Leveraging machine learning for column generation in the dial-a-ride problem with driver preferences

Sana Ouasaid, Mohammed Saddoune
Machine Intelligence Laboratory, Department of Computer Science, Faculty of Sciences and Technologies, University of Hassan II Casablanca, Casablanca, Morocco

Article Info

Article history:

Received Jun 18, 2024
Revised Mar 24, 2025
Accepted Jun 8, 2025

Keywords:

Binary classification
Clustering-based initialization
Column generation algorithm
Dial-a-ride problem
Pricing subproblem

ABSTRACT

The dial-a-ride problem (DARP) is a significant challenge in door-to-door transportation, requiring the development of feasible schedules for transportation requests while respecting various constraints. This paper addresses a variant of DARP with time windows and drivers' preferences (DARPDP). We introduce a solution methodology integrating machine learning (ML) into a column generation (CG) algorithm framework. The problem is reformulated into a master problem and a pricing subproblem. Initially, a clustering-based approach generates the initial columns, followed by a customized ML-based heuristic to solve each pricing subproblem. Experimental results demonstrate the efficiency of our approach: it reduces the number of the new generated columns by up to 25%, accelerating the convergence of the CG algorithm. Furthermore, it achieves a solution cost gap of only 1.08% compared to the best-known solution for large instances, while significantly reducing computation time.

Corresponding Author:

Sana Ouasaid
Machine Intelligence Laboratory, Department of Computer Science, Faculty of Sciences and Technologies
University of Hassan II Casablanca
Casablanca, Morocco
Email: sana.ouasaid1-etu@etu.univh2c.ma

1. INTRODUCTION

The dial-a-ride problem (DARP) is a well-known and thoroughly explored area of study that focuses on designing efficient routing schedules for a fleet of vehicles to fulfill transportation requests, each involving a pickup and delivery point [1]. Optimizing DARP requires balancing cost-effectiveness with high service quality. It considers factors such as customer ride times and deviations from desired departure or arrival times. This challenge becomes even more complex when incorporating driver preferences, as in the DARP with time windows and drivers' preferences (DARPDP) [2], [3]. In DARPDP, drivers aim to maximize served requests while also considering preferred destinations and arrival times, adding another layer of complexity to the optimization process. Existing approaches for DARPDP, such as those based on iterated local search metaheuristics, have shown promise but struggle with large-scale instances [3]. This scalability issue motivates the need for more efficient solution methodologies, particularly those well-suited to large-scale problems. One such method is column generation (CG), an iterative optimization technique that starts with a restricted set of columns representing a feasible solution. The master problem is solved using this initial subset, producing a current solution and dual variable values (shadow prices). These dual variables guide the pricing subproblem, which identifies new columns with negative reduced costs. If such columns are found, they are added to the master problem, and the process is repeated until no further improvement is possible.

CG has proven effective for addressing DARP and similar transportation challenges. Garaix *et al.* [4] optimized passenger occupancy in low-demand scenarios through efficient continuous relaxation of a set-partitioning model. Hybrid approaches further enhance CG, such as Parragh and Schmid [5], who combined it with variable and large neighborhood search to improve solutions. Rahmani *et al.* [6] used dynamic programming within a branch-and-price framework to handle ride-time constraints in the pricing problem. Beyond DARP, CG has been applied to other transportation contexts, including bus scheduling [7], ride-sharing [8], and selective DARP with Benders decomposition [9]. While CG is effective, integrating machine learning (ML) into DARP solutions offers new possibilities. For example, Markov decision processes were used to optimize dynamic DARP, improving customer insertion evaluation [10]. Random forests (RF) were utilized for operator selection in dynamic E-ADARP with a two-phase metaheuristic [11]. Thompson sampling effectively modeled dual values of requests, yielding faster solutions for large DARP instances [12]. An adaptive logit model significantly sped up traditional parallel insertion heuristics for generating feasible solutions [13].

Building on these efforts, integrating ML with CG enhances efficiency by guiding key decisions like pricing and column selection [14]. For example, deep learning has been used to predict flight connection probabilities [15] and promising aircrew pairings [16], while support vector machines (SVM) efficiently generated high-quality columns for graph coloring [17]. This integration of ML into CG opens promising avenues for improving large-scale DARP solutions. Recent work highlights its potential, particularly for intelligent column selection. Morabit *et al.* [18] used a graph neural network for column selection in vehicle and crew scheduling and vehicle routing, reducing computation time by up to 30%. They later extended this to arc selection, further refining the process [19]. Chi *et al.* [20] proposed a Q-learning-based single-column selection policy with a graph neural network, outperforming greedy heuristics. Gerbaux *et al.* [21] developed a CG heuristic for multi-depot electric vehicle scheduling using graph neural networks and transfer learning for larger instances.

Addressing the DARP using ML methods is a relatively recent area of research and remains underexplored. While significant efforts have focused on integrating ML into CG algorithms, particularly for problems like crew scheduling and vehicle routing, the application of such techniques to the DARP has been limited. Existing studies in this domain primarily enhance the pricing subproblem by predicting promising variables, such as routes or crew assignments, to improve optimization. However, for the DARPDP—a variant introduced in prior work and initially tackled using a metaheuristic—the main challenge extends beyond the pricing subproblem to include identifying the most relevant requests to be served. This aspect is critical and requires targeted strategies. Our approach builds upon this foundation by integrating ML into the request selection process. This guides the CG algorithm, offering a novel perspective for addressing the DARPDP.

This paper addresses the scalability gap in existing DARPDP solutions by introducing a novel approach that combines ML and CG. This integration offers a significant advancement over existing methods by leveraging ML's predictive capabilities to enhance the efficiency of CG. Our approach reformulates the DARPDP into a master problem and a pricing subproblem within the CG framework. The master problem selects optimal routes (columns), while the pricing subproblem generates promising new routes. We enhance the CG process with two key innovations: a clustering-based strategy for initial CG to effectively structure the search space, and a customized ML-based heuristic for solving the pricing subproblem. This ML heuristic learns from past instances and adapts to the evolving solution, guiding the generation of high-quality columns and accelerating convergence. This targeted approach allows us to tackle large-scale DARPDP instances more efficiently than existing methods, providing valuable insights into the interplay between ML and optimization algorithms for complex routing problems.

The remainder of this paper is organized as follows: section 2 details our proposed methodology, including the integration of ML into the request selection process for CG. In section 3, we present the experimental results and evaluate the performance of our approach. Finally, section 4 concludes the paper and discusses potential future research directions.

2. METHOD

Although traditional CG works well for solving complex problems, early attempts to apply it revealed limitations in exploring the entire range of potential solutions. Frequently, the algorithm became stuck in local minima, preventing the sub-problem from generating new columns. Our proposed framework improves the traditional CG process by integrating a learning mechanism. This mechanism improves both the quality of generated columns and the algorithm's convergence speed.

As outlined in Algorithm 1, the proposed ML-CG algorithm starts by training a binary classification model on optimally solved problem instances (line 1). To solve a new instance, it first extracts problem-specific features (line 2). Then, it generates an initial set of columns using a two-phase heuristic: clustering and the ”randomized nearest heuristic” (RNI) (line 3). These columns are used as the initial variables for solving the restricted master problem (RMP). The main part of the algorithm is an iterative loop (lines 4 to 11) that continues until the stopping condition is met. In each iteration, the RMP is solved using the columns in the pool up to that point (line 5). This solution provides the values for the dual variables of the constraints. Then, statistical features are computed based on the solutions found up to the current iteration (line 7), and these features are combined with the original problem-specific features and the dual information to create a comprehensive set of features (line 8). The output of the model guides a diversification heuristic to generate new columns with negative reduced costs, thus further reducing the objective function (line 9). These new columns are added to the pool of variables for the RMP. Once the algorithm finishes iterating, it returns the final solution by solving the RMP on the set of all columns generated during the execution of the algorithm (line 12).

Algorithm 1 Column generation algorithm (ML-CG)

```
Require: $\mathcal{P}$: Problem Instance, $\mathcal{T}$: Maximum Time Limit
Ensure: s: Solution
 1: $\mathcal{M} \leftarrow$ TrainBinaryClassificationModel()
 2: $\mathbf{f_1} \leftarrow$ ExtractProbFeatures($P$)
 3: $\mathcal{C} \leftarrow$ ClusteringBasedRNH()
 4: while $cpu \leq \mathcal{T}$ do
 5:     $\mathbf{s} \leftarrow$ SolveMP($\mathcal{C}$)
 6:     $\mathbf{D} \leftarrow$ SolveRLP($\mathbf{s}$)
 7:     $\mathbf{f_2} \leftarrow$ ExtractStatsFeatures($\mathbf{s}$)
 8:     $\mathbf{features} \leftarrow$ Combine($\mathbf{f_1}, \mathbf{f_2}, \mathbf{D}$)
 9:     $\mathcal{N} \leftarrow$ GeneratNewColumns($\mathcal{P}, \mathbf{features}, \mathcal{M}$)
10:     $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{N}$
11: end while
12: return $\mathbf{s}$
```

The ML-CG algorithm is built upon three fundamental components that work together to iteratively improve the solution. First, it begins with the construction of an initial pool of columns that serve as a starting point for the optimization. Then, through the formulation of the RMP and its subproblems, the algorithm repeatedly generates new columns with negative reduced costs to enhance the RMP and guide the solution process.

2.1. Generation of the initial set of columns

To generate initial columns, we propose a two-stage heuristic. In the first stage, requests are separated into clusters, and each cluster is assigned to a vehicle. We use three clustering approaches to diversify the initial columns: K-means, K-medoids, and random clustering. Then, for each cluster, a route is constructed using the randomized nearest insertion heuristic (RNI) [22]. The method is further detailed in Algorithm 2.

Algorithm 2 outlines the process of generating initial columns based on a given problem instance and clustering methods. For each method, requests are clustered, and for each cluster, a route is built using the RNI heuristic. Starting with a route from the vehicle's origin to its destination, at each iteration, RNI ranks the requests in increasing order of d, which is defined as the sum of four distances: the distances between their respective pickup points, the delivery points, and the cross-distances between the delivery point of the first request and the pickup point of the second request, and vice versa. RNI inserts the ℓth element in the ordered set (ℓ represents the randomized factor in the heuristic) into the route, and the remaining requests are sequentially inserted at the best feasible position, which minimizes the increase in the route's cost. The final route is converted into a column and added to the ColumnsPool.

Algorithm 2 Clustering based RNH

```
Require: $\mathcal{P}$: Problem instance, $\mathcal{CM}$: Clustering methods
Ensure: $\mathcal{I}$: Initial columns pool
 1: $ColumnsPool \leftarrow \emptyset$
 2: for $\mathcal{M}$ in $\mathcal{CM}$ do
 3:     Cluster requests ($\mathcal{P}$.requests) into $K$ groups, each associated with a vehicle in $\mathcal{P}$.vehicles
 4:     for $k$ in $\{1, \ldots, K\}$ do
 5:         $requests \leftarrow C_k.requests$
 6:         Rank the requests in $C_k$ in increasing order of $d$
 7:         Select the $\ell$th request as the seed request ($\ell$ is a randomized_factor()) and insert it into $route_k$
 8:         Delete the seed request from $requests$
 9:         for $request$ in $requests$ do
10:             Insert request into $route_k$ using the least cost insertion method
11:         end for
12:         $ColumnsPool \leftarrow ColumnsPool \cup$ {Convert $route_k$ to column}
13:     end for
14: end for
```

2.2. Restricted master problem and pricing subproblem

To implement CG, the original mixed-integer programming (MIP) model outlined in [3] needs to be reformulated as a RMP. This RMP considers a subset of columns for the solution. Subsequently, one or more subproblems are solved using the current dual solution of the RMP to identify new, advantageous columns to add to the RMP to enhance the objective value.

Let Ω be the set of all possible routes. A feasible route is a Hamiltonian path that starts at the driver's origin node, ends at the driver's destination node, and visits only a subset of nodes corresponding to the requests. Define Ω_k as the subset of Ω containing routes assignable to a specific vehicle $k \in K$. Use the variable y_{r_k} to indicate the selection of a potential route r_k, setting y_{r_k} to 1 when selecting r_k and to 0 otherwise. Represent the cost of a generated route r_k as c_{r_k}. Additionally, set the constant a_{i,r_k} to 1 if route r_k serves request i. Using the LP relaxation of the Dantzig-Wolfe reformulation, we can describe the DARPDP with the following model.

$$\text{Min} \sum_{k\in K} \sum_{r_k \in \Omega_k} c_{r_k} y_{rk} \tag{1}$$

Subject to the constraints:

$$\sum_{k\in K} \sum_{r_k \in \Omega_k} a_{i,r_k} y_{rk} \leq 1 \quad \forall i \in P \quad (\pi_i \leq 0) \tag{2}$$

$$\sum_{r_k \in \Omega_k} y_{rk} \leq 1 \quad \forall k \in K \quad (\mu_k \leq 0) \tag{3}$$

$$\sum_{i\in P} \sum_{k\in K} \sum_{r_k \in \Omega_k} a_{i,r_k} y_{rk} \geq \varepsilon \quad (\gamma \geq 0) \tag{4}$$

In this (master problem), the objective minimizes the total routing cost of the vehicle routes used; the first constraint ensures that each request is covered by at most one vehicle route, and the second constraint ensures that at most one route is selected per vehicle. Since the rejection of a request is allowed, the third constraint restricts the number of rejected requests. The dual variables associated with each constraint are specified between parentheses next to the constraint in the model. Let π_i be the nonnegative dual variable associated with the visit of request r (constraints 2), and let μ_k be the nonpositive dual variable associated with the fleet size constraint (constraint 3), and let γ the one restricts the number of rejected requests. We derive the RMP by replacing Ω_k in the master problem with a subset $\overline{\Omega_k}$ for which the problem is primal feasible.

The subproblem, corresponding to a column in the RMP, determines the route, schedule, and passenger assignments to a single vehicle. It retains the same variables as the original problem formulation [3], excluding the k index, as it now pertains to a specific vehicle. The objective function minimizes the reduced cost of the route, considering the route cost and dual prices from the RMP:

$$\min \left\{ c_{r_k} - \sum_{i \in P} a_{i,r_k} \pi_i - \mu_k - \gamma \sum_{i \in P} a_{i,r_k} \leq 0 : r_k \in \Omega_k \right\} \tag{5}$$

The cost of a route, c_{r_k}, aligns with the objective function of the original formulation but is specific to the individual route k. The subproblem inherits the constraints from the original formulation. These ensure that the generated route meets all feasibility requirements, including vehicle capacity, time windows, and passenger service guarantees.

2.3. Generate new columns

In the problem addressed, rejecting a small portion of the requests is allowed if it benefits the overall objective value. This means that the optimal solution does not necessarily serve all requests, which makes it challenging to identify the most profitable combination of requests to serve. It is clear that diversifying the search in the CG algorithm is necessary, but this should be done intelligently to avoid slowing down the process. To address this, we train a model on problem-specific features, historical solutions, dual values, and statistical measures to predict request profitability. Using the predicted probabilities, we rank requests from most to least profitable, which will guide a diversification heuristic called the learning-based diversification heuristic (LBDH) to solve each subproblem. In the following subsections, we first discuss the key aspects of training the binary classification model, including reformulating the request selection problem as a binary classification task, selecting and extracting relevant features, choosing suitable classification algorithms, optimizing hyperparameters, and evaluating the model's performance using specific metrics. Next, we describe the diversification process, where the ranked list of requests intelligently guides the diversification heuristic to explore the most promising combinations of requests.

2.3.1. Training a classification model for request profitability

This section focuses on training a binary classification model to predict the profitability of requests, a crucial step within our CG algorithm. We approach this as a standard classification problem. Each training example is represented as a tuple $(f_r \in \mathbb{R}^n, c_r \in 0, 1)$. Here, f_r denotes the feature vector, encompassing n distinct characteristics of the request. The variable c_r acts as the binary class label, taking on a value of 1 if the request is part of the optimal solution and 0 otherwise. By learning from these examples, the model aims to accurately classify new requests, guiding the CG process effectively.

a. Collecting data

Initially, we aimed to construct a training set using small problem instances solved to optimality and then apply this knowledge to tackle larger ones. However, the number of examples proved insufficient, and the efficiency of ML algorithms relies on a large dataset. To address this, we extended the dataset by generating additional small instances in the same manner as in [3] and solving them optimally using a commercial solver. Ultimately, we constructed 1000 examples in total.

Given the significant imbalance in our dataset, where the majority class ('1') covers almost 80% of the sample, undersampling techniques are applied to address this issue. These techniques involve removing instances from the training dataset that are part of the majority class to improve the balance between classes. This adjustment ensures that majority and minority classes are on a similar scale, enhancing the learning algorithm's sensitivity to the minority class.

In this study, we addressed class imbalance in each classification model by combining KNNOrder and synthetic minority over-sampling technique (SMOTE). KNNOrder, a KNN-based undersampling technique introduced by [23], reduces the imbalance by selectively removing examples from the majority class. We then apply SMOTE, a widely used oversampling method that generates synthetic examples for the minority class [24], to further balance the dataset. We demonstrated the effectiveness of KNNOrder + SMOTE by compared it with various other methods deal with unbalanced data: no sampling (serving as a baseline), random undersampling (randomly removing majority class examples), random oversampling (randomly duplicating minority class examples), SMOTE (generating synthetic examples for the minority class through interpolation), and finally KNNOrder (targeted removal of majority class examples near the decision boundary). The Table 1 presents the results obtained in terms of accuracy, F1-score, precision, and recall for each method.

Table 1. Comparison of sampling methods

Method	Accuracy	F1-score	Precision	Recall
No Sampling	0.80	0.87	0.91	0.83
Random Undersampling	0.85	0.90	0.88	0.92
Random Oversampling	0.79	0.86	0.91	0.82
KNNOrder	0.83	0.89	0.88	0.91
SMOTE	0.82	0.88	0.90	0.87
KNNOrder + SMOTE	0.86	0.91	0.88	0.94

The results demonstrate that KNNOrder + SMOTE combination achieves strong performance, with an F1-score of 0.91 and a recall of 0.94, surpassing the other tested techniques in terms of these metrics. A high recall suggests that this approach will be more effective at identifying instances from minority class. However, it is important to note that these results are specific to the dataset used in this study. Further investigation with other datasets is necessary to confirm the robustness and generalizability of the KNNOrder + SMOTE method.

b. Feature extraction

i) Problem-specific features: the features extracted describe each request $r \in R$ in the context of the DARPDP problem, namely:

– Request degree: indicates the number of requests that can be served simultaneously with this request without violating any constraints.

– Request ride time: represents the duration between a request's pickup and drop-off.

– Correlation between request pickup and other pickups: represents the average Pearson correlation coefficient between the pickup node of a given request (defined by its x-location, y-location, earliest time, and latest time) and the pickup nodes of all other requests within the problem instance.

– Correlation between request delivery and other deliveries: similar to pickup correlation, this metric assesses the relationship between the delivery node of a given request and the delivery nodes of all other requests within the problem instance.

– Dual information: indicates how much the overall solution cost would change in response to a marginal relaxation of the coverage constraint for a specific request.

– Request distance to vehicles: represents the average distance of each request from the set of all vehicles.

ii) Statistical-based features: in addition to the problem-specific features, two statistical measures are introduced, inspired by [25]. These measures are calculated from a set of up to N feasible solutions, selected from those obtained in previous iterations. x_r^i is a binary variable indicating whether request r is included in solution i ($x_r^i = 1$) or not ($x_r^i = 0$).

– Correlation-based measure: this measure evaluates the linear relationship between the presence of a request and the quality of feasible solutions. The Pearson correlation coefficient for request r is calculated as follows:

$$Corr(r) = \frac{\sum_{i \in N}(x_r^i - \bar{x}_r)(\text{obj}^i - \bar{\text{obj}})}{\sqrt{\sum_{i \in N}(x_r^i - \bar{x}_r)^2}\sqrt{\sum_{i \in N}(\text{obj}^i - \bar{\text{obj}})^2}} \quad (6)$$

Where obj^i is the objective function value associated with solution i, $\bar{x}_r$ is the average of x_r^i across all solutions, and $\bar{\text{obj}}$ represents the average of the objective function values. This measure identifies demands that contribute to higher-quality solutions. For instance, a correlation close to 1 indicates that the presence of the request is often associated with high-quality solutions, while a correlation close to -1 indicates the opposite.

– Ranking-based measure: this measure is based on the position of feasible solutions in a ranking determined by their quality (objective function value). For each solution i, its rank is denoted as rank_i, where a lower rank corresponds to a higher-quality solution. The score assigned to request r is given by:

$$RankScore(r) = \sum_{i \in N} \frac{x_r^i}{\text{rank}_i} \quad (7)$$

This measure favors requests that appear in higher-ranked solutions by assigning a higher score to those present in high-quality solutions. Requests with higher scores are thus highlighted for their potential contribution to optimal solutions. These statistical data are combined with the problem-specific features of DARPDP. This results in a detailed representation of the requests, enabling ML models to more effectively identify the best solutions to the problem.

c. Selection of the best classification model

We trained four state-of-the-art supervised ML algorithms to choose the appropriate classification algorithm. The selected classification algorithms include the Gaussian naïve Bayes (NB) classifier, an extension of the NB algorithm that relies on posterior probability estimation and assumes that the features used to characterize instances are conditionally independent. The SVM determines the best hyperplane-based decision boundaries that segregate n-dimensional space into classes, allowing for the easy categorization of new data points. RF is an ensemble-based method. Several decision trees are employed with a random sample of characteristics to improve predictive accuracy and manage complex relationships within the data. Finally, a multi-layer perceptron (MLP) is a feed-forward artificial neural network. It consists of interconnected nodes arranged into multiple layers (input, hidden, and output). For a comprehensive survey of supervised classification techniques, see [26].

Model optimization is finding the hyperparameters that minimize or maximize a scoring function for a specific task. Each model has its hyperparameters with a set of possible values. This research employs the grid search technique to uncover the optimum values of the hyperparameters. Grid search accepts the hyperparameter names (e.g., the learning rate in MLP or the kernel in SVM) and a vector of possible values for each. Then, the function goes through all the combinations and returns a fine-tuned model using the best combination of values. Even though grid search can require more resources and time than other optimization methods, it works better with our problem since the datasets are not enormous. Table 2 shows both the hyperparameter configuration of each algorithm used by grid search and the best value of the parameters.

Table 2. Hyperparameters configurations

Algorithm	Hyperparameter values	Selected hyperparameters
SVM	'C': [0.001, 0.01, 0.1, 1, 10, 100]	C= 10
	'kernel': ['linear', 'poly', 'rbf', 'sigmoid']	kernel='linear'
	'gamma': ['scale', 'auto', 0.001, 0.01, 0.1, 1, 10]	gamma='scale'
MLP	'activation': ['logistic', 'tanh', 'relu']	activation='tanh'
	'alpha': [0.0001, 0.001, 0.01, 0.1]	alpha=0.0001
	'learning_rate_init': [0.001, 0.01, 0.1, 0.5]	learning_rate_init=0.5
	'hidden_layer_sizes': [(15, 10, 5), (50, 25, 10), ((nbFeatures + nbClasses)/2,)]	hidden_layer_sizes= ((nbFeatures + nbClasses)/2,)
NB	'priors': [None, [0.3, 0.7], [0.4, 0.6], [0.5, 0.5]]	priors= None
	'var_smoothing': [1e-9, 1e-8, 1e-7, 1e-6, 1e-5]	var_smoothing= 1e-09
RF	'n_estimators': [10, 20, 30, 50]	n_estimators=50
	'max_depth': [20, 30, None]	max_depth=20
	'min_samples_split': [2, 5, 10]	min_samples_split=2

We evaluated the performance of the classification models over five runs for each of the SVM, RF, MLP, and NB algorithms. The metrics used for this evaluation are accuracy (Acc), precision (P), recall (R), F1-score (F1), and time (T) measured in seconds. The detailed results for each run are presented in Tables 3 and 4, allowing for an analysis of the variability in model performance.

Figure 1 illustrates through box plots, the variations in key performance metrics and computational time for each model over five runs. The MLP model stands out for its high performance and robustness, achieving a maximum accuracy of 98.05% and stable median values for precision (80.7%), recall (78.1%), and F1-score (79.4%). Despite its slightly longer computational time (median of 3.2 seconds), its low variability ensures consistent performance, making it a reliable option for applications requiring stability. The NB model offers a good balance, with a maximum accuracy of 94.71% and a short computational time (1.1 seconds), though its variability in recall and F1-score may reduce stability. The RF model demonstrates moderate performance (median accuracy of 88% and computational time of 2.3 seconds) but suffers from high variability across all metrics, limiting its reliability in certain applications. In comparison, although the SVM model is fast, the MLP model emerges as the most relevant choice due to its balance between accuracy and computational cost.

Table 3. Performances of SVM and RF algorithms for five runs

	SVM					RF				
Run	Acc (%)	P (%)	R (%)	F1 (%)	T (s)	Acc (%)	P (%)	R (%)	F1 (%)	T (s)
1	78.2	81.2	78.1	79.6	0.14	90.4	92.1	89.4	90.7	0.35
2	82.3	83.5	81.9	82.7	0.31	84.8	88.4	84.1	86.2	0.41
3	68.7	70.1	66.8	68.4	0.17	72.1	74.8	71.5	73.1	0.20
4	88.6	90.2	87.9	89.0	0.24	89.1	91.7	88.3	90.0	0.34
5	70.9	73.2	70.4	71.8	0.18	69.4	72.9	68.5	70.6	0.28

Table 4. Performances of MLP and NB algorithms for five runs

	MLP					NB				
Run	Acc (%)	P (%)	R (%)	F1 (%)	T (s)	Acc (%)	P (%)	R (%)	F1 (%)	T (s)
1	96.4	95.6	94.8	95.2	0.27	94.7	93.8	92.4	93.1	0.15
2	89.4	92.1	90.4	91.2	0.31	87.2	90.5	88.7	89.6	0.19
3	79.2	82.6	80.1	81.3	0.11	79.4	81.4	78.6	79.9	0.30
4	98.1	96.9	96.4	96.7	0.16	94.3	93.3	91.9	92.6	0.20
5	78.0	80.7	78.1	79.4	0.22	61.7	64.1	61.8	62.9	0.25

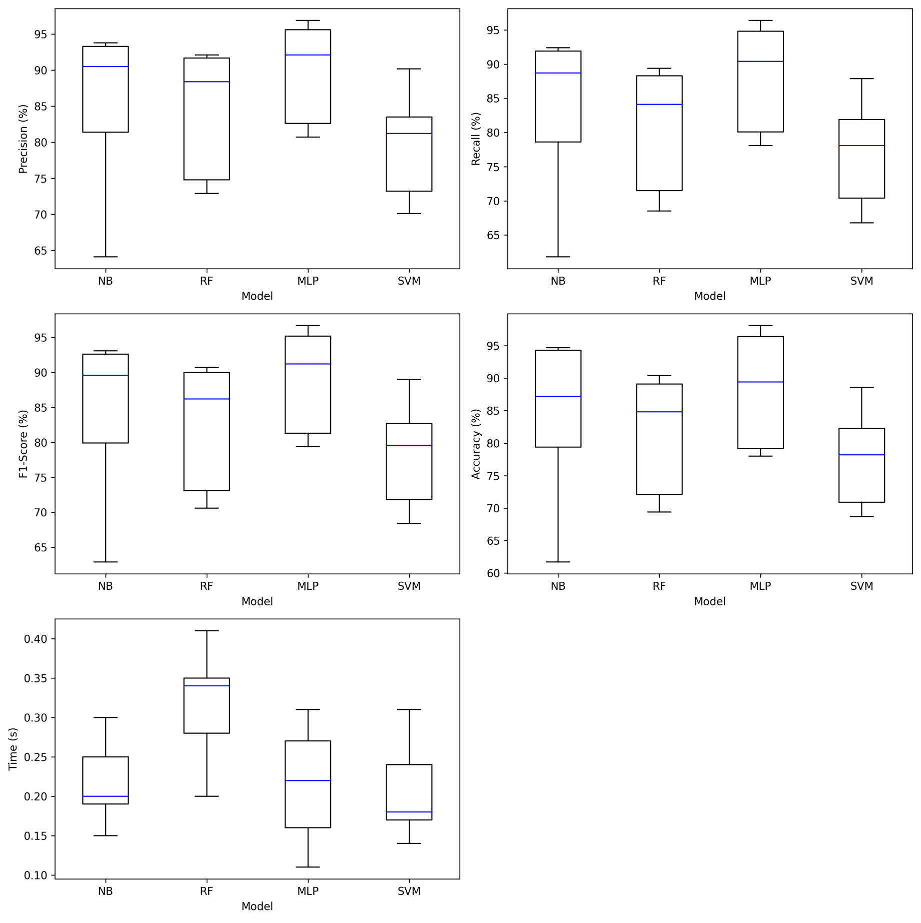

Figure 1. Comparative evaluation of four classification models: boxplot analysis

2.3.2. Learning-based diversification heuristic

ChatGPT said: At each iteration of the CG process, after solving the RMP and training the supervised classification model, we use the model to predict the profitability of each request. Based on these predictions, we construct three distinct subsets of requests from the current RMP solution. This is done to diversify the solutions explored for each subproblem:

– Reduced subset (R): created by removing a percentage p of the least profitable requests, i.e., those with the lowest predicted probabilities of being included in the optimal solution according to the classification model. This reduction focuses the exploration on potentially more relevant subsets of requests.
– Expanded subset (E): formed by adding percentage p of new requests deemed most profitable by classification model, selected from those not currently served by the route (column). This expansion enables the exploration of more complex solutions, integrating requests that could enhance the overall solution.
– Diversified subset (M): generated by swapping a percentage p of the initially served requests with an equal number of new requests. The removed requests are selected from the least profitable ones, while the added requests are chosen from the most profitable according to the model's predictions. This exchange introduces additional diversity by altering the composition of routes, allowing for the examination of potentially more effective alternative solutions.

After defining these subsets, we convert each into a graph. In these graphs, the nodes represent the requests and the vehicle, while the edges carry weights based on the reduced costs from the RMP. Using the CPLEX solver, we then solve these graphs to identify candidate columns with negative reduced costs.

3. RESULTS AND DISCUSSION

We designed a two-stage experimental setup to evaluate the effectiveness of the ML-CG algorithm. First, we assessed the benefits of integrating ML into CG by comparing ML-CG's performance against a standard CG approach. In the second stage, we compared the proposed ML-CG algorithm against the enhanced iterated local search (E-ILS) algorithm described in [3] and the CG algorithm from [5], referred to as constrained genetic programming (CGP). Both sets of tests utilized the same randomly generated data described in [3]. We developed the algorithms using Python 3.5 and executed them on a personal computer with an Intel Core i7-6700 processor operating at 2.6 GHz and 32 GB of RAM.

3.1. The effectiveness of incorporating machine learning techniques into CG

Initially, we discuss the number of columns handled by CG, examining scenarios with and without learning to solve the pricing problem. Table 5 displays the average number of columns generated by the CG algorithm, comparing results with and without the learning-based enhancement, across different numbers of requests. One can observe that even though we diversified the search and increased the number of subproblems solved on each iteration, the number of columns generated by CG with learning was consistently less than that generated without learning. This discrepancy in the number of columns generated may arise because many of these columns may have minimal or insignificant contributions to the RMP.

For example, in instances with 200 requests, the number of columns generated by ML-CG averaged 638.04, compared to 906.53 using traditional CG, resulting in an 18% reduction. This reduction suggests that ML-CG enables a more efficient search process by focusing on fewer, more relevant columns, while many of the columns generated by the standard CG approach may have minimal or insignificant contributions to the RMP. This observation can be further supported by examining the correlation between the convergence of the objective value and the CPU time. As shown in Figure 2 for certain selected instances, especially those with more requests (50, 100, 200)—given that the proposed algorithm exhibits poor performance compared to the commercial solver or other metaheuristics for small instances—CG with learning demonstrates a faster convergence, highlighting the significance of efficient columns in boosting the search procedure's effectiveness. Therefore, learning can yield significant benefits for solving large-scale data by efficiently generating columns without excessively increasing the number of variables for the RMP.

Table 5. Number of columns generated by CG algorithm with and without learning-based approach

Nb requests	Without learning	With learning
10	4927.49	3738.27
15	3845.37	3039.24
20	3887.67	2926.61
30	2771.32	1993.27
50	1522.45	1102.60
100	1456.92	974.49
200	906.53	638.04

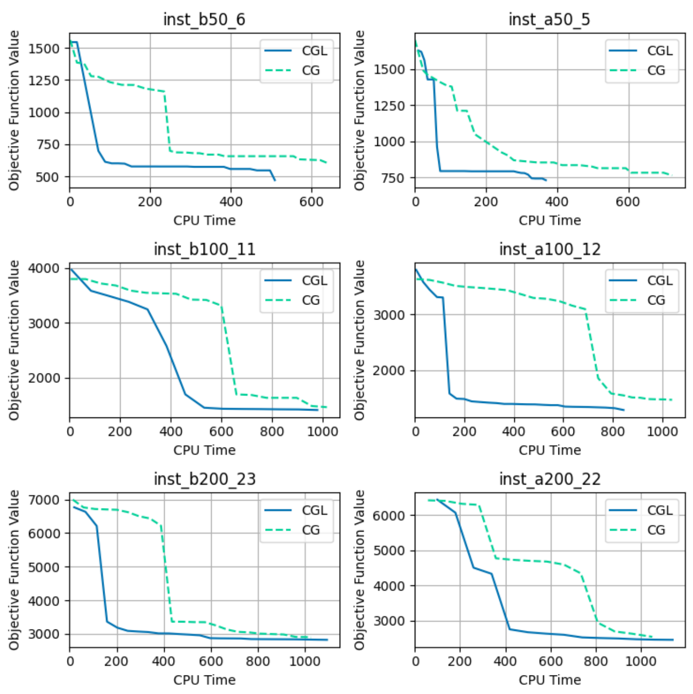

Figure 2. Plots showing the evolution of objective function values over CPU time for selected experiments (inst_b50_6, inst_a50_5, inst_b100_11, inst_a100_12, inst_b200_23, inst_a200_22)

3.2. Comparison with state-of-art algorithms

Tables 6 and 7 present the best objective values and average runtime for the a-type and b-type instances, respectively, achieved by the three algorithms under comparison. Due to the computational complexity of determining optimal solutions for all instances, we omit the gap to optimality. However, for instances with up to 20 requests, previous analyses using the CPLEX solver have confirmed optimality. To compare the performance of these algorithms, We compute the relative gap for each algorithm. This relative gap is calculated as the percentage difference between an algorithm's solution objective value and the best solution found by any of the three algorithms for a given instance. The relative gap Gap_i for algorithm i is defined as:

$$Gap_i = \frac{Obj_i - Obj_{\text{best}}}{Obj_{\text{best}}} \times 100\%$$

Where Obj_i is the solution objective value of algorithm i and Obj_{best} is the best solution objective value found by any of the three algorithms.

We evaluated the performance of each algorithm under the following configurations. All three algorithms used a maximum time limit as the stopping criterion. For instances with up to 15, 50, and 100/200 requests, the time limits were set to 300, 900, and 1200 seconds, respectively. The E-ILS algorithm retained its configuration as detailed in [3], while the CGP algorithm utilized the following parameters: $N^{interval} = 2$, $N^{iterations} = 200$, and $N^{LNS} = 50$. To fully understand these parameters and the CGP algorithm, consult the relevant paper [5]. Our proposed method's configuration involved dynamically adjusting the percentage (p) of requests considered for change within the LBDH. The percentage started at 20% and increased by 10% with each iteration, resetting to 20% after reaching 60%.

Tables 6 and 7 demonstrate that proposed ML-CG algorithm can achieve at most 2.51% (3.86%) less accuracy than the best solution for small-scale a-type instances (b-type). However, as complexity of problems increases, ML-CG outperforms state-of-the-art algorithms, such as CGP and E-ILS, demonstrating its practicality and high overall effectiveness, evidenced by its average gap of 1.08% (1.65%). In contrast, CGP algorithm exhibits moderate performance with average gap of 17.52% (18.76%), while E-ILS algorithm, with the highest average gap of 31.58% (35.27%), is the least effective in approximating optimal solutions. The incorporation of learning mechanisms into ML-CG enhances solution accuracy while maintaining comparable computational efficiency; specifically, its performance on larger instances highlights its potential in addressing DARPDP.

Table 6. Comparison of objective values and computational time for the three algorithms for the a-type instances

		E-ILS			CGP			ML-CG	
Instance	Obj	CPU(s)	Gap(%)	Obj	CPU(s)	Gap(%)	Obj	CPU(s)	Gap(%)
inst_a10_1	331.893	1.19	0	331.893	4.24	0	331.893	3.48	0
inst_a10_2	137.1	16.15	0	197.325	20.33	43.93	148.6917	12.38	8.45
inst_a15_1	272.922	27.97	0	272.922	19.17	0	272.922	18.65	0
inst_a15_2	318.586	17.81	0	329.28	25.01	3.36	322.038	48.4	1.08
inst_a20_2	410.026	93.5	0	487.982	62.36	19.01	410.026	73.67	0
inst_a20_3	353.498	136.26	0	380.765	246.18	7.71	373.017	208.76	5.52
inst_a30_3	909.882	446.44	11.08	1096.379	404.73	33.85	819.104	368.51	0
inst_a30_4	812.369	612.28	48.11	676.759	289.15	23.38	548.504	270.19	0
inst_a50_5	1147.149	848.16	71.31	764.571	723.1	14.18	669.629	368.42	0
inst_a50_6	1271.975	534.43	74.82	1034.511	953.84	42.18	727.585	827.32	0
inst_a100_11	2209.674	989.43	54.97	1714.193	1005.72	20.22	1425.891	882.45	0
inst_a100_12	1860.426	961.07	43.71	1475.75	1039.56	13.99	1294.546	843.35	0
inst_a200_22	4128.984	1007.03	59.96	2875.297	937.22	11.38	2581.325	1038.06	0
inst_a200_23	4600.176	1061.63	78.21	2894.439	1026.01	12.13	2905.262	1093.08	0
Average	1340.33	482.38	31.58	1038.00	482.62	17.52	916.46	432.62	1.08

Table 7. Comparison of objective values and computational time for the three algorithms for the b-type instances

		E-ILS			CGP			ML-CG	
Instance	Obj	CPU(s)	Gap(%)	Obj	CPU(s)	Gap(%)	Obj	CPU(s)	Gap(%)
inst_b10_1	172.758	4.47	0	172.758	4.82	0	172.758	5.02	0
inst_b10_2	197.29	28.14	0	201.72	20.93	2.25	207.511	15.74	5.18
inst_b15_1	271.757	29.5	0	271.757	36.72	0	271.757	44.52	0
inst_b15_2	333.858	120.82	0	454.382	112.25	36.1	351.79	103.07	5.37
inst_b20_2	253.495	349.72	0	354.326	308.49	39.78	269.89	269.83	6.47
inst_b20_3	403.057	671.9	0	457.778	552.35	13.58	427.691	423.12	6.11
inst_b30_3	784.352	871.79	60.92	769.495	613.94	57.87	487.413	456.03	0
inst_b30_4	732.154	629.29	57.59	626.338	589.88	34.81	464.594	520.69	0
inst_b50_5	1210.899	605.95	58.38	996.072	676.27	30.28	764.571	723.1	0
inst_b50_6	767.605	671.15	63.55	604.94	637.37	28.9	469.326	508.57	0
inst_b100_11	2580.933	1114.62	82.33	1467.14	1018.04	3.65	1415.54	980.52	0
inst_b100_12	1993.452	1021.65	46.96	1463.291	1022.63	7.88	1356.466	1044.63	0
inst_b200_22	4423.672	1104.43	80.81	2532.009	1049.51	3.49	2446.644	1139.56	0
inst_b200_23	3733.878	1089.74	43.19	2713.741	1125.07	4.07	2607.598	1065.46	0
Average	1275.65	593.80	35.27	934.70	554.88	18.76	836.68	521.42	1.65

3.3. Discussion

While the ML-CG algorithm demonstrates strong scalability and quality of solutions, its performance can be affected by the selection of hyperparameters for the learning model. Additionally, the existing implementation depends on synthetic datasets, which might not adequately reflect the intricacies of real-world situations. As a result, these aspects restrict the model's ability to generalize, especially when utilized in more diverse and dynamic cases of the DARPDP. Future research should focus on addressing these limitations. To enhance the model's predictive accuracy and robustness, investigating additional features and fine-tuning hyperparameters could be beneficial. The integration of more sophisticated ML approaches, like meta-learning, may enable the model to adjust more effectively to diverse problem characteristics. Additionally, evaluating the methodology on larger, real-world datasets would facilitate the assessment of its effectiveness and generalization in intricate, practical situations. Finally, merging ML-CG with other heuristic techniques could result in hybrid algorithms that boost scalability and flexibility, offering a more versatile solution for the DARPDP and comparable complex optimization challenges. The results demonstrate that integrating ML-CG significantly boosts efficiency when compared to traditional methods. By minimizing the number of generated columns and accelerating convergence, ML-CG proves particularly effective for larger instances. When evaluated against two benchmark algorithms, E-ILS and CGP, ML-CG delivers the best solutions among the three methods for instances a-type (b-type). As problem complexity increases, the advantages of ML-CG become increasingly evident, providing an optimal balance between accuracy and computational time. These findings suggest that ML-CG may offer an efficient and robust approach to complex problems like DARPDP.

4. CONCLUSION

This paper integrates ML into a CG algorithm to solve the DARPDP. Our proposed ML-CG algorithm utilizes a binary classification model to intelligently guide the CG process. This approach leads to a reduction in the number of generated columns by an average of 18% compared to the standard CG, resulting in faster computation times, especially for larger problem instances. ML-CG achieves an average deviation of 2.7% from CPLEX's optimal solutions (for instances where the optimum is known) and demonstrates superior performance compared to both the E-ILS and CGP heuristics, outperforming E-ILS in 60% of the tested cases and consistently outperforming CGP. The current algorithm relies on synthetic datasets and a limited set of features, which may restrict its ability to generalize to real-world scenarios. To enhance its applicability, future work should focus on expanding the training set and testing the algorithm on larger, more diverse datasets. Additionally, exploring meta-learning and hybrid approaches could improve its adaptability and scalability. Further investigation is also needed to assess the performance of the ML-CG framework across a broader range of DARP variants and more realistic scenarios.

FUNDING INFORMATION

The authors acknowledge the financial support of the National Center for Scientific and Technical Research (CNRST), Morocco, under Grant Number 38UH2C2019.

AUTHOR CONTRIBUTIONS STATEMENT

This journal uses the Contributor Roles Taxonomy (CRediT) to recognize individual author contributions, reduce authorship disputes, and facilitate collaboration.

Name of Author	C	M	So	Va	Fo	I	R	D	O	E	Vi	Su	P	Fu
Sana Ouasaid	✓	✓	✓	✓	✓	✓		✓	✓	✓			✓	
Mohammed Saddoune	✓		✓	✓		✓	✓	✓		✓	✓	✓	✓	

C	: **C**onceptualization	I	: **I**nvestigation	Vi	: **Vi**sualization
M	: **M**ethodology	R	: **R**esources	Su	: **Su**pervision
So	: **So**ftware	D	: **D**ata Curation	P	: **P**roject Administration
Va	: **Va**lidation	O	: Writing - **O**riginal Draft	Fu	: **Fu**nding Acquisition
Fo	: **Fo**rmal Analysis	E	: Writing - Review & **E**diting		

CONFLICT OF INTEREST STATEMENT

Authors state no conflict of interest.

DATA AVAILABILITY

This study reuses benchmark data in [3] at https://github.com/SanaaOsd/TestInstancesDARPDP. Additional data were generated using the same procedure for data augmentation. These data are available on request from the corresponding author, [SO].

REFERENCES

[1] H. N. Psaraftis, "A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem," *Transportation Science*, vol. 14, no. 2, pp. 130–154, 1980, doi: 10.1287/trsc.14.2.130.

[2] S. Ouasaid and M. Saddoune, "A dial-a-ride problem with driver preferences," in *7th International Conference Optimization and Applications (ICOA)*, 2021, pp. 1–6, doi: 10.1109/ICOA51614.2021.9442667.

[3] S. Ouasaid and M. Saddoune, "An enhanced approach for the dial a ride problem with drivers preferences," *OPSEARCH*, pp. 1–33, 2024, doi: 10.1007/s12597-024-00843-4.

[4] T. Garaix, C. Artigues, D. Feillet, and D. Josselin, "Optimization of occupancy rate in dial-a-ride problems via linear fractional column generation," *Computers & Operations Research*, vol. 38, no. 10, pp. 1435–1442, 2011, doi: 10.1016/j.cor.2010.12.014.

[5] S. N. Parragh and V. Schmid, "Hybrid column generation and large neighborhood search for the dial-a-ride problem," *Computers & Operations Research*, vol. 40, no. 1, pp. 490–497, 2013, doi: 10.1016/j.cor.2012.08.004.

[6] N. Rahmani, B. Detienne, R. Sadykov, and F. Vanderbeck, "A column generation based heuristic for the dial-a-ride problem," in *6th International Conference on Information Systems, Logistics and Supply Chain (ILS)*, Bordeaux, France, 2016. pp. 1-9.

[7] D.-Y. Lin and C.-L. Hsu, “A column generation algorithm for the bus driver scheduling problem,” *Journal of Advanced Transportation*, vol. 50, no. 8, pp. 1598–1615, 2016, doi: 10.1002/atr.1417.

[8] C. Riley, A. Legrain, and P. Van Hentenryck, “Column generation for real-time ride-sharing operations,” in *Proceedings CPAIOR 2019*, pp. 472–487, 2019, doi: 10.1007/978-3-030-19212-9_31.

[9] Y. Rist and M. Forbes, “A column generation and combinatorial benders decomposition algorithm for the selective dial-a-ride-problem,” *Computers & Operations Research*, vol. 140, 2022, doi: 10.1016/j.cor.2021.105649.

[10] C. Ackermann and J. Rieck, “New optimization guidance for dynamic dial-a-ride problems,” in *International Conference on Operations Research*, pp. 283–288, 2021, doi: 10.1007/978-3-031-08623-6_42.

[11] C. Bongiovanni, M. Kaspi, J.-F. Cordeau, and N. Geroliminis, “A machine learning-driven two-phase metaheuristic for autonomous ridesharing operations,” *Transportation Research Part E*, vol. 165, 2022, doi: 10.1016/j.tre.2022.102835.

[12] S. Dong, “New formulations and solution methods for the dial-a-ride problem,” *Ph.D. dissertation*, University of New South Wales, Sydney, Australia, 2022, doi: 10.26190/unsworks/24100.

[13] A. Mortazavi, M. Ghasri, H. Haghshenas, and T. Ray, “Adaptive logit models for constructing feasible solution for the dial-a-ride problem,” *SSRN*, 2022, doi: 10.2139/ssrn.4307357.

[14] Y. Bengio, A. Lodi, and A. Prouvost, “Machine learning for combinatorial optimization: a methodological tour d’horizon,” *European Journal of Operational Research*, vol. 290, no. 2, pp. 405–421, 2021, doi: 10.1016/j.ejor.2020.07.063.

[15] A. Tahir, F. Quesnel, G. Desaulniers, I. El Hallaoui, and Y. Yaakoubi, “An improved integral column generation algorithm using machine learning for aircrew pairing,” *Transportation Science*, vol. 55, no. 6, pp. 1411–1429, 2021, doi: 10.1287/trsc.2021.1084.

[16] F. Quesnel, A. Wu, G. Desaulniers, and F. Soumis, “Deep-learning-based partial pricing in a branch-and-price algorithm for personalized crew rostering,” *Computers & Operations Research*, vol. 138, 2022, doi: 10.1016/j.cor.2021.105554.

[17] Y. Shen, Y. Sun, X. Li, A. Eberhard, and A. Ernst, “Enhancing column generation by a machine-learning-based pricing heuristic for graph coloring,” in *AAAI Conference on Artificial Intelligence*, vol. 36, no. 9, 2022, doi: 10.1609/aaai.v36i9.21230.

[18] M. Morabit, G. Desaulniers, and A. Lodi, “Machine-learning–based column selection for column generation,” *Transportation Science*, vol. 55, no. 4, pp. 815–831, 2021, doi: 10.1287/trsc.2021.1045.

[19] M. Morabit, G. Desaulniers, and A. Lodi, “Machine-learning–based arc selection for constrained shortest path problems in column generation,” *INFORMS Journal on Optimization*, vol. 5, no. 2, pp. 191–210, Apr. 2023, doi: 10.1287/ijoo.2022.0082.

[20] C. Chi, A. Aboussalah, E. Khalil, J. Wang, and Z. Sherkat-Masoumi, “A deep reinforcement learning framework for column generation,” in *Advances in Neural Information Processing Systems*, vol. 35, pp. 9633–9644, 2022.

[21] J. Gerbaux, G. Desaulniers, and Q. Cappart, “A machine-learning-based column generation heuristic for electric bus scheduling,” *Computers & Operations Research*, vol. 173, 2025, doi: 10.1016/j.cor.2024.106848.

[22] J. E. Mendoza and J. G. Villegas, “A multi-space sampling heuristic for the vehicle routing problem with stochastic demands,” *Optimization Letters*, vol. 7, pp. 1503–1516, 2013, doi: 10.1007/s11590-012-0555-8.

[23] M. Bach, A. Werner, and M. Palt, “The proposal of undersampling method for learning from imbalanced datasets,” *Procedia Computer Science*, vol. 159, pp. 125–134, 2019, doi: 10.1016/j.procs.2019.09.167.

[24] M. Dudjak and G. Martinović, “In-depth performance analysis of SMOTE-based oversampling algorithms in binary classification,” *International Journal of Electrical And Computer Engineering Systems*, vol. 11, no. 1, pp. 13–23, 2020, doi: 10.32985/ijeces.11.1.2.

[25] Y. Sun, X. Li, and A. Ernst, “Using statistical measures and machine learning for graph reduction to solve maximum weight clique problems,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 5, pp. 1746–1760, 2019, doi: 10.1109/TPAMI.2019.2954827.

[26] U. Narayanan, A. Unnikrishnan, V. Paul, and S. Joseph, “A survey on various supervised classification algorithms,” in *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*, Chennai, India, 2017, pp. 2118–2124, doi: 10.1109/ICECDS.2017.8389824.

BIOGRAPHIES OF AUTHORS

Sana Ouasaid received a Ph.D. degree in Computer Science and Artificial Intelligence in 2024 from Hassan II University, Casablanca, Morocco. She currently conducts research at the Laboratory of Machine Intelligence (LIM), Faculty of Science and Technology of Mohammedia. Her research interests include combinatorial optimization, algorithms, and metaheuristics for large-scale problems, with a focus on the integration of optimization techniques and ML. She can be contacted at email: sana.ouasaid1-etu@etu.univh2c.ma.

Mohammed Saddoune obtained a Ph.D. from Polytechnique Montréal, Canada, in 2010. He is currently a Professor of Higher Education, the head of the Computer Science Department, and the Team Leader for Data Valorization and Systems Optimization (VADOS), a team within the Laboratory of Machine Intelligence (LIM), Faculty of Science and Technology of Mohammedia, Morocco. He is also the author or co-author of high-quality articles published in journals such as the European Journal of Operational Research, Computers and Operations Research, and Transportation Science. His research interests include optimization, logistics & transport, data science, and machine learning. He can be contacted at email: mohammed.saddoune@fstm.ac.ma.