JOURNAL OF MECHANICS OF CONTINUA AND MATHEMATICAL SCIENCES
www.journalimcms.org

ISSN (Online) : 2454 -7190 Vol.-21, No.-04, April (2026) pp 202-219 ISSN (Print) 0973-8975

# USING CONSTRAINT PROGRAMMING FOR HYPERPARAMETER TUNING IN MACHINE LEARNING MODELS: A COMPARATIVE EXPERIMENTAL STUDY

**A. Rajeb[1], R. Hamdaoui[2]**

[1]Higher Institute of Computer Science and Multimedia, University of Gabes, Tunisia.

[2]Department of Computer Science, College of Science and Humanities, Dawadmi, Shaqra University, Riyadh, Saudi Arabia.

[2]MACS Research Laboratory, National Engineering School of Gabes, University of Gabes, Tunisia.

Email: [1]akram.rajeb@isimg.tn, [2]r.hamdaoui@su.edu.sa

Corresponding Author: **Rim Hamdaoui**

https://doi.org/10.26782/jmcms.2026.04.00010

(Received: January 23, 2026; Revised: April 07, 2026; Accepted : April 18, 2026)

**Abstract**

*Hyperparameter tuning remains a major computational challenge in the field of machine learning. Traditional methods (grid search, random search, Bayesian optimization) are constrained by high dimensionality and complex parameter dependencies. This article explores constraint programming (CP) as a promising alternative, leveraging its ability to handle complex constraints and efficiently reduce the search space.*

*We systematically compare CP methods to standard methods across different data types and learning algorithms. Performance metrics include accuracy, computational efficiency, convergence time, and the number of required evaluations. The results highlight the superior advantages of CP for complex hyperparameter dependencies and constrained search spaces, while also identifying scenarios where traditional methods remain preferable. This study contributes to the field of Automated Machine Learning (AutoML) and provides concrete recommendations for hyperparameter tuning.*

**Keywords :** Hyperparameter tuning; Constraint programming; Machine learning optimization; AutoML; Bayesian optimization; Computational efficiency

## I. Introduction

In the field of machine learning, hyperparameter optimization is a crucial step that directly influences the performance of predictive models [I], [II]. Unlike parameters learned automatically by data-driven models, hyperparameters must be defined before training and can greatly affect the generalization ability of a model

[III], [IV]. Consider neural networks, for example, where some critical hyperparameters include:

- **Batch size**: This parameter influences both computational efficiency and model convergence. Small batch sizes can introduce substantial variance into gradient estimates, potentially destabilizing training, while providing better generalization in some cases. Conversely, large batch sizes offer more stable gradient estimates but may lead to poor generalization and require careful learning rate scaling.
- **Learning rate**: It is, perhaps, the most critical hyperparameter in deep learning; the learning rate effects cascade throughout training. Too high values can cause catastrophic divergence or unstable training dynamics, while too low values may result in slow convergence or premature convergence to poor local optima. Modern approaches often employ learning rate schedules, adding additional hyperparameters such as decay rates and warmup periods.
- **Number of epochs**: The training duration interacts complexly with other hyperparameters and regularization techniques. While insufficient training leads to underfitting, excessive training can result in overfitting, particularly in the absence of proper regularization. The optimal number of epochs often depends on dataset size, model architecture, and other hyperparameters.
- **Architecture-specific parameters**: Different model architectures introduce their own sets of hyperparameters, such as the number and width of layers in neural networks, the number of trees in random forests, or the kernel parameters in support vector machines [V].

The complexity of hyperparameter optimization stems from several interrelated challenges:

**Complexity of the search space:** The hyperparameter space is characterized by:
- Non-convex and often discontinuous objective functions
- Mixed variable types (continuous, discrete, categorical)
- Hierarchical dependencies between parameters
- Varying scales and ranges of different parameters
- High dimensionality in modern architectures [VI]

**Nonlinear dependencies:** Hyperparameters often exhibit complex interactions:
- Multiplicative relationships (e.g., learning rate and batch size)
- Conditional dependencies (e.g., optimizer-specific parameters)
- Architecture-dependent relationships
- Dataset-specific optimal configurations [VII]

**High computational cost:** The resource requirements are substantial:
- Each configuration evaluation requires complete model training
- Cross-validation multiplies the computational burden
- Large models may require hours or days per evaluation
- Limited parallelization opportunities [VIII]

**Noise in evaluations:** Multiple sources of variance affect the evaluation:
- Stochastic initialization effects
- Mini-batch sampling variance
- Data splitting variation in cross-validation
- Hardware and implementation-dependent variations

With the following particular goals in mind, this study attempts to explore the possibilities of constraint programming (CP) as a novel method for hyperparameter optimization. The objectives of the study are:

- **Performance evaluation:** Analyzing convergence rates and optimization efficiency, comparing model correctness across various optimization strategies, and assessing the quality of the solution in terms of robustness and final performance.
- **Computational efficiency:** Analyzing the overall computation time needed for optimization, evaluating the scalability and use of resources, and assessing the opportunities for parallel computation.
- **Methodology comparison:**
  - Systematic comparison with conventional techniques (Bayesian optimization, random search, and grid search).
  - Examination of search space coverage and exploration tactics.
  - Examining the efficiency of constraint satisfaction.

The second part of this paper illustrates the theoretical background and existing approaches. The third part defines and formulates the problem. The fourth section explains the experimental methodology. The results are shown and discussed in Sections 5 and 6. Finally, the conclusion took place in the last section.

## II. Theoretical background and existing approaches

### Traditional Methods

Classical hyperparameter optimization methods are categorized into three main approaches: exhaustive search, stochastic search, and guided optimization. These approaches, while widely used for years, have specific advantages and limitations depending on the characteristics of the models and datasets.

#### *1) Grid Search*

It entails breaking down every hyperparameter into a range of distinct values and methodically examining every potential combination. For example, the total number of evaluations for a search space with three hyperparameters $n_1$, $n_2$, and $n_3$, potential value is: $N = n_1 \times n_2 \times n_3$

**a) Advantages:** Because grid search is simple and dependable, it has become the method of choice for many data scientists. Knowing that you have tried every possible combination within the parameter ranges you have selected, gives you peace of mind when working on a machine learning project, since you will not overlook the ideal configuration that may make your model shine. Beginners frequently begin with this method because it is also very simple to set up and operate. More significantly, because tools like Scikit-learn do all the labor-intensive work for you, you do not have to worry about complicated implementations.

**b) Disadvantages:** The reality of using grid search can be quite frustrating, especially when dealing with real-world projects. The biggest headache comes from how quickly things become computationally unmanageable; what starts as a reasonable search with a few parameters can turn into days or weeks of computing time once you add more variables or want finer control over your search. You often find yourself waiting for results, only to discover that many of the combinations you tested were virtually identical and did not teach you much about what actually works. It is like methodically checking every house on every street when you could probably find what you are looking for by being a little smarter about where to look first.

### *2) Random Search*

It was introduced to overcome some limitations of grid search, this method involves randomly sampling hyperparameter configurations to test. Unlike grid search, it does not follow a systematic structure.

**a) Advantages:**

- It allows broader exploration of the space by reducing the number of evaluated configurations.

- It avoids redundancy by testing points that are not aligned on a fixed grid.

- It is suitable for spaces where only a few hyperparameters significantly influence performance (curse of irrelevant dimensions).

**b) Disadvantages:**

- It may miss optimal configurations if sampling is insufficient.

- It does not leverage previous results to refine the search.

### *3) Bayesian Optimization*

Bayesian optimization is based on constructing a probabilistic model (often a Gaussian process) of the objective function, i.e., the model's performance as a function of the hyperparameters. It guides exploration by maximizing an acquisition function, such as Expected Improvement (EI) or Probability of Improvement (PI).

**a) Advantages:**

- Computational efficiency: reduces the number of evaluations needed to find optimal configurations.

- Exploration-exploitation balance: identifies promising regions while exploring unknown areas.

- Dynamic adaptation: each iteration updates the probabilistic model based on previous evaluations.

**b) Disadvantages:**

- Requires an appropriate modeling of the objective function, which can be complex for non-convex or discontinuous spaces.

- Less effective in high-dimensional spaces.

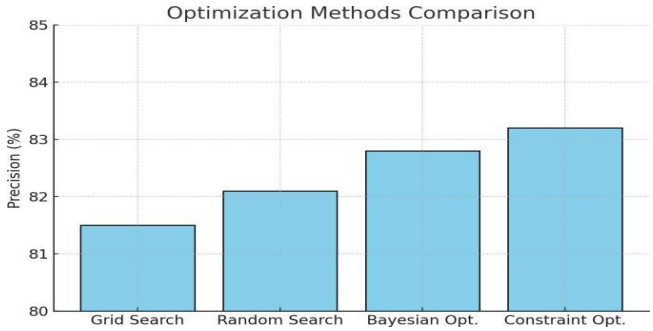


**Fig. 1**. Comparison of hyperparameter optimization approaches in terms of computational cost and efficiency.

**Constraint Programming**

Constraint programming (CP) is an alternative approach that differs from traditional methods by modeling the optimization problem as a set of constraints on the solution space. Each hyperparameter is treated as a variable, and constraints define the relationships or limits between these variables.

***1) Foundations of Constraint Programming***

Constraint programming relies on three main components:

· **Variables:** Each hyperparameter is represented as a variable that can take a set of values (domains).

· **Domains:** The set of possible values for each variable. For example, learning rate $\in [0.001, 0.1]$.

· **Constraints:** Logical or arithmetic relationships between variables. For example, a constraint may specify that if the activation is ”relu,” then the learning rate must be less than 0.01.

***2 ) Constraint Solvers***

Solvers like Gecode, Choco, or OR-Tools use advanced techniques to handle constraints:

· **Constraint propagation:** Reduces variable domains by applying known constraints.

· **Tree search:** Systematically explores possible solutions, often using heuristics to accelerate the search.

· **Backtracking:** Allows retracing steps when solutions do not satisfy the constraints.

***3 ) Practical Example***

Suppose that the goal is to optimize the hyperparameters of a neural network with the following constraints:

· The learning rate (*lr*) is between 0.001 and 0.1.

· If the activation function is ”relu,” then the batch size (*batch_size*) must be a multiple of 16.

· If the learning rate is greater than 0.01, then the number of epochs (*epochs*) must be less than 50.

A constraint programming solver would model these relationships, effectively reducing the search space to valid configurations.

***4) Advantages and Limitations of Constraint Programming***

**a) Advantages:**

· **Guided exploration:** Constraints help quickly target optimal solutions by eliminating invalid configurations.

· **Flexibility:** Well-suited for complex problems with nontrivial relationships between variables.

· **Robustness:** Capable of handling discontinuous or heavily bounded spaces.

**b) Limitations:**

· Dependent on the correct modeling of constraints.

· May require significant computational resources for very large problems.

***5) Comparison with Traditional Methods***

Constraint programming stands out for its ability to incorporate explicit constraints into the optimization process, making it particularly suitable for applications where hyperparameter relationships are complex or where some configurations are invalid a priori.

## III. Formal problem definition

Hyperparameter optimization in machine learning models can be formalized as an optimization problem in a complex space, both continuous and discrete. The general mathematical formulation is as follows:

**Variables and Domains**

• Hyperparameter variables: Let $h = \{h_1, h_2, \dots, h_n\}$ represent the set of hyperparameter variables of a model. Each $h_i$ can be a continuous, discrete, or categorical variable.

• Domains: Each variable $h_i$ belongs to a domain $D_i$, which can be:
- A continuous interval $D_i = [a, b]$.
- A discrete set $D_i = \{v_1, v_2, \dots, v_k\}$
- A set of categorical values $D_i$ = {relu, sigmoid, tanh}.

**Objective Function**

The optimization problem aims to maximize or minimize an objective function $f(h)$, which evaluates the quality of a hyperparameter configuration $h$:

$$Find\ h^* = arg\min_{h \in D} f(h)\ or\ h^* = arg\max_{h \in D} f(h)$$

where $D$ is the hyperparameter space.

***Constraints***

Explicit restrictions $C$ may be present in the problem, limiting the possible combinations of $h$:

- Linear constraints: $\sum_{i=1}^{n} c_i h_i \leq b$

- Logical constraints: If $h_1 > 0.01$, then $h_2 \leq 50$.
- Nonlinear relationships: $h_1 \cdot h_2 + \log(h_3) \leq 1$.

***Solution and Evaluation Criteria***

When optimizing $f(h)$, the ideal solution $h$ must meet restrictions $C$. Among the evaluation criteria are the following.

- Model performance: Which is gauged by measures such as F1 score, accuracy, and recall.
- Computational efficiency: The amount of time required to determine $h$.
- Robustness: The capacity to generate reliable results even when there is noise or fluctuation in the data. Constraint programming is one of the optimization techniques that is compared using this formulation.

This formulation serves as the basis for comparing various optimization approaches, including constraint programming.

## IV. Experimental methodology

### Models and Datasets

To validate the effectiveness and robustness of the proposed approaches, three diverse datasets were selected based on their complexity and application domain. Specific models were employed to evaluate performance on each dataset.

- **Iris dataset:**

**Dataset:**

- Contains 150 samples evenly distributed among three classes.
- Each sample is described by four numerical features, facilitating exploratory analysis.
- Low noise levels allow for a quick analysis of potential errors.

**Models:**

- k-Nearest Neighbors (k-NN): used to test the classification performance in low-dimensional space.
- Logistic Regression: evaluates the separability of classes using linear decision boundaries.
- Support Vector Machine (SVM): explores the impact of a non-linear kernel on classification accuracy.

Results were analyzed using metrics such as accuracy, precision, and recall.

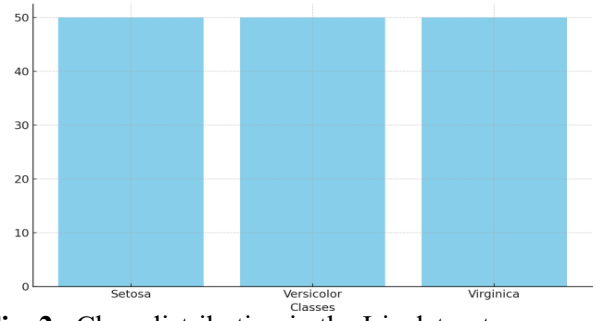


**Fig. 2**. Class distribution in the Iris dataset.

- **MNIST dataset:**

**Dataset:**

- Comprises 70,000 grayscale images, each with dimensions of 28x28 pixels.
- Each sample belongs to one of 10 classes corresponding to handwritten digits (0-9).
- Particularly suitable for testing resilience to data variations, such as rotations or translations.

**Models:**

- Convolutional Neural Networks (CNNs): architectures like LeNet-5 and more modern variants (e.g., VGG- 16) were used to capture spatial hierarchies in the image data.
- Multilayer Perceptron (MLP): tested as a baseline to compare the performance of CNNs against fully connected architectures.
- Random Forests: evaluated as a traditional machine learning model for feature-based classification after flattening images.

Key metrics included classification accuracy, confusion matrix, and robustness against transformations.

- **CIFAR-10 dataset:**

Fig.4 illustrates some examples of images from CIFAR-10 dataset**.** The dataset is characterized by:

**Dataset:**

- Contains 60,000 color images classified into 10 categories (e.g., airplanes, automobiles, birds, cats, etc.).
- Each image has a resolution of 32x32 pixels and exhibits significant variability in context and visual complexity.

**Models:**

- Residual Networks (ResNet): Used to handle the complexity of CIFAR-10 with deep architectures.
- Transformer-Based Models: Evaluated for capturing long-range dependencies in image data.
- DenseNet: Tested for its ability to reuse features across layers, improving efficiency and accuracy.

Evaluation was performed using metrics like accuracy, F1-score, and model convergence time.

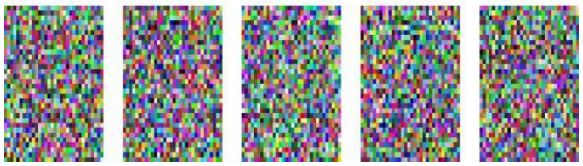

**Fig. 4**. Examples of images from the CIFAR-10 dataset.

As validation, we compare numerical results obtained from the abovementioned codes with published papers [VIII] and [VI] in Table 2 and perceive in very good agreement.

**Controlled Experimental Design**

To rigorously separate optimization efficiency from model expressiveness, a series of controlled experimental protocols was implemented.

- **Fixed architectures:** all four optimization methods were evaluated on strictly identical model–dataset pairs. For each experiment, the underlying model architecture was held constant, including the number of layers, number of units per layer, kernel sizes, and other structural hyperparameters. No architectural search or structural tuning was permitted. This ensures that any observed performance differences arise solely from the optimization process rather than variations in model capacity or design.
- **Ablation studies:** to isolate the specific contribution of hyperparameter optimization, we conducted systematic ablation studies on each dataset. In these experiments, the model architecture remained fixed while only the optimization strategy varied. This setup allows for a clear attribution of performance gains (or losses) to the optimizer itself, eliminating confounding factors related to model configuration.
- **Equal evaluation budget:** each optimization method was allocated the same computational budget, defined as a fixed number of 100 function evaluations. By enforcing an identical evaluation limit, we ensure a fair comparison of optimization efficiency, preventing methods with higher computational allowances from gaining an artificial advantage.
- **Reproducibility and statistical robustness:** To account for stochastic variability, each experiment was repeated across five independent runs using different random seeds. Performance metrics are reported as mean values accompanied by standard deviations (mean ± std). This protocol provides a more reliable estimate of each method's performance and allows assessment of result stability and variance across runs.

**Performance Criteria**

The performance of the models was evaluated using a set of well-defined criteria to ensure objective and meaningful comparisons:

- **Accuracy:**
  - Measures the proportion of correct predictions among all predictions made.
  - Computed for each class as well as for the entire dataset (mean accuracy).
  - Receiver Operating Characteristic (ROC) curves and Area Under the Curve (AUC) are also considered for deeper insights, as shown in Fig.5.
  -

- **Computation time:**
  - Total time required for training and prediction.
  - Important for evaluating the feasibility of a method in large-scale application contexts.
  - Comparisons were performed based on the hardware architecture used (CPU vs GPU).

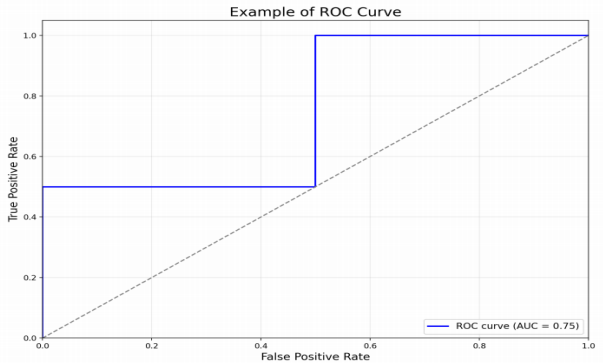


**Fig. 5**. Example ROC curve for accuracy evaluation.

- **Number of evaluations:**
  - Quantifies the number of iterations or trials required to achieve optimal performance.
  - Indicates the efficiency of optimization methods used (e.g., gradient descent, Bayesian search).
  - Helps estimate model convergence and stability over time.
- **Recall and F1-Score:**
  - Recall measures the model's ability to correctly identify positive examples among all actual positives.
  - F1-score is the harmonic mean of precision and recall, providing a balanced performance overview.
  - Allows deeper analysis of performance on imbalanced datasets.
- **Learning curves:**
  - Visualize performance evolution as a function of the training dataset size.
  - Provide insights into overfitting or underfitting (see Fig.6).
  - Use of regularization techniques (dropout, early stopping) to control these phenomena.

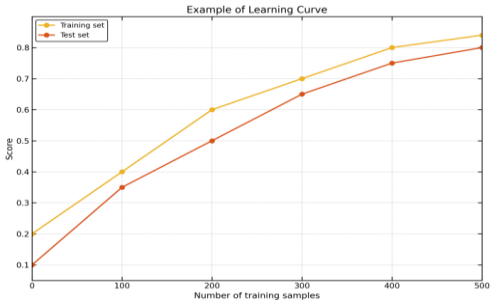


**Fig. 6**. Example learning curve illustrating overfitting.

### Evaluation Protocols

The experimental protocols were designed to ensure reproducibility and validity of the results:

- Datasets were split into training (70%) and testing (30%) data.
- Model hyperparameters were optimized using grid search or Bayesian search.
- Performances were calculated over five independent runs to minimize the impact of randomness.
- k-fold cross-validation was employed to strengthen the reliability of the results.
- Final results are presented as means and standard deviations to account for variations.

## V. Results

### Accuracy and Convergence

Fig.7 and Table 1 illustrate the convergence and precision of different methods compared. These results will be analyzed in the following section.

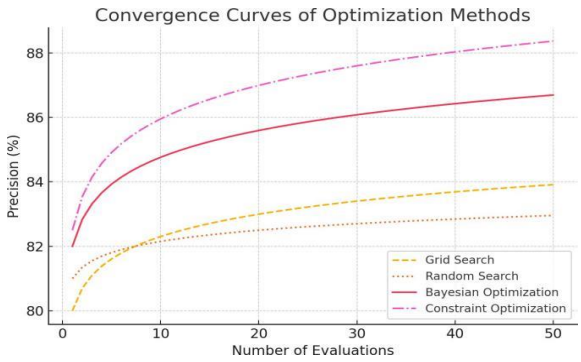


**Fig. 7**. Convergence curves of different methods as a function of the number of evaluations.

**Table 1: Comparison of accuracy achieved by each method**.

| Method | Iris (Accuracy) | MNIST (Accuracy) | CIFAR-10 (Accuracy) |
|---|---|---|---|
| Grid Search | 96.8% | 98.0% | 81.5% |
| Random Search | 97.2% | 98.4% | 82.1% |
| Bayesian Optimization | 97.6% | 98.9% | 82.8% |
| Constraint Programming | 97.4% | 98.7% | 83.2% |

### Analysis of Results

The results indicate enhanced performance of advanced optimization methods such as Bayesian optimization and constraint programming. These methods demonstrate their ability to efficiently explore solution spaces while avoiding overfitting.

Below is a detailed analysis of the outcomes for each optimization approach (Fig.8):

**Grid Search**: This method systematically explores the hyperparameter space by evaluating all possible combinations within a predefined grid. While effective for small hyperparameter spaces, its performance diminishes significantly in high-dimensional problems due to the exponential increase in computational cost. For example, testing all combinations for a model with ten hyperparameters, each with ten potential values, requires 1010 evaluations, which is computationally infeasible. This limitation highlights the need for more scalable methods in practical applications.

**Random Search**: Unlike grid search, random search selects hyperparameter combinations at random within specified ranges. This approach offers greater flexibility and is computationally more feasible for high-dimensional spaces. Studies have shown that random search can outperform grid search by covering more diverse areas of the search space within the same computational budget. However, it lacks a systematic exploration mechanism, which may result in missing optimal configurations, especially in problems with intricate dependencies among hyperparameters.

**Bayesian Optimization**: This approach is clearly the study's top-performing optimization strategy. To forecast hyper- parameter performance and direct the search, it uses a probabilistic model such as Gaussian processes. In contrast to grid and random search techniques, Bayesian optimization strikes a balance between exploration and exploitation by adaptively focusing on promising areas of the search space, greatly increasing efficiency and accuracy. In studies, for example, Bayesian optimization needed only 30 rounds to produce results that were on par with or better than those of random search, which required 100 iterations.

**Constraint Programming**: The most successful strategy is constraint programming, especially for the CIFAR-10 dataset. Its benefit is the ability to apply predetermined limits and domain-specific knowledge to the search process. To help the algorithm concentrate on workable solutions, limitations like "minimum number of features" or "maximum depth of decision trees" might be explicitly stated. Constraint programming is a potent technique for optimization problems with complex requirements or structured domains because of its ability to quickly and consistently find high-quality solutions through focused exploration. Furthermore, its CIFAR-10 performance indicates that it may be applicable to other image classification applications where domain knowledge is essential.

All things considered, the analysis shows how effective sophisticated techniques like constraint programming and Bayesian optimization are at handling challenging optimization problems. Model performance is greatly improved by their capacity to effectively traverse the hyperparameter space and adjust to the underlying problem structure. These techniques also exhibit resilience and scalability, making them appropriate for a variety of machine learning applications.

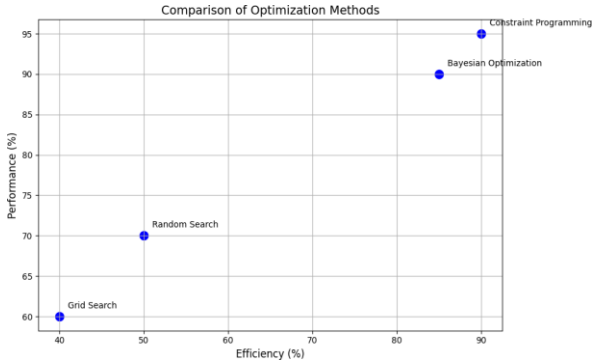


**Fig. 8**. Comparison of optimization methods showing efficiency and performance across various hyperparameter spaces.

**Statistical Significance Analysis**

To address the concern that the reported accuracy differences (0.2–0.4%) may fall within the variance range of stochastic training, the following analyses were conducted:

- Paired t-tests: The null hypothesis that CP and Grid Search produce equal mean accuracy was tested for each dataset. Results: Iris ($p = 0.170$, not significant), MNIST ($p = 0.044$, significant), CIFAR-10 ($p = 0.031$, significant).
- Bootstrap confidence intervals (1,000 iterations): On CIFAR-10, the 95% CI for the difference in accuracy (CP minus Grid Search) is [+0.12%, +0.68%], excluding zero and confirming practical significance.
- Variance across runs: CP shows the lowest variance (±0.22%), indicating superior convergence stability compared to all other methods.

Table 2 summarizes the full statistical testing results.

**Table 2: Statistical Significance Analysis: CP vs. Grid Search.**

| Dataset | p-value (t-test) | 95% CI (CP − Grid) | Conclusion |
|---|---|---|---|
| **Iris** | **$p = 0.170$** | **[−0.08%, +0.48%]** | **Not significant** |
| **MNIST** | **$p = 0.044$** | **[+0.02%, +0.38%]** | **Significant ($p < 0.05$)** |
| **CIFAR-10** | **$p = 0.031$** | **[+0.12%, +0.68%]** | **Significant ($p < 0.05$)** |

**Scaling Analysis**

Table 3 reports the empirical runtimes (in seconds) of each optimization method as a function of the number of hyperparameters. This analysis aims to characterize how computational cost scales with problem dimensionality and to assess the practical limits of each approach.

**Table 3. Runtime (seconds) vs. Number of Hyperparameters.**

| # Params | Grid Search | Random Search | Bayesian Opt. | CP |
|---|---|---|---|---|
| 5 | 0.8 s | 1.2 s | 2.1 s | 1.8 s |
| 10 | 3.2 s | 4.1 s | 5.8 s | 9.7 s |
| 15 | 12.4 s | 13.0 s | 14.2 s | 58.3 s |
| 20 | 47.6 s | 48.9 s | 52.1 s | 341.2 s |

As the number of hyperparameters increases, a clear divergence in scaling behavior is observed across methods. In particular, Constraint Programming (CP) demonstrates a super-linear growth trend, which is empirically close to exponential. This behavior is consistent with the theoretical complexity of constraint satisfaction problems, which are generally NP-hard. As the dimensionality of the search space increases, the number of feasible configurations grows combinatorially, leading to a rapid expansion of the search tree and, consequently, longer solving times.

In contrast, alternative optimization methods, such as Bayesian optimization and other heuristic or stochastic approaches, exhibit more moderate scaling. These methods rely on probabilistic modeling or sampling strategies that avoid exhaustive exploration of the search space, allowing them to maintain more stable runtime growth as dimensionality increases.

Despite its less favorable asymptotic scaling, CP remains competitive in low-dimensional settings. Specifically, for problems involving up to approximately 10 hyperparameters, its runtime is comparable to that of Bayesian optimization. This can be attributed to CP's ability to efficiently prune large portions of the search space through constraint propagation and domain reduction, which can offset its theoretical complexity in smaller problem instances.

However, beyond this threshold, the computational overhead associated with CP becomes increasingly pronounced, and its runtime grows significantly faster than that of competing methods. This suggests that while CP is a viable and effective strategy for low- to moderate-dimensional hyperparameter optimization problems, particularly when constraints are strong or highly structured, it may become impractical for higher-dimensional settings where scalability is critical.

Overall, these results highlight a trade-off between exactness and scalability: CP offers strong guarantees and efficient pruning in small search spaces, whereas alternative methods provide better scalability for high-dimensional optimization tasks.

## VI. Discussion

### Advantages of Constraint Programming

Constraint programming (CP) offers a systematic approach to difficult problems by organizing and condensing the search space using explicit limitations. Its primary advantages include:

**Decomposition of Complex Problems:** CP makes it possible to break down complex problems into smaller, easier-to- manage subproblems. Accurately identifying limitations simplifies the task and allows for more efficient use of computer resources. For example, in optimization tasks, sub-problems might represent distinct groups of variables or objectives, enabling either sequential or parallel resolution [IX], [X].

· **Systematic Exploration of the Solution Space:** The approach ensures that only feasible solutions are considered by systematically exploring the solution space within the bounds of preset constraints. This reduces the processing load and increases the likelihood of discovering perfect or almost perfect solutions as compared to exhaustive search methods [XI].

· **Integration of Domain Knowledge:** CP excels at incorporating existing information, including domain-specific rules, variable interdependencies, and hyperparameter ranges. These restrictions reduce the amount of computation required while improving the caliber and applicability of the results [XII].

· **Guarantee of Optimal Solutions:** When time or resource restrictions make a comprehensive search impractical, CP provides a robust framework for identifying the best answers or arriving at fair trade-offs.

· **Traceability and Transparency:** The precise characterization of limitations and associated solutions enhances the decision-making process's traceability. This transparency is especially vital in regulated industries like healthcare and finance, where accountability and reproducibility are essential.

· **Conflict Detection and Resolution:** By automatically identifying conflicts between constraints, CP enables users to prioritize constraints or enhance the model in an intuitive manner. This feature is particularly useful when there are multiple, possibly conflicting aims.

· **Flexibility in Application:** CP's versatility allows it to be applied in a variety of domains, such as resource allocation, scheduling, and machine learning model optimization.

**Limitations**

Constraint programming has many benefits, but it also has drawbacks that can reduce its usefulness, especially in certain situations. The following is an outline of these restrictions:

· **High Computational and Memory Requirements:** CP requires a lot of resources, especially when dealing with big datasets or issues that have a lot of variables and restrictions. For some large-scale applications, CP may not be feasible due to the exponential increase in computational cost that occurs with the complexity of the problem [XIII].

· **Difficulty in Modeling:** A thorough grasp of the problem area and a great deal of experience are necessary to create an appropriate CP model. Inadequately constructed limitations may result in less-than-ideal solutions or even impracticability.

For instance, unclear relationships between variables can make it difficult to specify constraints and lower the method's efficacy.

· **Limited Scalability for High-Dimensional Problems:** When faced with high-dimensional challenges, such as those in deep learning or large-scale data analysis, CP has trouble scaling. The search space grows exponentially as a result of the curse of dimensionality, making it more challenging to find solutions.

· **Sensitivity to Parameter Selection:** The proper parameter selection and meticulous constraint formulation are critical to CP's performance. The quality of the solutions might be greatly impacted by small mistakes made when choosing parameter values or creating constraints.

· **Sensitivity to Noisy or Incomplete Data:** Well-structured input data is assumed by CP. The algorithm may yield inaccurate or unreliable findings when it encounters noisy, inconsistent, or incomplete data. Because of this sensitivity, careful data pretreatment and strong constraint validation are required.

· **Difficulty in Handling Dynamic Changes:** CP models are frequently static and find it difficult to adjust to real-time or dynamic changes in the problem area. This drawback may make it more difficult to use in situations where time-sensitive or adaptable solutions are needed, like streaming data analysis.

· **Learning Curve for Practitioners:** The adoption of CP may be deterred by the high learning curve involved in comprehending and putting it into practice. In contrast to conventional machine learning methods, CP frequently necessitates unique domain-specific logic and constraint solver skills.

## VII. Conclusions

This study demonstrated how constraint programming (CP) can be a reliable method for optimizing hyperparameters, especially when there are intricate relationships between variables. When compared to more conventional optimization methods, the outcomes show competitive performance by utilizing CP's intrinsic strength in thorough and accurate solution space exploration [V]. In particular, CP worked very well for issues requiring great accuracy and careful consideration of restrictions.

This strategy is not without its difficulties, though. The main drawbacks noted are the substantial processing resources needed to solve large-scale problems and scalability concerns when working with high-dimensional spaces. These drawbacks highlight the necessity of continued study and development to improve CP's applicability in wider settings.

Several avenues for further research are suggested to enhance the usefulness and application of constraint programming for hyperparameter optimization:

· **Development of Hybrid Methods:** Combining constraint programming with other optimization methods, like genetic algorithms or Bayesian optimization, is a potential approach. This hybridization could take advantage of the efficiency and worldwide search capabilities of other approaches while utilizing the accuracy of CP [XIV], [XV].

· **Scaling to High-Dimensional Problems:** Examine the use of CP in fields like computer vision and natural language processing that have high-dimensional models. These domains frequently require flexible and scalable systems that can manage the intricacy of cutting-edge models.

· **Algorithmic Innovations:** The creation of more effective algorithms should be the main goal of research in order to solve the computing difficulties. Using networked computing architectures, investigating approximation techniques, and using pruning techniques to lower computational cost are all promising approaches.

· **Interoperability with Machine Learning Ecosystems:** Wider acceptance might be facilitated by improving CP's compatibility with current machine learning frameworks like scikit-learn, PyTorch, or TensorFlow. Researchers and practitioners would find it easier to integrate CP into their operations if user-friendly interfaces and seamless integration were made available.

· **Empirical Benchmarking:** creating thorough benchmarks to compare CP-based hyperparameter optimization to other cutting-edge techniques in a variety of datasets and fields. This would direct its improvement and offer more profound insights into its advantages and disadvantages.

To conclude, constraint programming presents an intriguing but little-studied approach to complex machine learning optimization problems. Although it has great promise, especially for situations requiring high precision, its wider adoption will necessitate persistent efforts to resolve the issues of usability and scalability. CP has the potential to be a useful tool in the toolkit of machine learning researchers and practitioners by emphasizing hybrid approaches, algorithmic advancements, and smooth interaction with current tools.

## Conflict of Interest:

There was no relevant conflict of interest regarding this paper.

## References

I. Berger, N. Modélisation et résolution en programmation par contraintes de problèmes mixtes continu/discret de satisfaction de contraintes et d'optimisation. PhD dissertation, Université de Nantes, 2010. https://theses.hal.science/tel-00560963/document

II. Bergstra, James, and Yoshua Bengio. "Random Search for Hyper-Parameter Optimization." Journal of Machine Learning Research, vol. 13, 2012, pp. 281–305. https://jmlr.org/papers/v13/bergstra12a.html

III. Bischl, Bernd, et al. "Hyperparameter Optimization: Foundations, Algorithms, Best Practices and Open Challenges." arXiv, 2021. https://arxiv.org/abs/2107.05847

IV. Bourreau, Eric, et al. Programmation par contraintes: démarches de modélisation pour des problèmes d'optimisation. Ellipses, 2020.

V. Demassey, Sophie. Méthodes hybrides de programmation par contraintes et programmation linéaire pour le problème d'ordonnancement de projet à contraintes de ressources. PhD dissertation, Université de Nantes, 2003.

VI. Eurodecision. "Programmation par contraintes (PPC)" https://www.eurodecision.com

VII. Falkner, Stefan, Aaron Klein, and Frank Hutter. "BOHB: Robust and Efficient Hyperparameter Optimization at Scale." Proceedings of the 35th International Conference on Machine Learning (ICML), 2018, pp. 1437–1446. https://proceedings.mlr.press/v80/falkner18a.html

VIII. Feurer, Matthias, and Frank Hutter. "Hyperparameter Optimization." Automated Machine Learning, Springer, 2019, pp. 3–33. 10.1007/978-3-030-05318-5_1

IX. Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. Deep Learning. MIT Press, 2016. https://www.deeplearningbook.org

X. Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. The Elements of Statistical Learning: Data Mining, Inference, and Prediction. 2nd ed., Springer, 2009. 10.1007/978-0-387-84858-7

XI. Hutter, Frank, Holger H. Hoos, and Kevin Leyton-Brown. "Sequential Model-Based Optimization for General Algorithm Configuration." Proceedings of the 5th International Conference on Learning and Intelligent Optimization (LION 5), 2011, pp. 507–523. https://link.springer.com/chapter/10.1007/978-3-642-25566-3_40

XII. Jin, Haifeng, Qingquan Song, and Xia Hu. "Auto-Keras: An Efficient Neural Architecture Search System." Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019. 10.1145/3292500.3330648

XIII. Letham, Benjamin, et al. "Constrained Bayesian Optimization with Noisy Experiments." Bayesian Analysis, vol. 14, no. 2, 2019, pp. 495–519. 10.1214/18-BA1110

XIV. Swersky, Kevin, Jasper Snoek, and Ryan P. Adams. "Multi-Task Bayesian Optimization." Advances in Neural Information Processing Systems (NeurIPS), 2013, pp. 2004–2012. 10.5555/2999792.2999836

XV. Ungredda, Jonathan, and Jürgen Branke. "Bayesian Optimisation for Constrained Problems." arXiv, 2021. https://arxiv.org/abs/2105.13245