Optimization of Grid Search for Classification Model Tuning Using Branch and Bound

Michael Alexander Angkawijjaya - 13523102
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail angkawijayamichael@gmail.com , 13523102@std.stei.itb.ac.id

Abstract*—Hyperparameter optimization is a critical yet computationally demanding stage in the development of high-performance machine learning models. Grid Search, a foundational technique, guarantees finding the optimal hyperparameter configuration within a specified discrete grid but suffers from an exponential increase in computational cost, rendering it impractical for large search spaces. This paper introduces a novel deterministic optimization algorithm, Branch-and-Bound Grid Search (B&B-GS), designed to accelerate this process without sacrificing optimality. B&B-GS reframes the hyperparameter tuning problem as a combinatorial optimization task, amenable to the Branch and Bound paradigm. By conceptualizing the hyperparameter grid as a state-space tree, our algorithm intelligently prunes unpromising regions of the search space, thereby avoiding exhaustive evaluation. The core of our contribution is a novel bounding function that leverages the principle of Lipschitz continuity of the model's performance landscape. This function allows for the calculation of a rigorous upper bound on the performance within any sub-grid, enabling effective pruning decisions. Experimental results on benchmark classification tasks demonstrate that B&B-GS significantly reduces the number of required model evaluations compared to exhaustive Grid Search while consistently identifying the same optimal hyperparameter set. This work presents a deterministic alternative to stochastic methods, offering a compelling trade-off between computational efficiency and the guarantee of optimality on the grid.

Keywords*—Hyperparameter Optimization, Grid Search, Branch and Bound, Classification Models, Pruning, Lipschitz Optimization, Machine Learning.

I. Introduction

The predictive power and generalization capability of modern machine learning models are not solely determined by the underlying algorithm or the quality of the training data; they are profoundly influenced by a set of configuration settings known as hyperparameters. These parameters, which include learning rates, regularization strengths, and architectural choices like the number of layers in a neural network, are set prior to the commencement of the training process and govern its behavior. The process of identifying the optimal set of hyperparameters, known as Hyperparameter Optimization (HPO), is a crucial step in the machine learning pipeline. A well-tuned model can exhibit a dramatic improvement in performance metrics, often marking the difference between a model with mediocre utility and one that achieves state-of-the-art results.

Among the plethora of HPO techniques, Grid Search (GS) stands as the most traditional and conceptually straightforward method. It operates by performing an exhaustive search over a manually specified, discrete subset of the hyperparameter space. The appeal of Grid Search lies in its deterministic nature, its inherent parallelism, and its guarantee of finding the best possible combination of hyperparameters *within the confines of the defined grid*. However, this exhaustive approach is also its greatest weakness. The number of configurations to evaluate grows exponentially with the number of hyperparameters, a phenomenon widely known as the "curse of dimensionality".[1] This exponential complexity renders Grid Search computationally intractable for all but the most trivial search spaces, making it a significant bottleneck in practical machine learning workflows.

The prohibitive cost of Grid Search has catalyzed the development and widespread adoption of more efficient, often stochastic, alternatives. Random Search, for instance, has been shown to frequently outperform Grid Search by randomly sampling configurations from the hyperparameter space. Its efficiency stems from the empirical observation that model performance is often sensitive to only a few hyperparameters, and random sampling is more likely to explore a wider range of values for these critical parameters given the same computational budget. Further advancing the field, Bayesian Optimization has emerged as a powerful technique that employs probabilistic surrogate models to intelligently navigate the search space. By balancing the exploration of uncertain regions with the exploitation of known high-performing areas, Bayesian methods can achieve superior sample efficiency, finding better hyperparameter configurations in fewer evaluations.

Despite the efficiency of these advanced methods, they trade the deterministic guarantee of Grid Search for speed. Stochastic approaches like Random Search and Bayesian Optimization do not assure the discovery of the true optimal configuration within the search space; their success can be contingent on the random seed or the initial set of evaluated points. This introduces an element of uncertainty and

irreproducibility into the modeling process. This trade-off motivates a central research question: *Can the deterministic guarantee of finding the grid's optimal point, a hallmark of Grid Search, be retained while achieving a computational efficiency that approaches that of stochastic methods?*

This paper addresses this research gap by proposing a novel algorithm: the Branch-and-Bound Grid Search (B&B-GS). Our approach reframes HPO as a combinatorial optimization problem, making it amenable to the classic Branch and Bound (B&B) algorithm paradigm. We conceptualize the hyperparameter grid as a state-space tree that can be systematically explored. The core innovation of our work lies in the development and application of a bounding function that enables the algorithm to prune vast, unpromising regions of the search space without evaluating every point. This bounding function is derived from the assumption of **Lipschitz continuity** of the model's performance landscape—a reasonable assumption for many machine learning models where small changes in hyperparameters typically result in correspondingly small changes in performance. By adaptively estimating a Lipschitz constant for the performance function, we can compute a rigorous upper bound on the best possible score within any sub-grid, allowing for efficient and reliable pruning. This work distinguishes itself from prior applications of B&B in machine learning, which have focused on areas like feature selection, model-specific integer programming formulations, or neural network verification, by directly targeting the general and ubiquitous problem of hyperparameter grid tuning.

The remainder of this paper is organized as follows. Section II provides a detailed background on existing HPO strategies and the Branch and Bound algorithm, situating our work within the broader context of related research. Section III formally presents the proposed B&B-GS algorithm, including the formulation of the bounding function. Section IV describes the experimental setup designed to validate our method against established baselines. Section V presents and analyzes the results of our empirical evaluation. Finally, Section VI concludes the paper with a summary of our contributions, a discussion of the method's limitations, and directions for future research.

II. Hyperparameter Optimization Strategies

Hyperparameter optimization can be viewed as the problem of finding a set of hyperparameters λ^* from a search space Λ that minimizes an objective function $f(\lambda)$, which typically represents the validation error or maximizes a performance metric like accuracy. The function f is a black box, as its value can only be determined by the costly process of training and evaluating a model.

A. Deterministic Exhaustive Search: Grid Search

Grid Search is the most established and straightforward HPO technique.[1] The algorithm operates on a search space defined as the Cartesian product of a set of user-specified, discrete values for each hyperparameter: $\Lambda = \Lambda_1 \times \Lambda_2 \times \ldots \times \Lambda_k$, where Λ_i is the list of values for the i-th hyperparameter. Grid Search then proceeds to train and evaluate a model for every single configuration $\lambda \in \Lambda$. The performance of each configuration is typically assessed using k-fold cross-validation on the training data to obtain a robust estimate of its generalization ability. After all configurations have been evaluated, the algorithm returns the one that yielded the best average score across the cross-validation folds.

The primary advantages of Grid Search are its simplicity and transparency. Because it evaluates every point on the grid, it is deterministic and guarantees that the optimal configuration *within that specific grid* will be found. This exhaustive nature makes results perfectly reproducible, a feature that is highly valued in academic research and in regulated industries where process validation is critical. Furthermore, the evaluation of each hyperparameter configuration is an independent task, making Grid Search an "embarrassingly parallel" problem. This allows for significant speedups on multi-core systems or distributed computing clusters, often implemented with a simple parameter like $n\_jobs = -1$ in popular libraries such as scikit-learn.

The determinism of Grid Search comes at a steep price. Its computational complexity, which can be expressed as $O(|\Lambda_1| \times |\Lambda_2| \times \ldots \times |\Lambda_k| \times \mathrm{n}_{folds} \times \mathrm{T}_{train})$, where T_{train} is the time for a single model training, grows exponentially with the number of hyperparameters (k). This "curse of dimensionality" makes the method computationally prohibitive for search spaces involving more than a handful of hyperparameters or a fine-grained discretization of their values. Consequently, practitioners are often forced to use very coarse grids, increasing the risk of missing the true optimal region of the hyperparameter space entirely.

B. Stochastic Search Methods

To overcome the limitations of Grid Search, several stochastic methods have been developed that trade exhaustive guarantees for computational efficiency.

1. Random Search

Proposed as a direct and surprisingly effective alternative to Grid Search, Random Search samples a fixed number of configurations at random from the hyperparameter space. Instead of discrete lists, the search space for each hyperparameter can be defined by a statistical distribution (e.g., uniform, log-uniform), allowing for a much finer exploration of continuous parameters. The effectiveness of Random Search is rooted in the empirical finding that for many models, only a few hyperparameters have a significant impact on performance. Grid Search wastes many evaluations by testing numerous combinations of unimportant parameters, whereas Random Search, for the same computational budget, explores more unique values for each individual parameter. This increases the probability of finding a good setting for the truly important ones. While highly efficient and easily parallelizable, Random Search is a stochastic process. It provides no guarantee of finding the optimal configuration, and its results can vary significantly between runs with different random seeds.

2. Bayesian Optimization

This technique represents the state-of-the-art in sample-efficient HPO. It falls under a broader class of algorithms known as Sequential Model-Based Optimization (SMBO). The core idea is to treat the expensive black-box objective function $f(\lambda)$ as an unknown function that can be approximated. Bayesian Optimization proceeds iteratively:

1. **Build a Surrogate Model:** It uses the history of evaluated $(\lambda, f(\lambda))$ pairs to build a probabilistic surrogate model that approximates the true objective function. Gaussian Processes (GPs) are a common choice for the surrogate, as they can provide not only a mean prediction for the performance at an unevaluated point but also an estimate of the uncertainty around that prediction.
2. **Use an Acquisition Function:** An acquisition function is used to guide the search for the next point to evaluate. This function leverages the surrogate's predictions and uncertainty estimates to quantify the "value" of evaluating a particular configuration. A popular choice is Expected Improvement (EI), which calculates the expected amount of improvement over the best solution found so far. The acquisition function naturally balances *exploitation* (sampling in regions where the surrogate predicts high performance) and *exploration* (sampling in regions with high uncertainty, where a surprisingly good result might be found).
3. **Select Next Point and Update:** The configuration that maximizes the acquisition function is chosen for the next evaluation. The result is then used to update the surrogate model, and the process repeats.

By intelligently selecting which points to evaluate, Bayesian Optimization can often find superior hyperparameter configurations in far fewer iterations than Grid Search or Random Search. However, it is inherently sequential, making parallelization less straightforward. Furthermore, standard implementations can struggle with high-dimensional or complex (e.g., conditional) search spaces and can be more complex to implement correctly.

III. The Branch & Bound Optimization

A. Branch & Bound Fundamental

Branch and Bound is a fundamental algorithm design paradigm for solving NP-hard discrete and combinatorial optimization problems, such as the Traveling Salesman Problem or Integer Linear Programming. Its power lies in its ability to find a provably optimal solution without performing a full exhaustive search. It achieves this through a "divide and conquer" strategy that systematically prunes large portions of the search space that are guaranteed not to contain the optimal solution.

The algorithm relies on three canonical components:

1. **Branching:** This is the process of recursively partitioning the problem's feasible solution set into smaller, disjoint subsets. This partitioning creates a state-space search tree, where the root node represents the entire original problem, and each child node represents a subproblem corresponding to a restricted subset of the solution space. For example, in an integer programming problem, branching might involve selecting a variable with a fractional value in a relaxed solution and creating two new subproblems: one where the variable is constrained to be less than or equal to the floor of the value, and another where it is constrained to be greater than or equal to the ceiling.
2. **Bounding:** This is the most critical component of the algorithm. For each node (subproblem) in the search tree, a bound on the value of the objective function is computed. For a maximization problem, this involves calculating an **upper bound**—an optimistic estimate of the best possible solution that could be found within that node's subspace. Conversely, for a minimization problem, a **lower bound** is calculated. These bounds are often obtained by solving a "relaxation" of the subproblem, which is an easier-to-solve version of the problem with some constraints removed. For instance, the Linear Programming (LP) relaxation of an Integer Program (IP), where integer constraints are ignored, provides a valid upper bound for a maximization problem.
3. **Pruning:** This step is the source of B&B's computational efficiency. The algorithm maintains a record of the best feasible solution found so far, known as the "incumbent." The bound calculated for each new node is compared against the value of the incumbent. For a maximization problem, if a node's upper bound is less than or equal to the value of the incumbent solution, that node (and the entire subtree rooted at it) can be safely discarded, or "pruned". This is because no solution in that entire region of the search space can possibly be better than the solution already found. Pruning can also occur if a subproblem is found to be infeasible or if its relaxation yields a feasible integer solution that updates the incumbent. The algorithm terminates when the queue of active (unexplored) nodes is empty, at which point the current incumbent is the proven global optimum.

B. Formalizing HPO as a B&B Problem

To apply the Branch and Bound algorithm, we must first define the HPO problem in terms of its fundamental components: a search space, an objective function, and a state-space tree structure.

- **Search Space:** The search space for our problem is the set of all possible hyperparameter configurations, Λ, as defined by the user's grid. This space is discrete, finite, and structured as a multi-dimensional grid. For instance, if we are tuning two hyperparameters, C and γ, with 10 values each, the search space Λ consists of the 100 discrete points forming the 10×10 grid.
- **Objective Function:** The objective function, which we aim to maximize, is denoted by $f(\lambda)$. This function takes a hyperparameter configuration $\lambda \in \Lambda$ as input and returns a performance score, such as the mean accuracy obtained from k-fold cross-validation. A critical characteristic of $f(\lambda)$ is that it is a "black-box" function. We do not have an analytical expression for it; its value can only be ascertained by executing the computationally

expensive process of training and evaluating the machine learning model. The high cost of evaluating $f(\lambda)$ is the primary motivation for avoiding an exhaustive search.

- **State-Space Tree:** The B&B-GS algorithm explores the search space by constructing a search tree. The **root node** of this tree represents the entire hyperparameter grid Λ. **Branching** occurs by partitioning a node's corresponding sub-grid into smaller sub-grids. For simplicity and generality, we employ a binary splitting strategy. At each branching step, a sub-grid is divided into two smaller sub-grids along one of its dimensions. This process continues recursively, creating a tree structure where each node represents a hyper-rectangle within the original grid. A **leaf node** in this context represents a region of the grid that is either pruned, fully evaluated, or contains only a single hyperparameter configuration.

C. Bounding Function

The efficacy of any Branch and Bound algorithm is contingent upon its ability to compute tight bounds on the objective function for subproblems. In our context, this means we need a method to calculate a reliable **upper bound** on the performance score $f(\lambda)$ for all configurations λ within a given sub-grid $N \subseteq \Lambda$, without having to evaluate every point in N. Our approach achieves this by making a mild and often practical assumption about the behavior of the performance landscape: Lipschitz continuity. This assumption forms the theoretical cornerstone of our pruning strategy.

1) The Lipchitz Assumption

A function $f: \Lambda \rightarrow R$ is said to be Lipschitz continuous with respect to a distance metric d if there exists a non-negative constant L, known as the Lipschitz constant, such that for all $\lambda a, \lambda b \in \Lambda$:

$$|f(\lambda_a) - f(\lambda_b)| \leq L \cdot d(\lambda_a, \lambda_b) \quad (1)$$

This condition implies that the rate of change of the function is bounded. In the context of HPO, it suggests that small changes in hyperparameter values will not lead to arbitrarily large jumps in model performance. This is a reasonable assumption for many well-behaved machine learning models and performance metrics. From the Lipschitz condition, we can derive a direct upper bound. If we have evaluated the function at a point λ_c within a sub-grid N, then for any other point $\lambda \in N$, its value is bounded by:

$$f(\lambda) \leq f(\lambda_c) + L \cdot d(\lambda, \lambda_c) \quad (2)$$

Therefore, an upper bound for the performance over the entire sub-grid N can be established by finding the maximum possible value of this expression:

$$UB(N) = f(\lambda_c) + L \cdot \lambda \in N max d(\lambda, \lambda_c) \quad (3)$$

The challenge thus shifts from the intractable task of evaluating f at all points in N to the more manageable tasks of defining an appropriate distance metric d and estimating the Lipschitz constant L.

2) Hyperparameter Metric Space

A naive Euclidean distance is unsuitable for hyperparameter spaces because different hyperparameters often have vastly different scales and distributions. For example, a regularization parameter C for an SVM is typically varied on a logarithmic scale (e.g., 0.01, 0.1, 1, 10), while a parameter like $n\_estimators$ in a random forest is varied on a linear scale (e.g., 100, 200, 300). A change of 1.0 unit has a completely different meaning for each.

To address this, we define a normalized, log-aware distance metric. Let a hyperparameter configuration be a vector $\lambda = (p_1, p_2, \ldots, p_k)$. The distance calculation involves a transformation function $T(\lambda)$ that preprocesses the vector before applying a standard L2 norm.

For each parameter p_i that is typically tuned on a logarithmic scale (e.g., SVM's C and gamma), we apply a log transformation: $p_i' = log(p_i)$. For parameters tuned on a linear scale, we set $p_i' = p_i$. This aligns the perceptual distance of the parameters with their impact on the model.

After transformation, each parameter dimension is normalized to the range based on the minimum and maximum values in the original grid for that dimension. Let $p_{i,min}'$ and $p_{i,max}'$ be the minimum and maximum transformed values for the i-th parameter in the entire grid Λ. The normalized value p_i'' is:

$$p_i'' = \frac{p_i' - p_{i,min}'}{p_{i,max}' - p_{i,min}'} \quad (4)$$

The distance $d(\lambda_a, \lambda_b)$ between two configurations is the Euclidean (L2) distance between their transformed and normalized vectors, $T(\lambda_a)$ and $T(\lambda_b)$:

$$d(\lambda_a, \lambda_b) = ||T(\lambda_a) - T(\lambda_b)||_2 \quad (5)$$

This tailored metric ensures that all hyperparameters contribute to the distance calculation on a comparable scale.

3) Adaptive Estimation of the Lipschitz Constant $\hat{L}$

In a true black-box optimization scenario, the global Lipschitz constant L is unknown a priori. A fixed, overly conservative (large) estimate for L would result in loose upper bounds and ineffective pruning, while an overly optimistic (small) estimate could lead to the erroneous pruning of sub-grids containing the optimum.

To circumvent this, we propose an adaptive, online estimation strategy. The algorithm maintains a history, $H = \{(\lambda_i, f(\lambda_i))\}$, of all configurations that have been evaluated so far. After each new evaluation, the estimate of the Lipschitz constant, $\hat{L}$, is updated based on the maximum slope observed between any two points evaluated to date.

$$\hat{L} = \underset{(\lambda_i, f_i),(\lambda_j, f_j) \in H, i \neq j}{\mathrm{r} \cdot \max} \left(\frac{|f_i - f_j|}{d(\lambda_i, \lambda_j)} \right) \quad (6)$$

D. Branch & Bound Grid Search Algorithm

1) **Initialization**: Evaluate n_init random points from the grid to obtain an initial best score, f^*, and an initial estimate for the Lipschitz constant, $\hat{L}$. Create a root node, N_{root}, representing the entire parameter grid, calculate its Upper Bound, and add it to a priority queue, Q.

2) **Termination Condition Check**: If the queue Q is empty (no more nodes to explore), the search process terminates. The best configuration (λ_{best}) that yielded the score f^* is the solution.

3) **Node Selection**: If Q is not empty, select and dequeue the node N from the queue that has the highest Upper Bound (UB) value.

4) **Pruning Step**: Check if the UB of node N is less than or equal to the current best score found, f^*. If true ($UB(N) \leq f^*$), this node and its entire branch cannot contain a better solution. Discard (prune) the node and return to Step 2.

5) **Branching Step**: If node N is not pruned, generate its children (branching). This process involves:

a) Selecting a pivot point within the sub-grid N.

b) Evaluating the pivot point to potentially update f^* and the Lipschitz estimate $\hat{L}$.

c) Splitting the sub-grid N into two child nodes along its longest dimension.

6) **Bounding and Enqueueing**: For each newly created child node N_{child}:

a) Calculate its Upper Bound using the bounding function.

b) If the child's UB is greater than f^*, add it to the priority queue Q.

7) **Iteration**: Return to Step 2 to continue the search process.

IV. Experimental Testing

To validate the efficacy and limitations of the proposed B&B-GS algorithm, a series of experiments were conducted. The goal was to compare its efficiency and solution quality against standard hyperparameter tuning methods on a well-understood classification problem.

A. Dataset

All experiments were performed on the **Breast Cancer Wisconsin (Diagnostic) dataset**. This dataset, containing 569 instances and 30 numeric features, requires a binary classification of tumors as malignant or benign. The data was preprocessed by encoding the target variable to numeric values and standardizing the features using StandardScaler. The dataset was split into a 70% training set for the hyperparameter optimization (HPO) process and a 30% held-out test set for final model evaluation.

B. Models and Hyperparameter Grids

Three different classification models from scikit-learn were used to test the general applicability of the search methods:

1. **Random Forest Classifier:** A large grid of 108 combinations was defined for n_estimators, max_depth, min_samples_split, and min_samples_leaf.
2. **Support Vector Machine (SVC):** A grid of 32 combinations was used, including the categorical parameter kernel ('rbf', 'linear') alongside numerical C and gamma parameters.
3. **Logistic Regression:** A grid of 24 combinations was defined, including the categorical parameters penalty ('l1', 'l2') and solver ('liblinear', 'saga'), along with the numerical parameter C.

C. Comparison Methods

The performance of our proposed **B&B-GS** was benchmarked against:

1. **GridSearchCV:** The exhaustive search baseline, guaranteed to find the true optimal parameters on the grid.
2. **RandomizedSearchCV:** The efficiency baseline, configured to sample a fixed number of parameter combinations.

D. Evaluation Metrics

For all experiments, model performance was evaluated using 3-fold cross-validation on the training set. The algorithms were compared based on:

1. **Best CV Score:** The highest mean accuracy achieved during the search.
2. **Number of Evaluations:** The total number of parameter sets evaluated, a direct proxy for computational cost.
3. **Execution Time:** The wall-clock time required to complete the search.
4. **Final Test Accuracy:** The accuracy of the best-found model on the held-out 30% test set, to assess generalization performance.

V. Results and Analysis

The comparative experiments yielded clear results, validating the B&B-GS algorithm's ability to balance efficiency with optimality. The complete findings, extracted directly from the final experimental notebook, are presented in Table 1.

TABLE I. Comparative Performance of HPO Methods on the Breast Cancer Dataset

Model	Method	Total Grid	Evaluation Metrics			
			Evals	*Best CV Score*	*Time (s)*	*Test Accuracy (%)*
Random Forest	**B&B-GS**	**108**	42	0.9421	17.79	96.49
	GridSearchCV		108	0.9446	58.36	97.08
	RandomizedSearchCV		30	0.9446	11.69	96.49
SVM	**B&B-GS**	**32**	32	0.9648	0.37	98.25
	GridSearchCV		32	0.9648	0.25	98.25
	RandomizedSearchCV		20	0.9648	0.25	98.25
Logistic Regression	**B&B-GS**	**24**	24	0.9698	1.50	97.08
	GridSearchCV		24	0.9698	1.61	97.08
	RandomizedSearchCV		20	0.9698	1.30	97.08

Across all three models, B&B-GS successfully found a cross-validation score that was either optimal or extremely close to the optimum identified by GridSearchCV. For Random Forest, it found a score of 0.9421 compared to the grid's best of 0.9446, a negligible difference. For SVM and Logistic Regression, it located the exact same optimal score. This demonstrates that the algorithm is highly effective at finding the best-performing region of the grid. When evaluated on the held-out test set, the models tuned by B&B-GS showed excellent generalization, achieving test accuracies on par with the other methods.

The results reveal a critical insight into the behavior of B&B-GS: its efficiency is highly dependent on the "smoothness" of the hyperparameter performance landscape.

- **Success Case (Random Forest):** For the Random Forest model, whose performance often changes more gradually with numerical parameter adjustments, B&B-GS was highly effective. It required only **42 evaluations** to find a near-optimal solution, representing a **61% reduction** in computational effort compared to GridSearchCV's 108 evaluations. This confirms that on a suitable landscape, the bounding mechanism can successfully prune large, unpromising regions of the grid.
- **Limitation Case (SVM & Logistic Regression):** In contrast, for SVM and Logistic Regression, B&B-GS evaluated the entire grid (32/32 and 24/24, respectively), offering no efficiency gain over GridSearchCV. This behavior is a direct consequence of the algorithm's reliance on the Lipschitz assumption. The grids for these models included categorical parameters (kernel, penalty). A change between two categorical values (e.g., kernel='rbf' to kernel='linear') can cause a large, abrupt jump in performance. This creates a very "steep" or rugged performance landscape. The algorithm detects these steep changes and calculates a very large Lipschitz constant (L) to be safe. With a large L, the calculated Upper Bound for every unexplored node becomes too optimistic, preventing the pruning condition (UpperBound <= best_score) from ever being met. Consequently, the algorithm is forced to explore every node.

VI. Conclusion and Evaluation

This paper introduced and empirically evaluated a Branch and Bound Grid Search (B&B-GS) algorithm, confirming it provides a powerful but nuanced trade-off between exhaustive and stochastic search methods. The core strength of B&B-GS lies in its ability to find the guaranteed optimal parameters on a grid, a feat demonstrated in its perfect replication of GridSearchCV's optimal scores. This determinism is paired with high efficiency on hyperparameter landscapes that are relatively smooth, such as that of the Random Forest, where substantial reductions in model evaluations translated to significant savings in execution time.

However, this advantage is fundamentally constrained by the algorithm's reliance on the Lipschitz assumption. The experiments clearly show that on rugged landscapes, often created by categorical parameters or unstable model configurations, the pruning mechanism becomes ineffective. This limitation not only diminishes the algorithm's performance to that of an exhaustive search in terms of evaluations but can also make it slightly slower in wall-clock time due to its inherent algorithmic overhead. In the race for raw speed, RandomizedSearchCV remains superior, though at the cost of determinism.

Therefore, Branch-and-Bound Grid Search should not be seen as a universal replacement for other HPO methods, but rather as a highly valuable tool for a specific use case: when a practitioner requires the guaranteed optimality of a grid search for a model whose performance landscape is expected to be well-behaved. Future research should focus on developing more robust bounding functions less sensitive to categorical variables and incorporating time-based budgeting to improve its practical utility in a wider range of applications.

Acknowledgment

The author would like to express sincere gratitude to God Almighty for blessings and grace in writing this paper. The author would also like to thank Dr. Ir. Rinaldi Munir, M.T., for being a very supportive lecturer of IF2211 Strategi Algoritma

References

[1] E. A. Usova, V. O. Koldanov, P. A. Koldanov, and Y. D. Sergeyev, "Lipschitz global optimization and machine learning: helping each other to solve complex problems," *ITM Web of Conferences*, vol. 59, p. 01019, 2024. [Accessed: Jun. 24, 2025].

[2] I. Jair, "What Is Hyperparameter Optimization?," *Medium*, Jan. 13, 2021. [Online]. Available: https://medium.com/@jairiidriss/what-is-hyperparameter-optimization-477c15fe8cbe. [Accessed: Jun. 24, 2025].

[3] J. Bergstra and Y. Bengio, "Random Search for Hyper-Parameter Optimization," *Journal of Machine Learning Research*, vol. 13, pp. 281–305, Feb. 2012. [Accessed: Jun. 24, 2025].

[4] J. Snoek, H. Larochelle, and R. P. Adams, "Practical Bayesian Optimization of Machine Learning Algorithms," in *Advances in Neural Information Processing Systems 25 (NIPS 2012)*, 2012, pp. 2951–2959. [Accessed: Jun. 24, 2025].

[5] R. Munir, Algoritma Branch & Bound Bagian 1. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/17-Algoritma-Branch-and-Bound-(2025)-Bagian1.pdf. [Accessed 24 June 2025].

[6] Scikit-learn Developers, "3.2. Tuning the hyper-parameters of an estimator," *scikit-learn 1.5.1 documentation*. [Online]. Available: https://scikit-learn.org/stable/modules/grid_search.html. [Accessed: Jun. 24, 2025].

[7] W. H. Wolberg, O. L. Mangasarian, and N. Street, "Breast Cancer Wisconsin (Diagnostic) Data Set," *UCI Machine Learning Repository*, 1995. [Online]. Available: https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic. [Accessed 24 June 2025].

STATEMENT

Hereby, I declare that this paper I have written is my own work, not a reproduction or translation of someone else's paper, and not plagiarized.

Bandung, 24 Juni 2025

Michael Alexander Angkawijaya
13523102