [1 to 10] Neurons per Layer: [1 to 1024] Activation Functions: {'elu', 'relu', 'tanh'} Dropout: {None, 0.15, 0.3} Optimizers: {'adam', 'nadam', 'sgd'} Epochs: 60
Support Vector Machine (SVM)	Constant: [1 to 100] Kernel: {'linear', 'poly', 'rbf', 'sigmoid'} Degree: {2, 3, 4} Gamma: {'scale', 'auto'}
Decision Tree (TREE)	Criterion: {'gini', 'entropy', 'log_loss'} Max Depth: [3 to 80] Min Samples Split: {2, 0.1, 0.01, 0.0001}
Random Forest Classifier (RFC)	Max Features: {'log2', 'sqrt'} Number of Trees: {100, 300}
K Neighbors Classifier (KNN)	N_Neighbors: [3 to 59] Weights: {'distance', 'uniform'} Metric: {'minkowski', 'euclidean', 'manhattan', 'chebyshev'}

TABLE II
Datasets Description.

Dataset	Number of Instances	Number of Features	Number of Classes
Australian	690	14	2
Breast Cancer	569	30	2
Diabetes	768	8	2
German	1,000	20	2
Heart	270	13	2
Ionosphere	351	34	2
Sonar	208	4	2
Wine	178	13	3

TABLE III
Average performance of each framework in terms of Accuracy and Time.

Dataset	ACOmv-CASH	h2o	MLJar
Australian	**91.7% (110s)**	87.8% (1h)	82.7% (155s)
Breast Cancer	**98.2% (57s)**	97.4% (1h)	94.9% (158s)
Diabetes	**81.2% (48s)**	77% (1h)	77.9% (156s)
German	**81% (111s)**	76.1% (1h)	75.2% (148s)
Heart	**100% (55s)**	98.4% (1h)	98.4% (155s)
Ionosphere	**94.3% (48s)**	92% (1h)	**94.3% (138s)**
Sonar	**94.7% (41s)**	72.1% (1h)	77% (137s)
Wine	**100% (61s)**	100% (1h)	97.2% (70s)

C. Architecture Evaluation

The **ACOmv-CASH** was evaluated through a series of tests using eight CSV datasets: Australian Credit, German Credit, Breast Cancer, PIMA Indian Diabetes, Heart Disease, Ionosphere, Connectionist Bench (Sonar), and Wine, all available in the University of California, Irvine (UCI) Machine Learning Repository. The description of these datasets is presented in Table II.

To evaluate and compare the performance of the proposed architecture, we compare it with two widely used frameworks today, H2O [27] and Supervised-MLJAR [28]. To this end, each framework, using its default settings, was subjected to five runs, and its performance is shown in Table III.

The best framework performance for each dataset evaluation is highlighted in Table III. It can be seen that the proposed architecture not only demonstrates competitive performance but also outperforms both competitors on most datasets, tying in one dataset.

Also, the results presented in Table IV demonstrate the effectiveness of the AutoML system in not only optimizing model performance but also in selecting classifiers that align well with the intrinsic characteristics of each dataset. The diversity of best-performing models, including K-Nearest Neighbors (KNN), Multilayer Perceptron (MLP), Support Vector Classifier

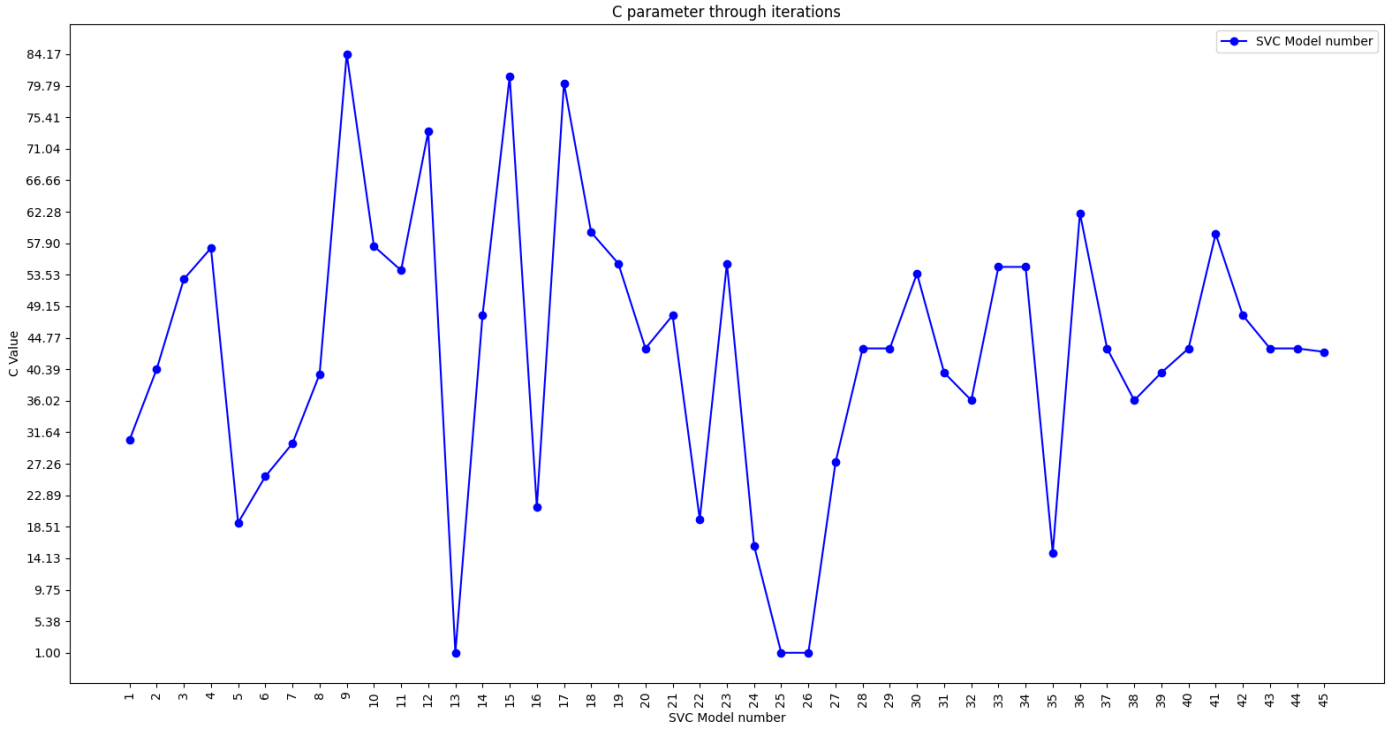

Fig. 3. C Variable Value Progression

(SVC), and Random Forest Classifier (RFC), suggests that the AutoML system could identify underlying structural patterns in the data and matching them with the most suitable modeling strategy.

TABLE IV
HIGHEST ACCURACY AND ITS CLASSIFIER FOR EACH DATASET.

Dataset	Best Model	Accuracy
Australian	KNN	93.4%
Breast	MLP	99.1%
Diabetes	SVC	83.5%
German	MLP	84.2%
Heart	RFC	100%
Ionosphere	SVC	98.5%
Sonar	RFC	100%
Wine	RFC	100%

1) Continuous Variable Construction: To demonstrate the capability of the continuous variable construction technique (Section III-B) in preserving and identifying the growth trend of continuous hyperparameters, we will focus on the constant hyperparameter C of the SVC model during training on the Ionosphere dataset. Before reaching the plateau, the value of C fluctuated between 1 and 84. Subsequently, the value stabilized and remained close to 43 (see Figure 3), along with the accuracy of the SVC (represented by the red line in Figure 4).

2) Pheromone Deposition.: The pheromone deposition equation (2) uses a sigmoid-like function that begins to rise sharply around an accuracy value of 0.6. This sharp increase improves the distinction between similar accuracy scores, ensuring that higher-performing solutions receive significantly more pheromone. However, since the plateau was not easily reached for the tested datasets, an additional component based on the ant's index, the current iteration, was introduced. This component helps later ants to add higher pheromone values across iterations, as shown in the right-hand plot of Figure 5.

The results obtained demonstrate that the **ACOMV-CASH** architecture not only delivers competitive performance compared to established market solutions, but also presents significant advantages in terms of execution time. The analysis of the pheromone deposition components, combined with the continuous modeling of variables, highlighted the effectiveness of the proposed approach in rapidly converging to high-quality solutions. These findings reinforce the potential of the technique as an efficient and robust alternative for automated model selection and hyperparameter tuning problems.

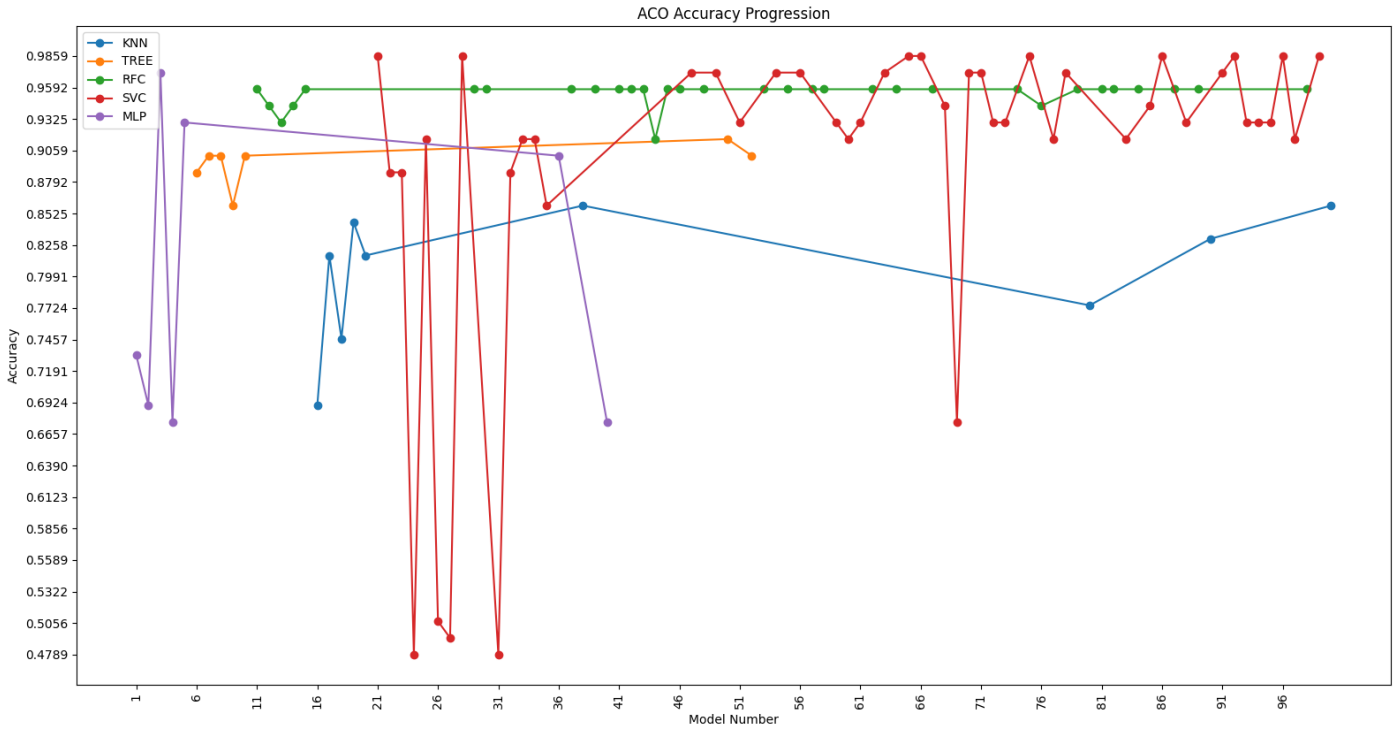

Fig. 4. **ACOMV-CASH** Model Search for the Ionosphere dataset

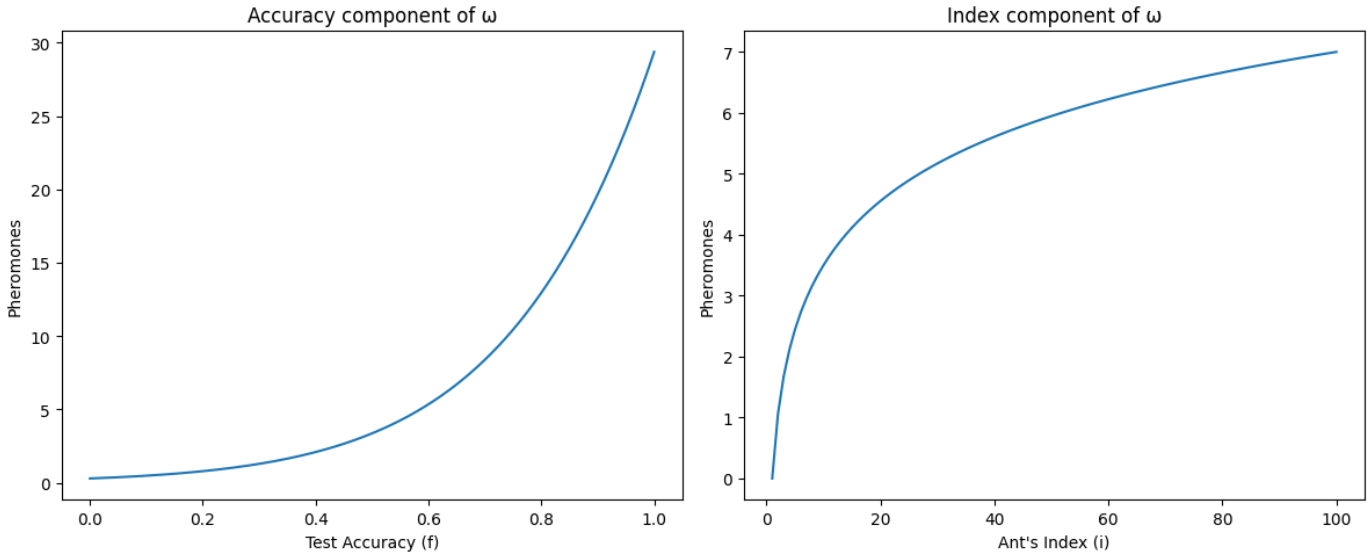

Fig. 5. Pheromone Deposition Components

V. Conclusions and Future Work

This work describes **ACOMV-CASH**, a hierarchical approach to the traditional ACO structure and the CASH problem in the context of classification AutoML. Five types of learning models (Decision Tree, RFC, SVM, MLP, and KNN) are simultaneously selected, and their hyperparameters are tuned

by the same search engine (ACO), as shown in Figure I.

The focus of this paper's contribution lies in the overall performance of the Auto Machine Learning technique, using novel approaches for the construction of continuous variables (see Section III-B), the pheromone deposition formula (see Equation 2), and the hierarchical graph structure that adapts the traditional ACO graph to handle the techniques mentioned above.

Experimental results on eight UCI datasets, in comparison to renowned frameworks, highlight the proposed architecture's robustness and low time cost, see Table III, considering the combination of traditional machine learning and deep learning in one meta-heuristic. This approach has proven to be an important aspect of the work, as MLP models performed best on two datasets (see Table IV).

This work presents a promising architecture for the CASH problem, primarily featuring implementations for model building. For future work, it is easy to foresee improvements from the framework's perspective, such as feature selection for preprocessing, stacked ensembles, and the extension of the methodology to additional domains, including image classification and regression tasks.

References

[1] F. Hutter, L. Kotthoff, and J. Vanschoren, *Automated machine learning: methods, systems, challenges*. Springer Nature, 2019.

[2] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Auto-weka: Combined selection and hyperparameter optimization of classification algorithms," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2013, pp. 847–855.

[3] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," *Advances in neural information processing systems*, vol. 28, 2015.

[4] H. Jin, Q. Song, and X. Hu, "Auto-keras: An efficient neural architecture search system," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, 2019, pp. 1946–1956.

[5] N. K. Kasabov, *Foundations of neural networks, fuzzy systems, and knowledge engineering*. Marcel Alencar, 1996.

[6] J. Waring, C. Lindvall, and R. Umeton, "Automated machine learning: Review of the state-of-the-art and opportunities for healthcare," *Artificial intelligence in medicine*, vol. 104, p. 101822, 2020.

[7] D. Whitley, "A genetic algorithm tutorial," *Statistics and computing*, vol. 4, pp. 65–85, 1994.

[8] Y. Sun, B. Xue, M. Zhang, G. G. Yen, and J. Lv, "Automatically designing cnn architectures using the genetic algorithm for image classification," *IEEE transactions on cybernetics*, vol. 50, no. 9, pp. 3840–3854, 2020.

[9] S. Nematzadeh, F. Kiani, M. Torkamanian-Afshar, and N. Aydin, "Tuning hyperparameters of machine learning algorithms and deep neural networks using metaheuristics: A bioinformatics study on biomedical and biological cases," *Computational biology and chemistry*, vol. 97, p. 107619, 2022.

[10] Y. Wang, H. Zhang, and G. Zhang, "cpso-cnn: An efficient pso-based algorithm for fine-tuning hyper-parameters of convolutional neural networks," *Swarm and Evolutionary Computation*, vol. 49, pp. 114–123, 2019.

[11] H. J. Escalante, M. Montes, and L. E. Sucar, "Particle swarm model selection." *Journal of Machine Learning Research*, vol. 10, no. 2, 2009.

[12] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, "Taking the human out of the loop: A review of bayesian optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2015.

[13] P. Liashchynskyi and P. Liashchynskyi, "Grid search, random search, genetic algorithm: a big comparison for nas," *arXiv preprint arXiv:1912.06059*, 2019.

[14] T. Domhan, J. T. Springenberg, and F. Hutter, "Speeding up automatic hyperparameter optimization of deep neural networks by extrapolation of learning curves." in *IJCAI*, vol. 15, 2015, pp. 3460–8.

[15] D. Corne, M. Dorigo, F. Glover, D. Dasgupta, P. Moscato, R. Poli, and K. V. Price, *New ideas in optimization*. McGraw-Hill Ltd., UK, 1999.

[16] T. M. Shami, A. A. El-Saleh, M. Alswaitti, Q. Al-Tashi, M. A. Summakieh, and S. Mirjalili, "Particle swarm optimization: A comprehensive survey," *Ieee Access*, vol. 10, pp. 10 031–10 061, 2022.

[17] B. Komer, J. Bergstra, and C. Eliasmith, "Hyperopt-sklearn: Automatic hyperparameter configuration for scikit-learn." in *Scipy*, 2014, pp. 32–37.

[18] I. Brezina Jr and Z. Čičková, "Solving the travelling salesman problem using the ant colony optimization," *Management Information Systems*, vol. 6, no. 4, pp. 10–14, 2011.

[19] K. Socha and M. Dorigo, "Ant colony optimization for continuous domains," *European journal of operational research*, vol. 185, no. 3, pp. 1155–1173, 2008.

[20] T. Liao, K. Socha, M. A. M. de Oca, T. Stützle, and M. Dorigo, "Ant colony optimization for mixed-variable optimization problems," *IEEE Transactions on evolutionary computation*, vol. 18, no. 4, pp. 503–518, 2013.

[21] V. O. Costa and C. R. Rodrigues, "Hierarchical ant colony for simultaneous classifier selection and hyperparameter optimization," in *2018 IEEE congress on evolutionary computation (CEC)*. IEEE, 2018, pp. 1–8.

[22] V. Rossum, "Python 3 reference manual," 2009.

[23] F. Chollet *et al.*, "Keras. https://keras.io," 2015.

[24] P. Fabian, "Scikit-learn: Machine learning in python," *Journal of machine learning research 12*, p. 2825, 2011.

[25] C. R. Harris, K. J. Millman, S. J. Van Der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith *et al.*, "Array programming with numpy," *Nature*, vol. 585, no. 7825, pp. 357–362, 2020.

[26] W. McKinney *et al.*, "Data structures for statistical computing in python." *SciPy*, vol. 445, no. 1, pp. 51–56, 2010.

[27] E. LeDell and S. Poirier, "H2o automl: Scalable automatic machine learning," in *Proceedings of the AutoML Workshop at ICML*, vol. 2020, 2020, p. 24.

[28] A. Płońska and P. Płoński, "Mljar: State-of-the-art automated machine learning framework for tabular data. version 0.10.3," Łapy, Poland, 2021. [Online]. Available: https://github.com/mljar/mljar-supervised