Article

Not peer-reviewed version

Escaping Stagnation through Improved Orca Predator Algorithm with Deep Reinforcement Learning for Feature Selection

Rodrigo Olivares * , Camilo Ravelo , Ricardo Soto , Broderick Crawford

Posted Date: 26 March 2024

doi: 10.20944/preprints202403.1533.v1

Keywords: biomimetic orca predator algorithm; deep reinforcement learning; feature selection

Article

Escaping Stagnation through Improved Orca Predator Algorithm with Deep Reinforcement Learning for Feature Selection

Rodrigo Olivares [1,*], Camilo Ravelo [1], Ricardo Soto [2] and Broderick Crawford [2]

[1] Escuela de Ingeniería Informática, Universidad de Valparaíso, 2362905, Valparaíso, Chile; camilo.ravelo@postgrado.uv.cl
[2] Escuela de Ingeniería Informática, Pontificia Universidad Católica de Valparaíso, 2362807, Valparaíso, Chile; ricardo.soto@uv.cl (R.S.); broderick.crawford@pucv.cl (B.C.)
* Correspondence: rodrigo.olivares@uv.cl

Abstract: Stagnation at local optima represents a significant challenge in bio–inspired optimization algorithms, often leading to suboptimal solutions. This paper addresses this issue by proposing a hybrid model that combines the Orca Predator Algorithm with Deep Q–Learning. Orca Predator Algorithm is an optimization technique that mimics the hunting behavior of orcas. It solves complex optimization problems by exploring and exploiting search spaces efficiently. Deep Q–Learning is a reinforcement learning technique that combines Q–Learning with deep neural networks. This integration aims to turn the stagnation problem into an opportunity for more focused and effective exploitation, enhancing the optimization technique's performance and accuracy. The proposed hybrid model leverages the biomimetic strengths of Orca Predator Algorithm to identify promising regions nearby in the search space, complemented by the fine–tuning capabilities of Deep Q–Learning to navigate these areas precisely. The practical application of this approach is evaluated using the high–dimensional Heartbeat Categorization Dataset, focusing on the feature selection problem. This dataset, comprising complex electrocardiogram signals, provided a robust platform for testing the feature selection capabilities of our hybrid model. Our experimental results are encouraging, showcasing the hybrid strategy capability to identify relevant features without significantly compromising the performance metrics of machine learning models. This analysis was performed by comparing the improved method of Orca Predator Algorithm against its native version and a set of state–of–the–art algorithms.

Keywords: biomimetic orca predator algorithm; deep reinforcement learning; feature selection

1. Introduction

Over the years, optimization mechanisms have significantly evolved, in tandem with the rise in scientific knowledge, as evidenced by the considerable progress in this field of study [1]. In this context, artificial intelligence has led the most significant innovations, and within this realm, bio–inspired optimization algorithms, especially those employing biomimetic approaches, have achieved significant benefits [2]. Despite their proven efficiency in solving a variety of complex optimization problems [3], these methods face a crucial challenge: stagnation at local optima. Many works have faced this issue as a problem to be normally solved by using sophisticated exploration techniques to visit other places of the search spaces [4–6]. However, this stagnation can be viewed as an opportunity to exploit promising zones of the search space with more focused and controlled diversification, potentially enhancing the algorithm's performance and accuracy.

In swarm intelligence algorithms, the stagnation problem typically occurs when solutions converge to a local optimum rather than the desired global optimum [7]. This is primarily due to artificial agents being guided by local information in their vicinity, leading to premature convergence to suboptimal solutions. When the swarm gets trapped in a local optimum, exploring other regions of the search space becomes challenging. This convergence is altered with random exploration processes. This implies that targeted tactics do not always enrich the search for the global optimum but is sometimes guided by a more random process. Although this approach may open new areas for exploration [8,9], it does not necessarily ensure a more effective path to optimal solutions.

 doi:10.20944/preprints202403.1533.v1

In this research, we pioneer the synergy of the Orca Predator Algorithm (OPA) and Deep Q–Learning (DQL), aiming to harness the strengths of both to identify and exploit promising solutions more effectively, building on the foundational work detailed in [10]. Traditional methodologies often fall short by prematurely converging on local optima; our strategy, however, capitalizes on such convergences, treating them as gateways for directed exploration towards areas of promise. The selection of OPA was meticulous, driven by its intricate parameter requirements and the profound influence of its calibration. This methodology is designed to be adaptable, suitable for application to other techniques with akin functional characteristics. The fusion of OPA and DQL marks a notable innovation in tackling optimization challenges, merging OPA's biomimetic prowess in swiftly pinpointing areas worth exploring with DQL's precision in refining searches within these zones.

Despite our proposal's significant advancements, we also face challenges inherent in integrating optimization techniques with deep learning algorithms [11]. One of the main challenges in this field is the management of high–dimensional search spaces, which often lead to a higher probability of generating similar solutions. Additionally, the computational complexity of combining OPA with DQL raises practical concerns regarding computational resources and execution time, particularly in large–scale problems [12]. We acknowledge the potential of our approach in various optimization scenarios while also considering that its applicability and effectiveness may need to be tailored to the specific characteristics and complexities of each unique problem set.

The proposal will be evaluated on the feature selection (FS) problem, aiming to identify a subset of features that offers comparable results with less computational effort. This is particularly relevant in datasets with irrelevant, noisy, and high–dimensional data [13,14]. The dataset selected for evaluation is the Heartbeat Categorization Dataset [15], available on Kaggle. This dataset contains electrocardiogram signals of heartbeats, which can be normal or affected by various arrhythmias and myocardial infarctions. The dataset includes 188 features, reflecting the dimensionality of our problem. To validate our proposal, the improved Orca Predator Algorithm will be rigorously compared with its native version and various established techniques that are already considered state–of–the–art, underscoring its competitive edge and potential for setting new benchmarks in the field.

This research is committed to exploring the frontiers of bio–inspired optimization methods and machine learning, seeking not only to advance in theory and methodology but also in the practical application of these techniques to real–world problems. With a focus on feature selection in complex datasets, this study aims to overcome current limitations and pave new paths for future research in this exciting and rapidly evolving field.

The structure of this work is presented as follows: Section 2 provides a robust analysis of the current work on hybridizations between learning techniques and metaheuristics. Section 3 details the conceptual framework of the study, emphasizing the biomimetic principles of OPA, the formal description of DQL, and the formulation of FS. Section 4 explains how reinforcement learning is integrated into the orca predator algorithm. Section 5 outlines the phases of the experimental design. Section 6 discusses the results achieved, underscoring the approach's effectiveness in feature selection. Finally, the conclusions and future work are presented in Section 7.

2. Related Work

In recent years, the integration of swarm intelligence algorithms and machine learning has been extensively researched [16]. Various approaches have been described to implement self–adaptation and learning capabilities in these techniques. For example, in [17], the virus optimization algorithm was modified to improve the capabilities of the parameters. The performance was compared with different optimization instances, showing similar or improved performance for the enhanced version. Similarly, in [18], the firefly algorithm was improved to automatically calculate the parameter a that controls the balance between exploration and exploitation. In [19], the cuckoo search algorithm is modified to balance the intensification and the diversification phases. The work in [20] proposes to improve the artificial bee colony by incorporating self–tuning of its agents, aiming to improve the convergence

 doi:10.20944/preprints202403.1533.v1

relationship by altering the parameter controlled during execution. A similar work is observed in [21], where the differential evolution strategy is modified by adding self–tuning to the scalability factor and the crossover ratio to increase the convergence rate. The work [22] describes an improvement of the discrete particle swarm optimization algorithm, which includes adaptive parameter control to balance social and cognitive learning. A new formulation updates the probability factor p_i in the Bernoulli distribution, which updates the parameters R_1 (social learning) and R_2 (cognitive learning). Following this line, in [23] and [24], self–adaptive evolutionary algorithms were proposed. The former details an improvement through a population of operators that change according to a punishment and reward scheme, depending on the operator's quality. The latter presents an improvement where the crossover and mutation probability parameters are adapted to balance exploration and exploitation in the search for solutions. Both cases show outstanding results. In [25], a self–adjustment was applied to the flower pollination algorithm, balancing exploration and exploitation during execution and using S_p as an adaptive strategy. A recent wolf pack algorithm was altered to self–adjust its parameter w that controls the perception of the prey's scent [26]. Here, the new version intensifies the local search towards more promising areas. Finally, the work [27] proposes integrating the autonomous search paradigm in the dolphin echolocation algorithm for population self–regulation.

The integration of metaheuristics with machine learning techniques, regression, and clustering has also been the subject of studies [28–32]. For example, in [33], an evolutionary algorithm is proposed to handle parameters and operators by integrating a controlling module that applies learning rules to measure the impact and assigns resets to the set of parameters. Under this paradigm, the work in [34] explores the integration of the variable neighborhood search algorithm with reinforcement learning, applying reactive techniques for parameter adjustment and selecting local searches to balance exploration and exploitation phases. In [35], an algorithm inspired by the behavior of a beetle can improve CNN performance in image classification tasks. Results show a high success rate for the proposed hybridization. In [36], a machine learning model was developed using support vector machines to predict the quality of solutions for a problem instance. This solution then adjusts the parameters and guides the metaheuristics towards more promising search regions. In [37] and [38], the authors propose the integration of PSO with regression models and clustering techniques, respectively. In [39], another combination of PSO and classifier algorithms is presented to de–parameterize the optimization method. In this approach, a previously trained model is used to classify the solutions found by the particles, improving the exploration of the search space and the quality of the obtained solutions. Similarly to previous works, in [40], PSO is again improved with a learning model for parameter control, obtaining competitive performance compared to other parameter adaptation strategies. In [41], the hybridization of PSO, Gaussian process regression, and support vector machine is presented for real–time parameter tuning. The study concluded that hybridization offers superior performance compared to traditional approaches. The work presented in [42] integrates randomized priority search with the inductive decision tree data mining algorithm for parameter tuning through a feedback loop. Finally, in [43], the authors propose the integration of ant colony optimization algorithms derived with fuzzy logic for controlling the parameters of the pheromone evaporation rate, the probability factor of exploration, and the number of ants to solve the feature selection problem.

More specifically, when reviewing studies on integrating metaheuristics and reinforcement learning, we find many works where combining these techniques improves the search in optimization problems [44]. For example, in [45], the authors proposed the integration of bee swarm optimization with Q–Learning to improve its local search. In this approach, the artificial individuals become intelligent agents that gain and accumulate knowledge as the algorithm progresses, thus improving the effectiveness of the search. Along the same lines, [46] proposes integrating a learning–based approach to the ant colony algorithm to control its parameters. This is done by assigning rewards to the parameter change in the algorithm, storing them in a matrix, and learning the best values to apply to each parameter at runtime. In [47], another combination of a metaheuristic algorithm with reinforcement learning techniques is proposed. In this case, the tabu search was integrated with Q–Learning to find

 doi:10.20944/preprints202403.1533.v1

promising regions in the search space when the algorithm is stuck in a local optimum. The work published in [48] also explored the application of reinforcement learning techniques in the context of an optimization problem. Here, the randomized biased heuristic with reinforced learning techniques was studied to consider the variations generated by the change in the rewards obtained. In [49], the implementation of a stateless Q–Learning algorithm is presented to help train neural networks to classify medical data. This approach used a parameter tuning process in radial–based neural networks via stateless Q–Learning. Although the latter is not an optimization algorithm, the work is relevant to this proposal. Finally, in [50], it is proposed to address the load balancing issue in cloud computing. This solution employs the Fire Hawk Optimizer combined with Deep Q–Learning to enhance the multi–objective programming problem, aiming to minimize time, cost, and resource utilization.

Integrating bio–inspired techniques and machine learning also plays a fundamental role in feature selection. This approach is an essential component in the effectiveness of machine learning models. Recent research has explored various methodologies and techniques to enhance feature selection. For instance, [51] offers a comprehensive review of the use of metaheuristic algorithms in feature selection, suggesting paths for future research in developing or modifying these algorithms for classification tasks. In [52], the performance of recent metaheuristic algorithms in feature selection is evaluated, highlighting the challenges and opportunities in this field.

The models presented in [53,54] demonstrate the application of hybrid approaches that combine optimization techniques and deep reinforcement learning for feature selection in network intrusion detection. Additionally, [55] compiles a series of research on feature selection, where a taxonomy of objective functions is presented, and related metrics are classified into four categories: classifiers, metaheuristics, features, and statistical tests. This classification facilitates effective comparison of different feature selection methodologies, focusing on metaheuristics' implementation and hybridization strategies.

Finally, [56] introduces a technique for predicting customer churn in the telecommunications sector. The Archimedes Optimization Algorithm is employed for optimal feature selection, combined with a hybrid deep learning model for customer churn prediction. The study is notable for its focus on efficient feature selection and hyperparameter tuning, achieving significant improvements in the accuracy and effectiveness of the predictive model. The study's results indicate the high efficiency of the proposed technique compared to other methods, achieving outstanding accuracy.

3. Preliminaries

In this section, we introduce the conceptual framework of our study, focusing on the biomimetic principles underlying the Orca Predator Algorithm, providing a formal description of Deep Reinforcement Learning through Deep Q–Learning, and outlining the formulation of Feature Selection. This framework forms the foundation of our approach, integrating these distinct yet complementary components to address the challenges at hand.

3.1. Biomimetic Orca Predator Algorithm

Orca predator algorithm is a biomimetic method that simulates the hunting behavior of the orca [57]. Orcas are highly social marine animals that share prey and temporarily leave their group. They use sonar to explore the aquatic environment and communicate with their pod to plan hunting tactics and feed. An orca is a n-dimensional vector that stores a potential solution. The population is represented by $X = [x_1, x_2, \ldots, x_n]^T$.

The algorithm presents two phases: the chase phase and the attack phase. The first one is divided into two behaviors: drive the prey (to the surface) and encircle the prey. Here, OPA uses a parameter p to adjust the probability of the orca performing these two processes separately. If $p < r$, the driving phase will be performed. Otherwise, the encircling phase will be performed. Here, r is randomly generated value in $[0, 1]$. The second phase is called attack and simulates the hunting behavior of orcas,

 doi:10.20944/preprints202403.1533.v1

employing tactics to close the distance, subdue the prey, and consume it. The algorithm's success depends on sensory input, decision–making, and potential collaboration with other agents.

3.1.1. Chase Phase

This mechanism is based on specific conditions. The first condition occurs when the small fish shoal reduces the orca's swimming space dimension. The second condition is the opposite: the larger the fish shoal, the larger the hunting environment. Considering these conditions, two techniques are abstracted for pursuing the prey.

$$v^t_{chase,1,i} = a \times (d \times x^t_{best} - F \times (b \times M^t + c \times x^t_i)) \tag{1a}$$

$$v^t_{chase,2,i} = e \times x^t_{best} - x^t_i \tag{1b}$$

$$M = \frac{\sum_{i=1}^{N} x^t_i}{N} \quad \wedge \quad c = 1 - b \tag{1c}$$

$$x_{new} = \begin{cases} x^t_{chase,1,i} = x^t_i + v^t_{chase,1,i} \; if \; q > rand \\ x^t_{chase,2,i} = x^t_i + v^t_{chase,2,i} \; if \; q \leq rand \end{cases} \tag{1d}$$

The velocity and position are defined in Equations (1a), (1b), (1c), and (1d). Here, $v^t_{chase,1,i}$ indicates the first pursuit method, and $x^t_{chase,1,i}$ represents the position, both of the i-th orca at time t. Then, $v^t_{chase,2,i}$ and $x^t_{chase,2,i}$ represent the same for the second pursuit method. Similarly, x^t_i denotes the general position of the i-th orca at time t, and x^t_{best} indicates the position of the orca that is currently closest to the prey or has the most effective pursuit strategy at time t. M represents the average position of the group of orcas, a, b, and d are random numbers between $[0, 1]$, respectively. Similarly, e is a random number between $[0, 2]$, F stores the attraction force or influence that one agent exerts over another, and q is a number between $[0, 1]$ that represents the probability of choosing a particular prey pursuit method.

Now that the fish school is on the surface, the second act is to encircle the prey. Orcas use sonar to communicate and determine their next position based on nearby orcas. Here, we assume that orcas are positioned based on the positions of three randomly selected orcas. Then their position after moving can be calculated, as shown in Equations (2a), (2b), and (2c).

$$x^t_{chase,3,i,k} = x^t_{j_1,k} + u \times (x^t_{j_2,k} - x^t_{j_3,k}) \tag{2a}$$

$$u = 2 \times (rand - 0.5) \times \frac{MaxIter - t}{MaxIter} \tag{2b}$$

$$x_{new} = \begin{cases} x^t_{chase,i} = x^t_{chase,i} \; if \; f(x^t_{chase,i}) < f(x^t_i) \\ x^t_{chase,i} = x^t_i \; if \; f(x^t_{chase,i}) \geq f(x^t_i) \end{cases} \tag{2c}$$

where $MaxIter$ represents the maximum number of iterations, j_1, j_2, and j_3 indicate the three randomly selected orcas from the population, with $j_1 \neq j_2 \neq j_3$. $x^t_{chase,3,i,k}$ is the position of the i-th orca after choosing the third pursuit method in iteration t.

During the pursuit, orcas detect the location of prey through sound and adjust their positions accordingly. Orcas will continue pursuing the fish if they perceive it is approaching; otherwise, they will remain in their original position.

 doi:10.20944/preprints202403.1533.v1

3.1.2. Attack Phase

Orcas surround their prey and then take turns entering the circuit formed to attack the prey, thereby feeding. Subsequently, they return to their original position within the enclosure to replace another orca. Suppose four orcas correspond to the four ideal attack positions in the circle. Based on the positions of these four orcas, other orcas may enter the circuit if they wish to do so. The direction in which the orcas wish to return to the enclosure circle after feeding to replace other orcas can be determined according to the positions of nearby orcas selected randomly. This behavior is mathematically modeled in Equations (3a), (3b), and (3c).

$$v^t_{attack,1,i} = (x^t_{1st} + x^t_{2nd} + x^t_{3rd} + x^t_{4rd})/4 - x^t_{chase,i} \tag{3a}$$

$$v^t_{attack,2,i} = (x^t_{chase,j_1} + x^t_{chase,j_2} + x^t_{chase,j_3})/3 - x^t_i \tag{3b}$$

$$v^t_{attack,i} = x^t_{chase,i} + g_1 \times v^t_{attack,1,i} + g_2 \times v^t_{attack,2,i} \tag{3c}$$

where $v^t_{attack,1,i}$ represents the velocity vector of the i-th orca hunting the prey at iteration t, $v^t_{attack,2,i}$ indicates the velocity of the vector of the i-th orca returning to the rodeo circuit at iteration t, x^t_{1st}, x^t_{2nd}, x^t_{3rd}, and x^t_{4th} represent the four orcas in the best position. j_1, j_2, j_3 are the three orcas randomly selected from the population seen in the pursuit and where $j_1 \neq j_2 \neq j_3$. $v^t_{attack,i}$ represents the position of the i-th orca at iteration t after the attack phase, g_1 is a random value between $[0, 2]$, and g_2 is a random number between $[-2.5, 2.5]$.

After this, the orcas use sonar to locate the prey and adjust their positions, similar to the pursuit process. The minimum limit value (lb) of the potential range of the problem is used to determine the orca's position according to the conditions of the algorithm in [57].

Algorithm 1 details the orca predator procedure. The pseudocode requires the definition of inputs such as population size (S), probabilities for selecting methods (p and q), and dimensionality (n), aiming to produce the global best solution. Thus, the algorithm begins by initializing the objective function and randomly computing the first generation of S orcas (Line 1). For each orca and each decision variable, positions and velocities are assigned randomly (Lines 4–6). The fitness of each orca is calculated, and the best orca is found. If an orca is better than the global best, it updates the global best solution.

Subsequently, the algorithm enters a loop to produce generations of orcas until a predefined iteration limit is reached (Line 16). Within this loop, each orca is processed. The algorithm selects three random orcas (Line 18). Based on the probability p, the algorithm decides whether to perform the chase phase to drive to the prey or encircle the prey for each variable (Lines 19–29). If p is more significant than a random value, the chase phase is executed by either driving to the prey or encircling it based on the value of q (Lines 20–28). If p is less than a random value, the attack phase is performed for each variable (Lines 29–33). The fitness of each orca is computed again (Line 34), and the global best is updated if an orca is found to be better (Lines 36–40). The algorithm concludes by returning the post–processed results and visualization (Line 42).

3.2. Deep Reinforcement Learning

Reinforcement Learning (RL) is an advanced machine learning technique focusing on autonomous agents striving to maximize rewards through their actions [58]. These agents learn by trial and error, identifying actions that lead to greater rewards, both immediate and long–term, a distinguishing feature of RL [59].

In the RL process, agents continually interact with their environment, involving key components such as the value function, policy, and occasionally, a model of the environment [60–63]. The value function evaluates the effectiveness of the agent's actions in the environment, and the agent's policy adjusts based on received rewards.

 doi:10.20944/preprints202403.1533.v1

Algorithm 1: Pseudocode for the orca predator method.

Input: S: population size; p: probability to select driving prey or encircle prey method; q: probability to select a method when the orca group is small or large; n: dimensionality.
Result: The global best solution

objective function $f(\vec{x})$, $\vec{x} = \langle x_1, \ldots, x_n \rangle$
// produce the first generation of S orcas, randomly.
foreach orca o, $(\forall\, i = \{1, \ldots, S\})$ **do**
    foreach variable j, $(\forall\, j = \{1, \ldots, n\})$ **do**
        position $x_{ij}^{t=0} \leftarrow$ Random() and velocity $v_{ij}^{t=0} \leftarrow$ Random()
    end
    Compute fitness
end
Find global best
foreach orca o, $(\forall\, i = \{1, \ldots, S\})$ **do**
    if orca o is better than global best **then**
        Update global best
    end
end
// produce generations of S orcas.
while iteration up to limit **do**
    foreach orca o, $(\forall\, i = \{1, \ldots, S\})$ **do**
        Select randomly three orcas
        if $p <$ Random() **then**
            if $q >$ Random() **then**
                foreach variable j, $(\forall\, j = \{1, \ldots, n\})$ **do**
                    Chase phase for driving to the prey via Equations (1a), (1b), (1c), and (1d)
                end
            else
                foreach variable j, $(\forall\, j = \{1, \ldots, n\})$ **do**
                    Chase phase for encircling the prey via Equations (2a), (2b), and (2c)
                end
            end
        else
            foreach variable j, $(\forall\, j = \{1, \ldots, n\})$ **do**
                Attack phase via Equations (3a), (3b), and (3c)
            end
        end
        Compute fitness
    end
    foreach orca o, $(\forall\, i = \{1, \ldots, S\})$ **do**
        if orca o is better than global best **then**
            Update global best
        end
    end
end
return post–process results and visualization

Q–Learning, a fundamental technique in RL, focuses on establishing a function to assess the effectiveness of a specific action a_t in a given state s_t at time t [64,65]. The Q function updates using Equation (4):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \times \left[r_{t+1} + \gamma \times \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right] \tag{4}$$

 doi:10.20944/preprints202403.1533.v1

where α is the learning rate and γ is the discount factor. r_{t+1} is the reward obtained after performing action a_t.

Deep Reinforcement Learning (DRL) integrates deep learning techniques with RL, enabling addressing problems with greater complexity and dimensionality [66]. In DRL, deep neural networks approximate value functions or policies. Deep Q–Learning is a DRL technique that uses a neural network to approximate the Q value function to measure the expected cumulative reward for acting in a particular state. The Q value function learns through an iterative process when the agent takes actions in the environment and receives rewards.

In DQL, the Q function is defined as $Q(s_t, a_t; \theta)$, where s_t represents the current state, a_t the action taken by the agent at time t, and θ the network weights [67]. The Q function is updated by using Equation (5):

$$Q(s_t, a_t; \theta) \leftarrow Q(s_t, a_t; \theta) + \alpha \times \left[r_{t+1} + \gamma \times \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta^-) - Q(s_t, a_t; \theta) \right] \tag{5}$$

where, s_{t+1} and a_{t+1} represent the new state and action at time $t+1$, respectively. The parameter α, known as the learning rate, regulates the magnitude of the update to the Q value function at each training step. A high value of α accelerates the adaptation of the Q function to environmental changes, being beneficial in the initial stages of learning or in highly dynamic environments. However, an excessively high learning rate can cause a disproportionate response to random variations in rewards, leading to instability in learning [67]. On the other hand, a lower α promotes a more gradual and stable learning process but may prolong the time required to achieve convergence. The factor γ, known as the discount factor, assigns relevance to future rewards compared to immediate ones. A value close to 1 encourages the agent to give almost equal importance to long–term rewards as to current ones, thereby incentivizing strategies that aim to maximize long–term benefits. In contrast, a lower γ prioritizes short–term rewards, which is advantageous in scenarios where future rewards are less predictable or when effective policies are required in reduced time horizons. Finally, r_{t+1} is the reward received after executing the action a_t in the state s_t. θ^- represents the parameters of a target neural network that is periodically updated with the values of θ to improve training stability.

A distinctive feature of DQL is its use of Replay Memory, which constitutes a crucial part of its learning architecture [68,69]. Replay Memory stores the agent's past experiences in the form of tuples $\langle s_t, a_t, r_{t+1}, s_{t+1} \rangle$. Each tuple represents an individual experience of the agent, where s_t is the current state, a_t the action taken, r_{t+1} the reward received, and s_{t+1} the subsequent state. This approach of storing and reusing past experiences aids in enhancing the efficiency and effectiveness of the learning process, allowing the agent to learn from a more diverse set of experiences. With that, it reduces the correlation between consecutive learning sequences, a critical aspect to avoid over–dependence on recent data and to promote more generalized learning. Additionally, the mini–batch technique is implemented for sampling experiences from Replay Memory during the training process [70]. Instead of learning from a single experience at each step, the algorithm randomly selects a mini–batch of experiences. This batch sampling method contributes to the stability of learning by promoting the independence of samples and allows for more efficient use of computational resources.

Finally, learning in DQL is guided by a loss function according to Equation (6), which measures the discrepancy between the estimated Q and target values.

$$\text{Loss}(\theta_t) = E \times \left[(y - Q(s_t, a_t; \theta))^2 \right] \tag{6}$$

where y is the target value, calculated by Equation (7):

$$y = r_{t+1} + \gamma \times \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \theta^-) \tag{7}$$

Here, r_{t+1} is the reward received after taking action a_t in the state s_t, and γ is the discount factor, which balances the importance of short–term and long–term rewards. The formulation

 doi:10.20944/preprints202403.1533.v1

$\max_{a'} Q(s_{t+1}, a_{t+1}; \theta^-)$ represents the maximum estimated value for the next state s_{t+1}, according to the target network with parameters θ^-. $Q(s_t, a_t; \theta_t)$ is the Q value estimated by the evaluation network for the current state s_t and action a_t, using the current parameters θ_t. In each training step in DQL, the evaluation network receives a loss function backpropagated based on a batch of experiences randomly selected from the experience replay memory. The evaluation network's parameter, θ, is then updated by minimizing the loss function through the Stochastic Gradient Descent (SGD) function. After several steps, the target network's parameter, θ^-, is updated by assigning the latest parameter θ to θ^-. After a period of training, the two neural networks are trained stably.

Lastly, the epsilon–greedy [71] strategy in DQL crucially balances exploration and exploitation. It does this by initially favoring exploration with a probability of ϵ for selecting actions randomly and gradually shifting towards exploitation, choosing actions that maximize the Q function with a probability of $1 - \epsilon$. As the agent gains more knowledge, ϵ decreases, effectively transitioning from exploration to exploitation. This approach prevents the agent from settling into suboptimal strategies, ensuring adaptability and learning from diverse experiences.

3.3. Feature Selection

Feature selection is an integral component in data preprocessing, which identifies and selects a subset of relevant features (variables, predictors) for use in model construction [72]. It is crucial in enhancing the efficiency of machine learning models by eliminating redundant and irrelevant data, thus streamlining the computational process and potentially improving model performance [73,74].

Formally, let us consider a dataset with a set of features $D = \{x_1, x_2, x_3, \ldots, x_n\}$ and a class label Y. The objective of feature selection is to find a subset $D' \subset D$ such that $|D'| < |D|$ and the predictive power of D' concerning Y is maximized. This process involves evaluating the importance of each feature in D and retaining only those that contribute significantly to our understanding of Y.

There are three primary approaches to feature selection [75,76]: filter methods, wrapper methods, and embedded methods. Filter methods rank features based on statistical measures independent of any learning algorithm. Wrapper methods use a predictive model to score feature subsets and are computationally intensive as they involve training models on different subsets. Embedded methods perform feature selection as part of the model training process and are specific to particular algorithms.

We focus on the wrapper approach, particularly its application in selecting features that provide high classification accuracy while minimizing computational resources. In this approach, subsets of features are iteratively evaluated using a specific learning algorithm, and the subset that yields the highest classification performance is selected. This process can be computationally expensive due to the need to train a model for each subset.

Our work utilizes an improved bio–inspired metaheuristic algorithm to optimize the feature selection process. Each potential solution in this algorithm represents a subset of features encoded as a binary string where one indicates the inclusion of a feature and zero is its exclusion. The algorithm iteratively refines these solutions, guided by the learning model's performance on these subsets [77]. The performance of these subsets is evaluated using metrics like F_1 score and accuracy. F_1 score provides a balance between precision and recall, while accuracy measures the proportion of correctly predicted instances [78].

Feature selection is undoubtedly a vital step in the data preprocessing phase, aiding in reducing the dimensionality of the dataset, curtailing overfitting, and enhancing the generalizability and efficiency of the learning models [79]. Therefore, our work contributes to this field by applying an advanced metaheuristic approach to optimize the feature selection process, thereby striking a balance between model complexity and computational efficiency.

4. Developed Solution

Our proposed solution integrates the Orca Predator Algorithm with Deep Q–Learning for dynamically optimizing OPA's parameters. This approach leverages the predatory behavior of orcas

 doi:10.20944/preprints202403.1533.v1

and DQL's management of large state and action spaces to enhance feature selection efficiency. OPA, inspired by orca hunting strategies [57], offers an innovative optimization approach, while DQL addresses high–dimensional space challenges in complex combinatorial problems [80].

DQL plays an essential role in transitioning towards exploitation, particularly in the later stages of the optimization process. As OPA explores the solution space by mimicking orcas' collaborative and adaptive strategies in their natural habitat, DQL refines this exploration by targeting the most promising regions. This shift from exploration to exploitation is efficiently managed through DQL's epsilon–greedy policy, significantly enhancing the selection of optimal actions as the model accumulates knowledge.

OPA operates as a metaheuristic where each orca represents an independent search agent, exploring the solution space characterized by a binary vector. These vectors represent selected features from the database, with each vector being individually evaluated through a variety of machine learning algorithms, such as Decision Trees (DT), Random Forests (RF), Support Vector Machines (SVM), and Extremely Randomized Trees (ERT). F_1 score and accuracy derived from these evaluation methods provide crucial feedback to OPA about the quality of the solutions, thereby allowing the orcas to adjust their strategies and explore bounded promising areas of the solution space.

DQL intervenes in this dynamic adaptive process by providing a sophisticated reinforcement learning–based strategy to regulate the operational parameters of OPA. DQL accumulates experiences from the evaluations of the orcas, including rewards based on the diversity of solutions and performance metrics. These experiences are essential for updating the decision policies in OPA, precisely calibrating the probabilities and parameters that guide both the exploration and exploitation of the search space. Through the implementation of replay memory, DQL learns from past experiences, enabling it to anticipate and maximize future rewards. This leads to a significant and iterative improvement in feature selection, constantly refining the solution search process.

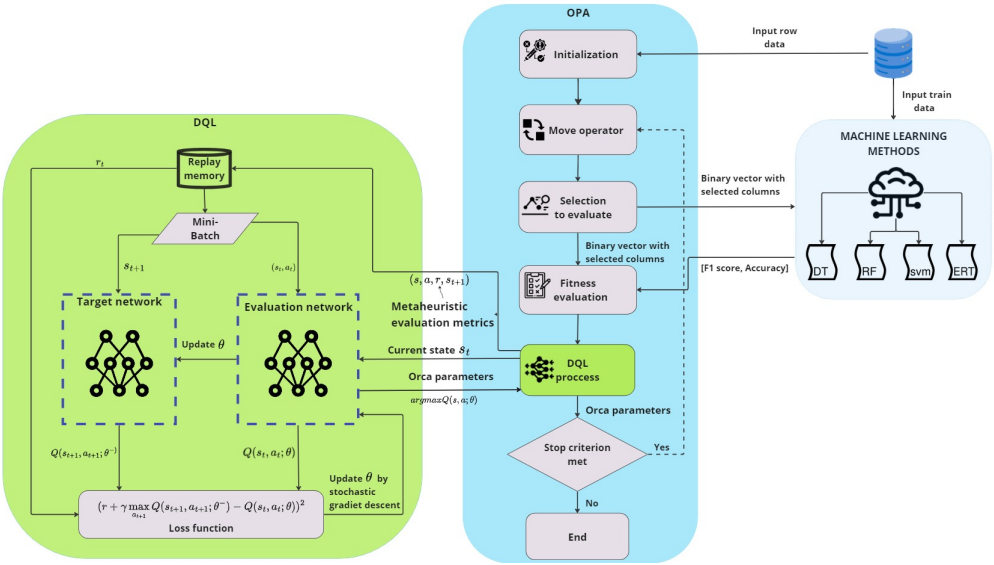


Figure 1. Scheme of the hybrid proposal combining OPA and DQL for the feature selection problem.

Figure 1 illustrates in detail the flow of the proposed solution process, highlighting the synergy between OPA and DQL in this advanced optimization method. It underscores how the interaction between these two components leads to a more efficient and effective search for optimal solutions, combining OPA's nature–inspired exploration with DQL's outcome–oriented and adaptive learning strategy.

The core of our methodology is embodied in the pseudocode of Algorithm 2. The algorithm begins by taking data from a dataset, such as dimensionality and row data, for training to find the

 doi:10.20944/preprints202403.1533.v1

global best solution. Then, it continues by initializing a swarm of orcas and setting up the initial state and action parameters based on performance metrics.

Algorithm 2: Pseudocode for the improved orca predator method.

Input: Data from dataset, i.e, dimensionality and row data for training.
Input: OPA's parameters.
Result: The global best solution
1 Initialize a swarm S of orcas
2 state ← performance metrics
3 action ← initial values for OPA's parameter
4 **while** *iteration up to limit* **do**
5 **foreach** *orca* o, $(\forall\, i = \{1, \ldots, S\})$ **do**
6 Select randomly three orcas
7 Update position and velocity of orca
8 Call training phase to orca o and compute its fitness via Eq. (8)
9 **if** *orca* o *is better than global best* **then**
10 Update global best
11 Calculate reward based on performance metrics via Eq. (9)
12 state, action ← `DQLProcess(`*state, action, reward*`)` via Eq. (11)
13 **end**
14 **end**
15 **end**
16 **return** Post–process and visualize results
17 **Function** `DQLProcess(`*state*`):`
18 Initialize Q–estimation neural network
19 Initialize memory buffer for storing experiences
20 Select action using epsilon–greedy strategy
21 Execute action and observe new state, reward
22 Store experience in memory buffer
23 **if** *memory buffer is large enough* **then**
24 Extract mini–batch of experiences
25 **foreach** *experience in mini–batch* **do**
26 Calculate target value using Q–Learning
27 Update neural network with state and target value
28 **end**
29 Update epsilon value
30 **end**

The main loop of the hybridization continues until a specified iteration limit equal to the original version is reached. During each iteration, every orca in the swarm is processed. First, it selects three random orcas and updates the position and velocity of each orca. These two steps are also equivalent to the original version. Next, a machine learning technique is invoked to compute fitness (Line 8). This process uses the solution stored in the orca, corresponding to a n–dimensional vector composed of zeros and ones. The machine learning method generates a trained model only with selected columns (ones of the vector).

We strongly emphasize critical performance metrics in evaluating our machine learning model, notably the F_1 score and accuracy. We have adopted a comprehensive multi–objective strategy to enhance the model's effectiveness. This approach doesn't just focus on maximizing these essential metrics but also includes the solution diversity metric and a reduction component aimed at optimizing the balance between the number of features used and the total available features. This optimization is crucial for streamlining our algorithm and making it more efficient in feature selection.

 doi:10.20944/preprints202403.1533.v1

Integrating these individual metrics into a unified measure, the Feature Efficiency Index (FEI), is central to this study. The maximum FEI value embodies the culmination of our analytical efforts, synthesizing the model's performance across various dimensions. It's a holistic metric that reflects not only the F_1 score and accuracy but also the diversity of the solutions and the effectiveness of our reduction strategy. We achieve a multi–faceted evaluation of the model via objective functions by calculating the FEI through a carefully designed linear scalarization method, detailed in Equation (8). This approach allows us to gauge the overall efficiency and effectiveness of the model, providing a comprehensive view of its performance in utilizing the selected features.

$$\underset{max}{\text{FEI}} = \sum_{(i,j)_{i\neq j} \in K} \overbrace{\frac{\widehat{c} - f_i(\vec{x})}{\widehat{c} - e_i(\vec{x}^{best})} \times \omega_i}^{min} + \overbrace{\frac{f_j(\vec{x})}{e_j(\vec{x}^{best})} \times \omega_j}^{max} \quad \wedge \sum_{(i,j)_{i\neq j} \in K} \omega_{(i,j)} = 1,\ \omega_{(i,j)} \geqslant 0 \tag{8}$$

Here, $\omega_{(i,j)}$ represents the weight of objective functions. The $\widehat{c}$ values represent an upper bound to minimize single–objective functions. Finally, $f_{(i \text{ or } j)}(\vec{x})$ is the single–objective function, and $e_{(i \text{ or } j)}(\vec{x}^{best})$ stores the best value encountered independently. For this study, f_1 and f_2 describe the maximization of F_1 score and accuracy, respectively. Conversely, f_3 corresponds to the minimization of the heterogeneity of solutions (diversity). Last, f_4 represents the maximization of not used columns (reduction). With that, a reward is generated based on performance metrics. Changes in the F_1 score, accuracy, diversity, and reduction are computed by comparing the current values with previous values. Equation (9) details the used formula.

$$\begin{aligned} reward = \delta \times (&F_1 \text{ score}_{new} - F_1 \text{ score}_{previous}) + \\ \epsilon \times (&\text{accuracy}_{new} - \text{accuracy}_{previous}) + \\ \zeta \times (&\text{diversity}_{previous} - \text{diversity}_{new})) + \\ \eta \times (&\text{reduction}_{previous} - \text{reduction}_{new})) \end{aligned} \tag{9}$$

where δ, ϵ, ζ, and η serve as weights, each assigned to quantify the significance of corresponding terms in the reward calculation. The terms F_1 score$_{new}$ and $F1$ score$_{previous}$ represent the F_1 scores of the current and previous best orcas, respectively. Similarly, $accuracy_{new}$ and $accuracy_{previous}$ refer to the accuracies of the current and previous best orcas, respectively. Both F_1 scores and accuracies, values close to 1 indicates a good performances.

Diversity is evaluated using the Euclidean distance between the positions of each pair of orcas across the entire swarm. For a set of orcas, with each individual's position denoted by the vector p_i, diversity is determined in accordance with Equation (10).

$$diversity = \frac{2}{S(S-1)} \times \sum_{i=1}^{S-1} \sum_{j=i+1}^{S} \sqrt{\sum_{k=1}^{n} (p_{ik} - p_{jk})^2} \tag{10}$$

where S denotes the total number of orcas, n represents the dimension of the solution space, and p_{ik} and p_{jk} are k–th decision variables of solutions p_i and p_i, respectively. Values close to 0 note a good performance. Finally, the reduction metric is calculated by determining the percentage of columns not accessed by the most proficient orca in relation to the total number of available columns. Again, values close to 1 indicates a good performances.

We can see that the reward function is tailored to encourage solutions that not only enhance performance in terms of F_1 score an accuracy but also maintain diversity and utilize a reduced number of features. Next, updating of states and actions is conducted through DQL process (Line 12). In DQL,

 doi:10.20944/preprints202403.1533.v1

the state is composed of the current performance metrics and the diversity of the orca swarm, and is represented as follows:

$$state = [F_1 \text{ score, accuracy, diversity, reduction}] \tag{11}$$

In the training phase of DQL, the formulation of states is a critical aspect, as it provides the necessary context for decision–making [81]. This state design (Equation (11)) ensures that all critical dimensions of the problem are represented, allowing DQL to make informed adjustments in OPA's operational parameters. This information is compiled in real–time and feeds into DQL to facilitate continuous and adaptive adjustment decisions.

The actions in DQL are decisions on how to adjust OPA's operational parameters, identified as a, b, d, and e. These parameters are vital in defining the movement and adaptation strategies of the orcas in OPA. Specifically, a and b control the intensity and direction of the orcas' movement, influencing their capacity to explore the solution space. On the other hand, d and e determine the orcas' sensitivity to environmental signals, affecting their ability to adapt and converge towards optimal solutions.

In DQL, two key neural networks are implemented: 1) the evaluation network, which is constantly updated with new experiences, and 2) the target network, which provides a stable estimation of Q–function values and is periodically updated to stabilize the learning process [82]. This approach helps mitigate issues associated with data correlation and rapid changes in Q–estimation, which can lead to unstable learning. The loss function in DQL is critical for calibrating the evaluation network. Huber loss, a combination of mean squared and absolute errors, is employed as it is less sensitive to outliers in predictions and provides more stable training. The loss is calculated by comparing the predictions of the evaluation network with the target values generated by the target network. Here, the gradient descent algorithm plays a crucial role, as it adjusts the network parameters by minimizing this loss function, allowing the model to learn from its errors effectively.

During DQL training, each accumulated experience, based on the performance and diversity of the orcas' solutions, contributes to updating the parameters to be improved. The aim is to achieve an effective balance between exploring new areas and exploiting the most promising ones in the solution space. The parameter tuning is carried out through an iterative reinforcement learning process, where the DQL model evaluates the impact of different configurations of these parameters on the overall system performance. Through the reward function, DQL identifies which adjustments in these parameters enhance the efficacy of the search, thus guiding the evolution of the exploration and exploitation strategy in OPA.

The value of OPA's parameters conducted through the DQL training function, employing a neural network for Q function value estimation and a memory buffer for storing experiences (Lines 18–19). Action selection follows an epsilon–greedy strategy, ensuring a proper mix between exploration and exploitation. After executing an action, the new state and reward are observed (Lines 21–22) and stored in the memory buffer. Once sufficient experiences are accumulated, a mini–batch is extracted to train the evaluation network. This training process involves updating the target network with state–action pairs and their corresponding target values, simultaneously adjusting the values of parameters (Lines 26–27).

This iterative DQL training method is reintegrated into the main algorithm, allowing optimization cycles to continue. With each iteration, DQL adapts and improves its policy, thus refining OPA's search for focused exploitation in a promising area of the solution space. This leads to more precise and efficient feature selection for machine learning models (Line 16). This iterative procedure is repeated, progressively refining the solutions proposed by the orcas through a reward that reflects both the quality of the solution in terms of model performance and the exploitation of new potential solutions. With each iteration, DQL adapts, improving its policy and refining OPA's search for more effective exploitation of the solution space, leading to a more precise and efficient feature selection for machine learning models.

 doi:10.20944/preprints202403.1533.v1

Finally, regarding computational complexity, our metaheuristic exhibits $O(kn)$ complexity, where n denotes the dimension of the problem and k represents either the number of iterations or the population size, encapsulating the total number of function evaluations throughout the algorithm's execution. We also evaluate the complexity of the DQL algorithm, which is generally $O(MN)$ [83,84], with M indicating the sample size and N the number of network parameters, both essential for thorough dataset analysis. Despite the potentially large values of M and N, they remain fixed, making the integration of DQL a justifiable increase in computational demand for superior outcomes. Additionally, the continual advancement in computing technology significantly mitigates the impact of this heightened complexity.

5. Experimental Setup

We used a rigorous quantitative methodology to evaluate our proposal, comparing the hybrid approach with the traditional version of the optimization algorithm. Moreover, we tested each machine–learning method independently. This involved detailed planning, execution, and assessment, including statistical testing and in–depth analysis.

5.1. Methodology

To evaluate the enhanced metaheuristic's performance rigorously, we have established a comprehensive quantitative methodology that aligns with the principles outlined in [85]. Our approach involves a comparative analysis between the solutions provided by our proposed hybridization, the native version of the optimization algorithm, and the results generated by machine learning working alone. For that, we employed the following methodological strategy:

- Preparation and planning: Define specific multi–objective goals for feature selection effectiveness, aiming to minimize the number of selected features while simultaneously maximizing accuracy and the F_1 score. Design experiments to systematically evaluate the enhanced technique under controlled conditions, ensuring a balanced optimization of these criteria.
- Execution and assessment: Perform a multi–faceted evaluation of the technique, assessing not only the quality of the solutions generated but also the computational efficiency and convergence properties. Employ rigorous statistical tests to compare the performance with baseline methods. Here, to evaluate data independence and statistical significance, we use the Kolmogorov–Smirnov–Lilliefors test for assessing sample autonomy and the Mann–Whitney–Wilcoxon test for comparative analysis. This approach involves calculating the fitness from each one executions per instance.
- Analysis and validation: Conduct thorough in–depth analysis to understand the Deep Q–Learning's parameter influence and the orca predator algorithm's behavior on the feature selection task. This involves iterating over a range of hyperparameters to fine–tune the model, using the dataset to validate the consistency and stability of the selected features. To ensure the validity of the results generated by our proposal, we conducted tests to evaluate the final outcomes. We can assure that all simulated experiments were carried out with reliability.

5.2. Dataset

Our research utilized the "ECG Heartbeat Categorization Dataset" from Kaggle [15], which comprises records of ECG signals indicative of various heart conditions. These signals are classified into five categories: Normal (N), Supraventricular (S), Ventricular (V), Fusion (F), and Unclassified (Q). The dataset is rich with 188 distinct features per record, encapsulating the complexity of our problem space, represented as n. The dataset is divided into two primary files: the training set, containing 87,554 records, and the test set, with 21,892 records. For our analysis, we further split the test set equally into two parts: one half (10,946 records) is utilized for cross–validation to fine–tune the model parameters, while the remaining half is reserved for the final testing phase to evaluate the model's performance. This significantly reduces the risk of overfitting.

 doi:10.20944/preprints202403.1533.v1

We employed a strategic sampling approach to address class imbalance during training. For classes 1 (S), 2 (V), and 4 (Q), which had a larger number of records, we randomly selected a subset of 1000 records each. This was done to ensure that these classes did not overwhelm the learning process. In contrast, for class 3 (F), which had fewer instances, we included all 641 available records. This selective sampling was crucial to maintain a balanced representation across all classes, enhancing the classifier's ability to identify each category accurately without bias.

During the testing or prediction phase, we utilized the available records. This comprehensive approach in the testing phase allowed us to assess the model's performance across a diverse and complete range of data. In both the training and testing stages, we applied all the 188 features in the dataset to the machine learning techniques, ensuring a thorough analysis and utilization of the available data. This exhaustive feature application was key to comprehensively training and evaluating the machine learning models, providing us with a detailed understanding of the dataset and the effectiveness of our approach.

5.3. Implementation Aspects

We conducted empirical assessments of the Orca Predatory Algorithm. These trials involved independent executions of four distinct machine learning models: Decision Tree (DT), Random Forest (RF), Support Vector Machine (SVM), and Extremely Randomized Trees (ERT). In subsequent stages, we evaluated the hybridization of the OPA algorithm with each of these classification techniques.

In [57], the parameter values yielding the best average performance regarding swarm behavior are described. Based on this, we developed a set of preliminaries evaluations with a reduced test scenario in order to find the best initial configuration. We found setting q to 0.9 notably boosts OPA performance because the algorithm has more opportunities to search solutions in the exploration phase than a local search procedure. These tests generated good results and we can observe that a, and d parameters take values closer to 1 than 0. The b value is kept random without unpredictable behavior. Finally, these some experiments show the F value is always close to 2. Consequently, our proposal will be challenged to operate and improve in an initially adverse situation. We extracted results from 30 runs per algorithm, each running iterating 100 times and $S = 10$ orcas. In order to identify how different values for the parameters affect the final performance, we plotted its values against the performance metrics, enabling us to analyze trends and optimal values that enhance the model's effectiveness. This analysis will revealed which parameters have a critical behaviour for the algorithm performance.

We established the following parameters for the Deep Q–Learning configuration: The state size was set at 5, with an action size of 40 to encompass a broad range of potential parameter adjustments in OPA. We constructed a sequential neural network with dense layers and Dropout for regularization. The network consists of layers with 24 neurons each, using ReLU activation, and a final layer of size equal to the number of possible actions (40), with linear activation. The Huber loss function and the RMSprop optimizer with a learning rate 0.001 were used. Regarding the Epsilon–Greedy policy, we started with an epsilon value of 1.0, decaying to a minimum of 0.01 to balance exploration and exploitation. Using a Double Q–Learning approach, the network trains with random experiences from a mini–batch during training and updating. The target network is updated every 50 training steps to stabilize the estimation of Q–values. The initial values for parameters a, b, d, and e, used in moving the orcas and updating their positions and velocities, are randomly generated in the range of [0, 1] at the start of each OPA algorithm execution.

Another key aspect is the adaptation of OPA to the binary domain, which necessitates the inclusion of a binarization phase following alterations to the solution vector. In this instance, the transformation function utilized was the conventional Sigmoid function. This entails that if $[1/(1+e^{-x})] > \phi$, with ϕ representing a random value uniformly distributed within the interval $[0, 1]$. Then, discretization is achieved by setting $x \leftarrow 1$. Conversely, should this condition not hold, $x \leftarrow 0$ is applied.

 doi:10.20944/preprints202403.1533.v1

Finally, we used Python 3.10 to code all algorithms. The computer used for each test had the following specifications: macOS 14.2.1 Darwin Kernel version 23 with an Ultra M2 chip and 64 GB of RAM. All codes are available in [86].

6. Results and Discussion

The initial comparative analysis conducted between our approach and the traditional version of OPA focuses on the runtime. On one hand, we consider the time it takes to generate the solution vector, encompassing all necessary actions, as well as the time required to process and evaluate fitness accuracy. On the other hand, the time spent on the classification phase using learning techniques is not taken into account because this phase is considered part of the model's training process, where the primary focus is on achieving high accuracy and reliability rather than speed.

OPA achieves the best runtime in all the tests performed, with a value of 200 ms. In contrast, the best time achieved by OPADQL is 500 ms, which is even worse than the average time of OPA, equivalent to 220 ms. There are no instances where OPADQL's runtime is inferior to OPA's runtimes. However, it is important to note that although OPADQL does not outperform OPA in terms of runtime, the difference is negligible. The anticipated benefit in terms of prediction accuracy and feature reduction justifies the additional execution time required by OPADQL.

Furthermore, we acknowledge the importance of examining how the selected features vary across different runs. To this end, we conducted several runs of the OPA and OPADQL methods, analyzing the consistency in feature selection. We found that, although there is inherent variability due to the stochastic nature of the algorithms, certain features tend to be selected more frequently, indicating their potential relevance to the problem at hand. We can see that discarded features are usually related to constant or poorly varied values, since they do not provide useful information for differentiating between heartbeat categories, as well as those that offer redundant information.

Continuing with experimental results, Tables 1 and 2 present our computational findings. The table is structured into five sections, each comprising six rows corresponding to evaluation metrics. We define the best value as the optimal performance attained across each model and feature selection technique combination. Conversely, the worst value signifies the least effective yield. The mean value provides an average of the results, while the standard deviation (std) value quantifies the variability in findings. Additionally, the median value and interquartile range (iqr) value represent the middle value and the spread of the middle 50% of performances. Regarding columnar representation, n/o denotes the absence of an optimizer method, this is, a random search for the solution vector.

 doi:10.20944/preprints202403.1533.v1

Table 1. Comparative analysis of algorithms' performance: OPA vs. OPADQL for DT and RF.

	Metrics	DT			RF		
		n/o	**OPA**	**OPADQL**	**n/o**	**OPA**	**OPADQL**
max **F_1 score**	*best*	0.7911	0.8242	**0.8250**	0.9154	0.9165	**0.9186**
	worst	0.7753	0.7601	**0.7936**	0.8671	0.8921	**0.9087**
	mean	0.7829	0.7845	**0.8093**	0.8956	0.9110	**0.9124**
	std	0.0037	0.0062	**0.0133**	0.0016	0.0045	**0.0137**
	median	0.7835	0.7851	**0.8087**	0.9002	0.9107	**0.9120**
	iqr	0.0045	0.0070	**0.0209**	0.0023	0.0062	**0.0194**
max **Accuracy**	*best*	0.7408	0.7886	**0.9035**	0.8978	0.8989	**0.9035**
	worst	0.7207	0.7225	**0.7432**	0.8897	0.8342	**0.8673**
	mean	0.7299	0.7307	**0.8921**	0.8731	0.8921	**0.8943**
	std	0.0047	0.0071	**0.0159**	0.0021	0.0071	**0.0176**
	median	0.7304	0.7328	**0.8922**	0.8782	0.8922	**0.8940**
	iqr	0.0065	0.0091	**0.0240**	0.0035	0.0108	**0.0257**
min **Diversity**	*best*	n/a	**0.9647**	0.9691	n/a	0.9666	**0.9692**
	worst	n/a	0.9576	**0.9116**	n/a	0.9420	**0.9405**
	mean	n/a	**0.9611**	0.9659	n/a	0.9647	**0.9616**
	std	n/a	**0.0096**	0.0019	n/a	0.0096	**0.0053**
	median	n/a	**0.9636**	0.9663	n/a	0.9663	**0.9637**
	iqr	n/a	0.0003	0.0003	n/a	0.0014	**0.0003**
max **Reduction**	*best*	n/a	0.6043	**0.9645**	n/a	0.6043	**0.8014**
	worst	n/a	0.5026	**0.6099**	n/a	0.5133	**0.5390**
	mean	n/a	0.5378	**0.6811**	n/a	0.5575	**0.6622**
	std	n/a	0.0234	**0.7328**	n/a	0.0259	**0.8782**
	median	n/a	0.5957	**0.6738**	n/a	0.5615	**0.6773**
	iqr	n/a	0.0321	**0.0390**	n/a	0.0401	**0.1170**
max **FEI**	*best*	n/a	0.7800	**0.8991**	n/a	0.8432	**0.9259**
	worst	n/a	0.7646	**0.7992**	n/a	0.8154	**0.8615**
	mean	n/a	0.7710	**0.8293**	n/a	0.8300	**0.8881**
	std	n/a	0.0042	**0.0117**	n/a	0.0059	**0.0160**
	median	n/a	0.0770	**0.8276**	n/a	0.8311	**0.8871**
	iqr	n/a	0.0065	**0.0178**	n/a	0.0082	**0.0265**

OPA stands for a bio–inspired optimizer lacking a learning component, and OPADQL represents our enhanced version of OPA. Finally, we employ F_1 score, accuracy, diversity, reduction, and FEI for metrics. All of them were defined in the previous section. In the context of the F_1 score and accuracy, the OPADQL model exhibits a remarkable and consistent enhancement in performance when contrasted with the native OPA and non–optimized learning methods.

This improvement is particularly pronounced in models employing RF and SVM, underscoring the advanced capability of OPADQL in managing intricate feature selection challenges. This integration enables a more nuanced and precise feature selection process. Consequently, this leads to a significant boost in model performance, as evidenced by the higher F1 scores and accuracy rates, which are critical indicators of a model's predictive power and reliability in classification tasks.

Concerning the diversity metric, an essential measure of the variation and uniqueness in the solutions generated, both OPA and OPADQL showcase commendable performances. However, OPADQL distinguishes itself by achieving marginally inferior values. This slight but critical advantage highlights the efficacy of the hybrid model in sustaining the comprehensive exploitation of the solution space.

 doi:10.20944/preprints202403.1533.v1

Table 2. Comparative analysis of algorithms' performance: OPA vs. OPADQL for SVM and ERT.

	Metrics	SVM			ERT		
		n/o	**OPA**	**OPADQL**	**n/o**	**OPA**	**OPADQL**
max **F_1 score**	*best*	0.8841	0.8843	**0.9155**	0.9168	0.9180	**0.9228**
	worst	0.8841	0.8644	**0.8645**	0.9108	0.8392	**0.9048**
	mean	0.8841	0.8766	**0.8919**	0.9145	0.8946	**0.9153**
	std	0.0000	0.0051	**0.0129**	0.0014	0.0048	**0.0177**
	median	0.8841	0.8771	**0.8954**	0.9147	0.8974	**0.9155**
	iqr	0.0000	0.0082	**0.0175**	0.0021	0.0066	**0.0234**
max **Accuracy**	*best*	0.8565	0.8594	**0.8966**	0.8975	0.9013	**0.9078**
	worst	0.8565	0.8328	**0.8344**	0.8910	0.8023	**0.8826**
	mean	0.8565	0.8488	**0.8682**	0.8951	0.8710	**0.8973**
	std	0.0000	0.0069	**0.0158**	0.0016	0.0066	**0.0218**
	median	0.8565	0.8493	**0.8713**	0.8953	0.8755	**0.8976**
	iqr	0.0000	0.0106	**0.0215**	0.0026	0.0091	**0.0276**
min **Diversity**	*best*	n/a	0.9671	**0.9638**	n/a	**0.9637**	0.9661
	worst	n/a	0.9525	**0.9306**	n/a	0.9326	**0.9557**
	mean	n/a	0.9650	**0.9593**	n/a	**0.9629**	0.9642
	std	n/a	0.0053	**0.0088**	n/a	**0.0015**	0.0061
	median	n/a	0.9658	**0.9631**	n/a	**0.9634**	0.9659
	iqr	n/a	0.0003	**0.0041**	n/a	0.0010	0.0010
max **Reduction**	*best*	n/a	0.5829	**0.8014**	n/a	0.6043	**0.8014**
	worst	n/a	0.5133	**0.5673**	n/a	0.5133	**0.6312**
	mean	n/a	0.5519	**0.7026**	n/a	0.5565	**0.7272**
	std	n/a	0.0190	**0.8713**	n/a	0.0214	**0.8755**
	median	n/a	0.5508	**0.7270**	n/a	0.5615	**0.7234**
	iqr	n/a	0.0254	**0.0975**	n/a	0.0254	**0.0408**
max **FEI**	*best*	n/a	0.8220	**0.9342**	n/a	0.8434	**0.9265**
	worst	n/a	0.8014	**0.8650**	n/a	0.8237	**0.8798**
	mean	n/a	0.8105	**0.8955**	n/a	0.8333	**0.9039**
	std	n/a	0.0048	**0.0168**	n/a	0.0048	**0.0110**
	median	n/a	0.8100	**0.8981**	n/a	0.8333	**0.9043**
	iqr	n/a	0.0046	**0.0223**	n/a	0.0059	**0.0139**

The OPADQL's ability to try to maintain a diverse set of solutions while simultaneously homing in on the most effective ones exemplifies its balanced approach to optimization. This diverse exploitation ensures that the algorithm does not prematurely converge to suboptimal solutions, thereby enhancing the robustness and reliability of the outcomes.

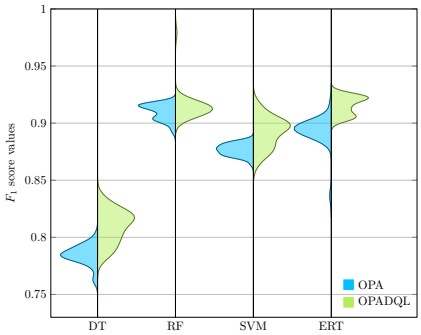
The results of the reduction metric are particularly striking. In this aspect, OPADQL outshines OPA by a significant margin, indicating a more proficient approach in maximizing the number of features while not preserving or even enhancing model accuracy. This attribute of OPADQL is especially beneficial in dealing with high–dimensional data, where reducing the feature set without compromising the model's effectiveness is a challenging but crucial task. The ability to selectively reduce features contributes to more streamlined models, reducing computational complexity and enhancing efficiency.

Finally, when considering the Feature Efficiency Index, which amalgamates all the above–discussed metrics, the superiority of OPADQL becomes even more evident. The FEI values for OPADQL consistently surpass those of the native OPA and non–optimized methodologies across all tested machine–learning techniques. This comprehensive index affirms the overall effectiveness of OPADQL in feature selection. By excelling in multiple metrics —including accuracy, diversity, reduction, and the FEI— OPADQL proves itself as a robust, versatile, and effective tool for feature selection. Its ability to deliver high–quality results across different evaluation criteria and machine learning models signifies

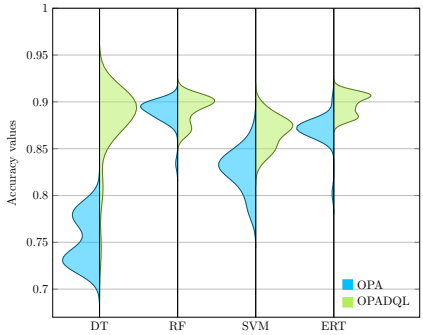
 doi:10.20944/preprints202403.1533.v1

a significant advancement in the field, offering a sophisticated solution for complex feature selection challenges.

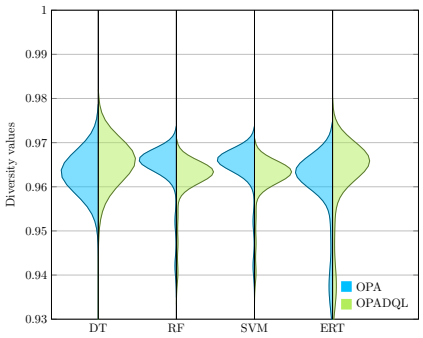
Charts deployed on Figure 2 enrich our comprehension of the effectiveness of OPA (left side) and OPADQL (right side) across different ML models.



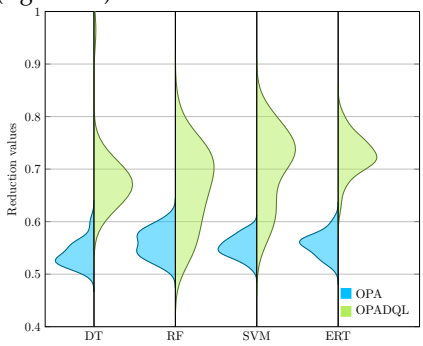
(a) Distribution of F_1 score values obtained by OPA (left side) and OPADQL (right side).



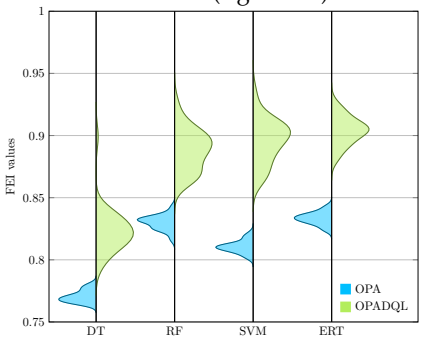
(b) Distribution of accuracy values obtained by OPA (left side) and OPADQL (right side).



(c) Distribution of diversity values obtained by OPA (left side) and OPADQL (right side).



(d) Distribution of reduction values obtained by OPA (left side) and OPADQL (right side).



(e) Distribution of FEI values obtained by OPA (left side) and OPADQL (right side).

Figure 2. Distributions for all metrics employed to analyze the effectiveness of OPA and OPADQL on machine learning methods.

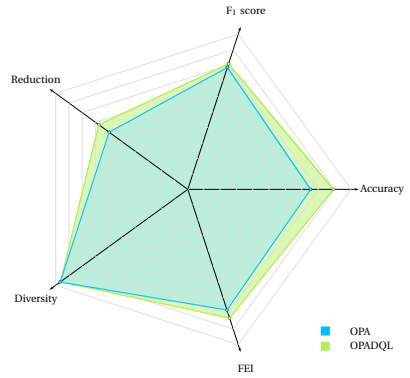
These graphical illustrations reveal the data's distribution and density, highlighting OPADQL's supremacy in several dimensions. Specifically, Figures 2a and 2b, showcasing the F_1 score and accuracy of OPADQL, respectively, demonstrate its notable superiority, indicating more accurate and dependable classification performance.

Regarding diversity, OPADQL exhibits an outstanding yield by achieving slightly lower values than its native version (see Figure 2c). This performance is due to the intrinsic nature of the native

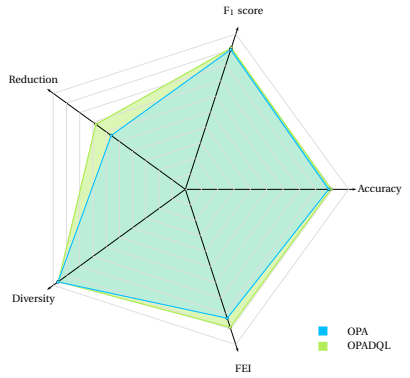
 doi:10.20944/preprints202403.1533.v1

algorithm that attempts to explore promising areas. Regarding feature reduction, OPADQL is noted for its capability to efficiently reduce the number of features without sacrificing model precision (refer to Figure 2d). Moreover, in the context of the Feature Efficiency Index (FEI) (refer to Figure 2e), OPADQL showcases a comprehensive advantage, underscoring its efficiency in feature selection across diverse ML models. This evidence underlines OPADQL's robustness and adaptability, positioning it as a crucial tool for overcoming feature selection challenges.

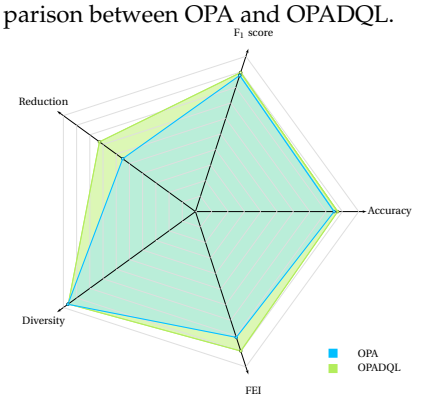
Additionally, we employed two–dimensional charts to represent the multivariate variables influencing the performance of these algorithms (see Figure 3).



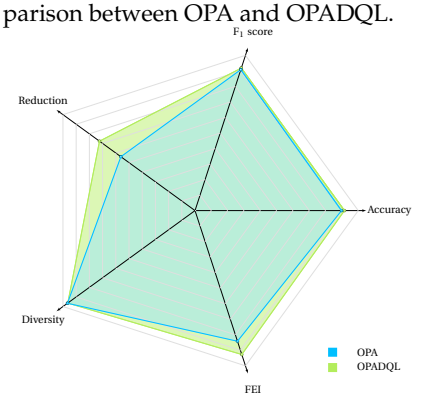
(a) DT radar analysis: Performance comparison between OPA and OPADQL.



(b) RF radar analysis: Performance comparison between OPA and OPADQL.



(c) SVM radar analysis: Performance comparison between OPA and OPADQL.



(d) ERT radar analysis: Performance comparison between OPA and OPADQL.

Figure 3. Radar analysis comparing OPA and OPADQL performance across all machine learning techniques.

In general terms, OPADQL exhibits superiority over OPA in various ML models. For example, Figure 3a, which shows the DT learning model, reveals OPADQL's distinct advantage across all performance metrics, accentuating its selection efficacy. Similarly, Figure 3b, related to the RF technique, highlights OPADQL's exceptional outcomes, particularly in F_1 score and accuracy, with a noted reduction in diversity, suggesting a more targeted feature selection.

In the SVM context (see Figure 3c), OPADQL's performance in F_1 score and accuracy further proves its effective classification capability, alongside a remarked decrease in solution diversity. Lastly, Figure 3d, corresponding to the ERT model, demonstrates OPADQL's superiority in all evaluated metrics, showcasing its ability to balance feature selection efficiency with precision and diversity maintenance. These findings affirm OPADQL's ability to navigate the complexities of various ML models, optimizing feature selection to enhance model performance while ensuring the balance between diversity and accuracy.

 doi:10.20944/preprints202403.1533.v1

6.1. Statistical test

To showcase that our proposal overcomes the native version, we employ a robust statistical strategy exploiting generated results. This strategy includes two contrasted hypotheses: a) evaluation of normality and b) hypothesis testing to ascertain whether the samples originate from a uniformly distributed sequence. To assess whether the observations (runs per instance) form a Gaussian distribution, we define H_0 to propose that samples do not adhere to a normal distribution, with H_1 positing the contrary. For that, Kolmogorov–Smirnov–Lilliefors test was utilized. The significance level for the p–value was set at 0.05, indicating that results falling below this threshold would suggest that the test is significant. Consequently, this leads to a failure to reject the null hypothesis H_0.

The findings revealed that the samples do not conform to a normal distribution, allowing the non–parametric Mann–Whitney–Wilcoxon test. In this context, H_0 suggests that there is a difference between the distributions of the two samples $P(\text{samples}_{OPA} < \text{samples}_{OPADQL})$. H_1 states the opposite. The following table shows the Mann–Whitney–Wilcoxon test results. Acronym *sws* indicates statistically without significance.

Table 3 shows that for F_1 score and accuracy metrics, the results are highly significant across all models, with p–values far below the 0.05 threshold, illustrating OPADQL's superior performance. In terms of diversity, significant differences are observed in the RF and SVM models, as indicated by their p–values, whereas DT and ERT models show statistically insignificant differences. For reduction and FEI metrics, the p–values again highlight OPADQL's enhanced efficiency across most models, demonstrating its effectiveness in feature selection and overall model performance optimization.

Table 3. p–values obtained from Wilcoxon–Mann–Whitney Test.

Metrics	OPADQL v/s OPA			
	DT	RF	SVM	ERT
F_1 score	$\mathbf{1.6564 \times 10^{-10}}$	$\mathbf{3.2788 \times 10^{-5}}$	$\mathbf{3.2529 \times 10^{-6}}$	$\mathbf{6.2779 \times 10^{-9}}$
Accuracy	$\mathbf{1.1289 \times 10^{-10}}$	$\mathbf{9.7638 \times 10^{-6}}$	$\mathbf{1.9864 \times 10^{-6}}$	$\mathbf{7.1438 \times 10^{-9}}$
Diversity	sws	$\mathbf{1.4360 \times 10^{-11}}$	$\mathbf{1.4360 \times 10^{-11}}$	sws
Reduction	$\mathbf{1.3330 \times 10^{-11}}$	$\mathbf{9.7638 \times 10^{-6}}$	$\mathbf{3.3184 \times 10^{-11}}$	$\mathbf{1.3483 \times 10^{-11}}$
FEI	$\mathbf{1.5876 \times 10^{-11}}$	$\mathbf{2.9094 \times 10^{-7}}$	$\mathbf{1.4360 \times 10^{-11}}$	$\mathbf{1.4360 \times 10^{-11}}$

6.2. Comparing OPADQL vs state–of–the–art algorithms

Tables 4, 5, 6, and 7 present a comparative analysis between OPADQL and various state-of-the-art algorithms, including Genetic Algorithm (GA), Particle Swarm Optimization (PSO), Bat Algorithm (BAT), Black Hole Optimization (BH), Grey Wolf Optimizer (GWO), and a random strategy. This analysis is conducted using machine learning models and key metrics used in Tables 1 and 2. Additionally, we include a random strategy for the construction of the solution vector for feature selection; however, this strategy is not applicable to the diversity metric due to the generation of a unique random solution.

 doi:10.20944/preprints202403.1533.v1

Table 4. Comparative analysis of the performance: OPADQL using DT vs. state–of–the-art algorithms.

	Metrics	DT						
		Random	**GA**	**PSO**	**BAT**	**BH**	**GWO**	**OPADQL**
max **F$_1$ score**	*best*	0.8191	0.8085	0.8106	0.8126	0.8147	0.8168	**0.8250**
	worst	0.7781	0.7777	0.7797	0.7817	0.7837	0.7857	**0.7936**
	mean	0.7929	0.7931	0.7951	0.7972	0.7992	0.8012	**0.8093**
	std	0.0011	0.0049	0.0069	0.0049	0.0059	0.0109	**0.0133**
	median	0.7935	0.7931	0.7951	0.7971	0.7991	0.8011	**0.8087**
	iqr	0.0011	0.0207	0.0177	0.0027	0.0201	0.0200	**0.0209**
max **Accuracy**	*best*	0.7720	0.8854	0.8877	0.8899	0.8922	0.8945	**0.9035**
	worst	0.7107	0.7283	0.7302	0.7321	0.7339	0.7358	**0.7432**
	mean	0.7499	0.8743	0.8765	0.8787	0.8809	0.8832	**0.8921**
	std	0.0117	0.0049	0.0049	0.0049	0.0049	0.0049	**0.0159**
	median	0.7463	0.8741	0.8761	0.8791	0.8811	0.8830	**0.8922**
	iqr	0.0115	0.0215	0.0211	0.0225	0.0235	0.0217	**0.0240**
max **Diversity**	*best*	n/a	**0.9645**	0.9650	0.9701	0.9748	0.9698	0.9691
	worst	n/a	0.9574	0.9575	0.9576	0.9577	0.9578	**0.9116**
	mean	n/a	**0.9600**	0.9611	0.9612	0.9613	0.9614	0.9659
	std	n/a	0.0048	0.0049	0.0052	0.0053	0.0052	**0.0019**
	median	n/a	**0.9590**	0.9600	0.9615	0.9611	0.9612	0.9663
	iqr	n/a	0.0048	0.0049	0.0050	0.0051	0.0052	**0.0003**
max **Reduction**	*best*	0.5634	0.6041	0.6042	0.6043	0.6044	0.7045	**0.9645**
	worst	0.4564	0.5024	0.5025	0.5026	0.5027	0.5928	**0.6099**
	mean	0.4873	0.5376	0.5377	0.5378	0.5379	0.6380	**0.6811**
	std	0.0375	0.0276	0.0278	0.0277	0.0279	0.02790	**0.7328**
	median	0.4863	0.5370	0.5371	0.5371	0.5373	0.6373	**0.6738**
	iqr	0.0144	0.0276	0.0277	0.0278	0.0279	0.0280	**0.0390**
max **FEI**	*best*	0.7200	0.8570	0.7756	0.8100	0.7890	0.8700	**0.8991**
	worst	0.6320	0.7900	0.7100	0.7110	0.6915	0.7950	**0.7992**
	mean	0.6767	0.8200	0.7400	0.7710	0.7215	0.8250	**0.8293**
	std	0.0204	0.0100	0.0102	0.0104	0.0106	0.0110	**0.0117**
	median	0.6747	0.8230	0.7354	0.7340	0.7545	0.8270	**0.8276**
	iqr	0.0276	0.0160	0.0162	0.0164	0.0166	0.0170	**0.0178**

In relation to DT, OPADQL surpasses all other algorithms in F_1 score, with a notable difference compared to the second–best result (GWO). This trend remains consistent across other metrics such as accuracy and FEI, where OPADQL also achieves the best results. In RF, OPADQL continues to be the most effective algorithm, reaching the highest value in accuracy and F_1 score, surpassing the results of other state–of–the–art algorithms. Additionally, OPADQL exhibits an exceptional ability to reduce the number of features without compromising model accuracy, as reflected in superior reduction and FEI values compared with other algorithms. In SVM, OPADQL performs outstandingly, surpassing other algorithms in several key metrics. Although GWO presents a slightly higher F_1 score, OPADQL stands out in the accuracy metric and in FEI, highlighting its ability to identify feature sets that are not only predictively powerful but also efficient in terms of the number of features used. Finally, in ERT, OPADQL continues to demonstrate its superiority, achieving the highest values in both accuracy and F_1 score, compared to state-of-the-art algorithms and the random strategy. This efficacy is particularly notable in the reduction metric, where OPADQL equals or surpasses other methods, evidencing its exceptional skill in maintaining a simplified model without compromising performance.

 doi:10.20944/preprints202403.1533.v1

Table 5. Comparative analysis of the performance: OPADQL using RF vs. state–of–the-art algorithms.

	Metrics	RF						
		Random	**GA**	**PSO**	**BAT**	**BH**	**GWO**	**OPADQL**
***max* F_1 score**	*best*	0.8915	0.9002	0.9025	0.9048	0.9071	0.9094	**0.9186**
	worst	0.8671	0.8905	0.8928	0.8951	0.8973	0.8996	**0.9087**
	mean	0.8956	0.8942	0.8964	0.8987	0.901	0.9033	**0.9124**
	std	0.0016	0.0213	0.0213	0.0213	0.0213	0.0213	**0.0137**
	median	0.8802	0.8940	0.8960	0.8990	0.9010	0.9030	**0.9120**
	iqr	0.0023	0.0173	0.0183	0.0131	0.0013	0.0113	**0.0194**
***max* Accuracy**	*best*	0.9001	0.8854	0.8877	0.8899	0.8922	0.8945	**0.9035**
	worst	0.8897	0.8556	0.8521	0.8543	0.8565	0.8586	**0.8673**
	mean	0.8731	0.8764	0.8786	0.8809	0.8831	0.8854	**0.8943**
	std	0.0021	0.0213	0.0213	0.0213	0.0213	0.0213	**0.0176**
	median	0.8782	0.876	0.879	0.881	0.883	0.885	**0.8940**
	iqr	0.0015	0.0213	0.0225	0.0215	0.0213	0.0223	**0.0257**
***max* Diversity**	*best*	n/a	0.9725	0.9701	0.9766	0.9699	0.9749	**0.9692**
	worst	n/a	0.9574	0.9575	0.9576	0.9577	0.9578	**0.9405**
	mean	n/a	0.9710	0.9654	0.9681	0.9613	0.974	**0.9616**
	std	n/a	0.0048	0.0049	0.0050	0.0051	0.0052	**0.0053**
	median	n/a	0.9711	0.961	0.961	0.961	0.961	**0.9637**
	iqr	n/a	0.0048	0.0059	0.0050	0.0051	0.0052	**0.0003**
***max* Reduction**	*best*	0.5469	0.6041	0.6242	0.6143	0.6344	0.7045	**0.8014**
	worst	0.4475	0.5024	0.5025	0.5026	0.5027	0.5028	**0.5390**
	mean	0.4948	0.5376	0.5377	0.5578	0.5479	0.6380	**0.6622**
	std	0.0269	0.0276	0.0277	0.0278	0.0289	0.0280	**0.8782**
	median	0.4917	0.5378	0.5485	0.5599	0.5364	0.6370	**0.6773**
	iqr	0.0469	0.0276	0.0277	0.0278	0.0279	0.0280	**0.1170**
***max* FEI**	*best*	0.7823	0.8000	0.78920	0.7821	0.8021	0.8823	**0.9259**
	worst	0.7257	0.6250	0.5355	0.6160	0.7065	0.8300	**0.8615**
	mean	0.7567	0.7010	0.6635	0.7040	0.7445	0.8660	**0.8881**
	std	0.0150	0.0140	0.0142	0.0144	0.0146	0.0150	**0.0160**
	median	0.7564	0.7050	0.6655	0.7060	0.7465	0.8670	**0.8871**
	iqr	0.0257	0.0257	0.0252	0.0254	0.0056	0.0260	**0.0265**

The superiority of OPADQL is significantly attributed to its advanced configuration, which incorporates a higher number of adjustable parameters in comparison to other state-of-the-art algorithms. This additional complexity, far from being an obstacle, becomes a decisive advantage thanks to the integration of DQL. Being a central component of OPADQL, it grants a decisive edge by enabling a dynamic and detailed adaptation of these parameters according to the complexities of the data and the variety of the solution space.

OPADQL's ability to effectively adjust these additional parameters contrasts with the rigidity of traditional state-of-the-art algorithms, which, while efficient within their own frameworks, lack the flexibility to adapt to the complex variations inherent in high-dimensional data. The integration of DQL into OPADQL not only enhances performance through a more accurate and relevant feature selection but also reflects the algorithm's capacity to learn and continuously improve through experience, adjusting its internal parameters to efficiently explore the search space and exploit the most promising regions.

 doi:10.20944/preprints202403.1533.v1

Table 6. Comparative analysis of thw performance: OPADQL using SVM vs. state–of–the-art algorithms.

	Metrics	SVM						
		Random	**GA**	**PSO**	**BAT**	**BH**	**GWO**	**OPADQL**
***max* F_1 score**	*best*	0.8874	0.9052	0.9073	0.9094	0.9115	**0.9170**	0.9155
	worst	0.8485	0.8550	0.8560	0.8570	0.8580	**0.8660**	0.8645
	mean	0.8666	0.8890	0.8895	0.8900	0.8905	**0.8930**	0.8919
	std	0.0087	0.0125	0.0126	0.0127	0.0128	0.0130	**0.0131**
	median	0.8664	0.8890	0.8895	0.8900	0.8905	**0.8978**	0.8954
	iqr	0.0098	0.0160	0.0162	0.0164	0.0166	**0.0178**	0.0175
***max* Accuracy**	*best*	0.8615	0.8860	0.8865	0.8870	0.8875	**0.8994**	0.8966
	worst	0.8144	0.8250	0.8260	0.8270	0.8280	0.8290	**0.8344**
	mean	0.8361	0.8610	0.8615	0.8620	0.8625	**0.8692**	0.8682
	std	0.0113	0.0145	0.0146	0.0147	0.0148	0.0149	**0.0158**
	median	0.8360	0.8610	0.8615	0.8620	0.8625	0.8630	**0.8713**
	iqr	0.0131	0.0200	0.0002	0.0204	0.0006	0.0308	**0.0215**
***max* Diversity**	*best*	n/a	0.9636	0.9637	0.9638	0.9639	0.9640	**0.9638**
	worst	n/a	0.9304	0.9305	0.9306	0.9307	0.9308	**0.9306**
	mean	n/a	0.9591	0.9592	0.9593	0.9594	0.9595	**0.9593**
	std	n/a	0.0086	0.0087	0.0088	0.0089	0.0090	**0.0088**
	median	n/a	0.9591	0.9592	0.9593	0.9594	0.9595	**0.9631**
	iqr	n/a	0.0038	0.0039	0.0040	0.0041	0.0042	**0.0041**
***max* Reduction**	*best*	0.5524	0.6812	0.4813	0.5814	0.6715	0.7916	**0.8014**
	worst	0.4088	0.5571	0.4072	0.5573	0.5574	0.5575	**0.5673**
	mean	0.4987	0.6024	0.4525	0.6926	0.5927	0.6928	**0.7026**
	std	0.0350	0.8608	0.8609	0.8610	0.8611	0.8612	**0.8713**
	median	0.5027	0.6024	0.4625	0.6926	0.5927	0.6928	**0.7270**
	iqr	0.0441	0.0950	0.0435	0.0252	0.0053	0.0954	**0.0975**
***max* FEI**	*best*	0.7671	0.6299	0.6038	0.7240	0.8242	0.9044	**0.9342**
	worst	0.6960	0.5547	0.5248	0.6049	0.7550	0.8051	**0.8650**
	mean	0.7388	0.5842	0.5643	0.6545	0.7746	0.8548	**0.8955**
	std	0.0120	0.0164	0.0165	0.0166	0.0167	0.0168	**0.0168**
	median	0.7341	0.5942	0.5678	0.6445	0.7646	0.8580	**0.8981**
	iqr	0.0223	0.0011	0.0112	0.0213	0.0014	0.0215	**0.0223**

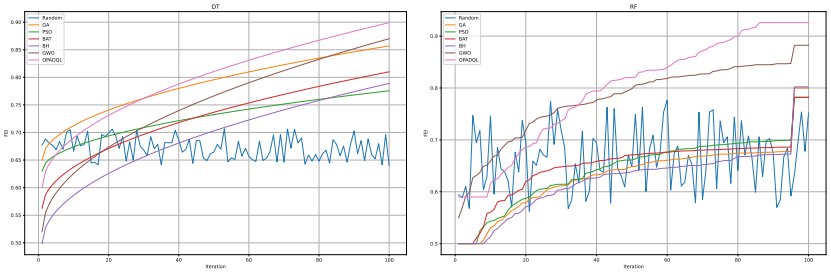
 doi:10.20944/preprints202403.1533.v1

Table 7. Comparative analysis of thw performance: OPADQL using ERT vs. state–of–the-art algorithms.

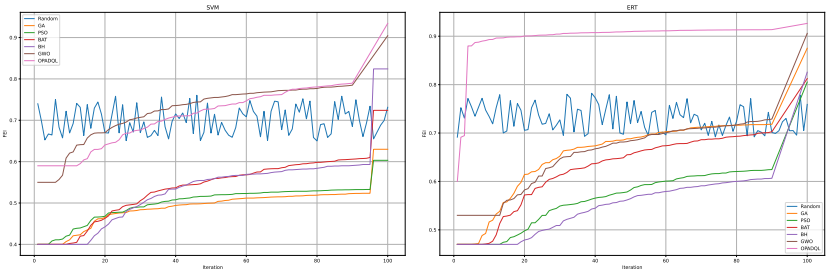
	Metrics	ERT						
		Random	**GA**	**PSO**	**BAT**	**BH**	**GWO**	**OPADQL**
max F_1 **score**	*best*	0.9121	0.9201	0.9202	0.9203	0.9204	**0.9240**	0.9228
	worst	0.8823	0.9020	0.9021	0.9022	0.9023	**0.9060**	0.9048
	mean	0.9023	0.9140	0.9141	0.9142	0.9143	**0.9160**	0.9153
	std	0.0014	0.0172	0.0173	0.0174	0.0175	**0.0179**	0.0177
	median	0.8947	0.9140	0.9141	0.9142	0.9143	**0.9160**	0.9155
	iqr	0.0021	0.0225	0.0226	0.0227	0.0228	**0.0235**	0.0234
max **Accuracy**	*best*	0.9052	0.9050	0.9055	0.9060	0.9065	0.9070	**0.9078**
	worst	0.8620	0.8800	0.8805	0.8810	0.8815	0.8820	**0.8826**
	mean	0.8806	0.8950	0.8955	0.8960	0.8965	0.8970	**0.8973**
	std	0.0016	0.0205	0.0206	0.0207	0.0208	0.0209	**0.0218**
	median	0.8953	0.8950	0.8955	0.8960	0.8965	0.8970	**0.8976**
	iqr	0.0026	0.0260	0.0262	0.0264	0.0266	0.0268	**0.0276**
max **Diversity**	*best*	n/a	0.9659	0.9660	0.9661	0.9662	0.9663	**0.9661**
	worst	n/a	0.9555	0.9556	0.9557	0.9558	0.9559	**0.9557**
	mean	n/a	0.9640	0.9641	0.9642	0.9643	0.9644	**0.9642**
	std	n/a	0.0060	0.0061	0.0062	0.0063	0.0064	**0.0061**
	median	n/a	0.9640	0.9641	0.9642	0.9643	0.9644	**0.9659**
	iqr	n/a	0.0008	0.0009	0.0010	0.0011	0.0012	**0.0010**
max **Reduction**	*best*	0.5414	0.7912	0.7913	0.7914	0.7915	0.7916	**0.8014**
	worst	0.4530	0.6210	0.6211	0.6212	0.6213	0.6214	**0.6312**
	mean	0.4983	0.7170	0.7171	0.7172	0.7173	0.7174	**0.7272**
	std	0.0270	0.8650	0.8651	0.8652	0.8653	0.8654	**0.8755**
	median	0.5000	0.7170	0.7171	0.7172	0.7173	0.7174	**0.7234**
	iqr	0.0359	0.0395	0.0396	0.0397	0.0398	0.0399	**0.0408**
max **FEI**	*best*	0.7894	0.8752	0.8053	0.8125	0.8257	0.9059	**0.9265**
	worst	0.7334	0.8087	0.6888	0.7189	0.7590	0.8091	**0.8798**
	mean	0.7604	0.84027	0.7228	0.7630	0.7731	0.8433	**0.9039**
	std	0.0156	0.0106	0.0107	0.0108	0.0109	0.0110	**0.0110**
	median	0.7691	0.8427	0.7328	0.7630	0.7701	0.8333	**0.9043**
	iqr	0.0020	0.0035	0.0136	0.0107	0.0108	0.0039	**0.0139**

The convergence graphs for DT, RF, SVM, and ERT models presented in Figure 4 consistently demonstrate OPADQL's superior performance in terms of the FEI metric, surpassing state-of-the-art algorithms and the random strategy across iterations. While the other algorithms also show progress, they do not reach the level of efficacy of OPADQL. The random strategy, on the other hand, shows the lowest performance, highlighting the importance of meticulous parameter selection and adjustment. OPADQL not only achieves a higher score in FEI but its curve suggests a faster convergence compared to the other algorithms, which is particularly notable in the SVM and ERT models, where the competition is closer. Through these models, OPADQL demonstrates its ability to efficiently optimize feature selection, adapting to different dynamics and data requirements. The robustness and versatility of OPADQL become evident as it maintains a clear advantage over alternative algorithms in a wide variety of machine learning contexts, demonstrating its potential for practical applications by effectively improving model performance.

 doi:10.20944/preprints202403.1533.v1



(a) DT. Convergence analysis. **(b)** RF. Convergence analysis



(c) SVM. Convergence analysis **(d)** ERT. Convergence analysis

Figure 4. Convergence analysis of OPADQL versus state-of-the-art algorithms across various Machine Learning models.

7. Conclusions

This study introduces a hybrid optimization method that synergizes the biomimetic prowess of the Orca Predator Algorithm with the adaptive intelligence of Deep Q–Learning. This approach was meticulously designed to enhance the process of feature selection in machine learning, enabling the construction of more efficient models by selecting the most relevant subset of features. However, our approach is not solely confined to these. When applying our method to regression tasks, it is critical to make certain adaptations. For instance, instead of relying on the F1 score and accuracy, we recommend employing analogous metrics such as Mean Squared Error, Root Mean Squared Error, and Mean Absolute Error. These metrics are broadly utilized for their effective ability to capture the discrepancy between predicted and actual values, thereby providing a clear measure of prediction accuracy and model performance in continuous output spaces.

To rigorously evaluate the efficacy of our proposed model, we deployed it on the ECG Heartbeat Categorization Dataset, characterized by its high–dimensional feature space of 188 attributes. We further validated our approach across a diverse array of machine learning models, to ensure a comprehensive assessment under varied modeling techniques.

The results obtained from the extensive evaluation clearly demonstrate that the orca predator algorithm hybridized with Deep Q–Learning significantly outperforms the standalone OPA across all tested machine learning models. According to findings, we can see that the proposal seeks to increase the values of a and b while reducing the value of q, signifying a decrease in exploration probability, that is, intensifying the search process. For that, our analysis revealed that OPADQL consistently achieved higher F_1 scores and accuracy, effectively enhancing the predictive performance of Decision Trees, Random Forests, Support Vector Machines, and Extremely Randomized Tree models. Furthermore, OPADQL exhibited a superior ability to reduce the feature space without compromising the model's effectiveness, as evidenced by its higher reduction scores and the Feature Efficiency Index (FEI). These results not only underscore the enhanced optimization capability of OPADQL but also highlight its potential to significantly streamline the feature selection process, making it an invaluable tool for building more robust and efficient machine learning models.

 doi:10.20944/preprints202403.1533.v1

Based on all generated results, we can infer that our solution proposal outperforms traditional a–posteriori techniques random permutation feature importance, thanks to our use of sophisticated metaheuristics for an exhaustive analysis of the feature space. This approach not only pinpoints essential features and their synergies but also significantly enhances model generalization. Although our technique may require more time, it delivers unparalleled accuracy, robustness, and dependability, vital for applications where precise outcomes are paramount.

Finally, some future studies in this domain could focus on exploring the integration of DQL with other bio–inspired optimization algorithms and advanced machine learning techniques, aiming to further enhance its feature selection capabilities. Additionally, adapting OPADQL to address multi–objective optimization problems presents an intriguing avenue for expanding its applicability across various disciplines. Another promising line of inquiry involves investigating the scalability and efficiency of OPADQL in processing extremely large datasets, particularly in fields such as genomics and text mining, where high–dimensional data is prevalent. These future directions not only promise to refine the efficacy and versatility of OPADQL but also open up new possibilities for innovative applications in the ever–evolving landscape of machine learning and data analysis.

Author Contributions: Formal analysis, R.O., R.S., and B.C.; Investigation, R.O., C.R., R.S., and B.C.; Methodology, R.O. and R.S.; Resources, R.O.; Software, C.R. and R.O.; Validation, R.O., R.S., and B.C.; Writing—original draft, C.R. and R.O.; Writing—review and editing, R.O., C.R., R.S., and B.C. All the authors of this paper hold responsibility for every part of this manuscript. All authors have read and agreed to the published version of the manuscript.

Funding: Rodrigo Olivares is supported by grant ANID/FONDECYT/INICIACION/11231016. Broderick Crawford is supported by grant ANID/FONDECYT/REGULAR/1210810.

Institutional Review Board Statement: Not applicable

Data Availability Statement: Data is available on https://doi.org/10.6084/M9.FIGSHARE.25126043.

Conflicts of Interest: The authors declare no conflicts of interest. The founding sponsors had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Gad, A.G. Particle swarm optimization algorithm and its applications: a systematic review. *Archives of computational methods in engineering* **2022**, *29*, 2531–2561.
2. Salhi, S.; Thompson, J. An overview of heuristics and metaheuristics. *The Palgrave Handbook of Operations Research* **2022**, pp. 353–403.
3. Abualigah, L.; Yousri, D.; Abd Elaziz, M.; Ewees, A.A.; Al-Qaness, M.A.; Gandomi, A.H. Aquila optimizer: a novel meta-heuristic optimization algorithm. *Computers & Industrial Engineering* **2021**, *157*, 107250.
4. Prabha, S.; Yadav, R. Differential evolution with biological-based mutation operator. *Engineering Science and Technology, an International Journal* **2020**, *23*, 253–263.
5. Olivares, R.; Soto, R.; Crawford, B.; Riquelme, F.; Munoz, R.; Ríos, V.; Cabrera, R.; Castro, C. Entropy–based diversification approach for bio–computing methods. *Entropy* **2022**, *24*, 1293.
6. Molina, D.; Poyatos, J.; Ser, J.D.; García, S.; Hussain, A.; Herrera, F. Comprehensive taxonomies of nature- and bio-inspired optimization: Inspiration versus algorithmic behavior, critical analysis recommendations. *Cognitive Computation* **2020**, *12*, 897–939.
7. Ahmed, H.R. An efficient fitness-based stagnation detection method for particle swarm optimization. Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation, 2014, pp. 1029–1032.
8. Worasucheep, C. A particle swarm optimization with stagnation detection and dispersion. 2008 IEEE congress on evolutionary computation (IEEE world congress on computational intelligence). IEEE, 2008, pp. 424–429.
9. Zaman, H.R.R.; Gharehchopogh, F.S. An improved particle swarm optimization with backtracking search optimization algorithm for solving continuous optimization problems. *Engineering with Computers* **2022**, *38*, 2797–2831.

10. Dokeroglu, T.; Kucukyilmaz, T.; Talbi, E.G. Hyper-heuristics: A survey and taxonomy. *Computers & Industrial Engineering* **2024**, *187*, 109815. doi:10.1016/j.cie.2023.109815.
11. Chen, X.; Zhang, K.; Ji, Z.; Shen, X.; Liu, P.; Zhang, L.; Wang, J.; Yao, J. Progress and Challenges of Integrated Machine Learning and Traditional Numerical Algorithms: Taking Reservoir Numerical Simulation as an Example. *Mathematics* **2023**, *11*, 4418. doi:10.3390/math11214418.
12. Peres, F.; Castelli, M. Combinatorial Optimization Problems and Metaheuristics: Review, Challenges, Design, and Development. *Applied Sciences* **2021**, *11*, 6449. doi:10.3390/app11146449.
13. Remeseiro, B.; Bolon-Canedo, V. A review of feature selection methods in medical applications. *Computers in biology and medicine* **2019**, *112*, 103375.
14. Colaco, S.; Kumar, S.; Tamang, A.; Biju, V.G. A review on feature selection algorithms. *Emerging Research in Computing, Information, Communication and Applications: ERCICA 2018, Volume 2* **2019**, pp. 133–153.
15. Fazeli, S. ECG heartbeat categorization dataset, 2018.
16. Calvet, L.; de Armas, J.; Masip, D.; Juan, A.A. Learnheuristics: hybridizing metaheuristics with machine learning for optimization with dynamic inputs. *Open Mathematics* **2017**, *15*, 261–280. doi:10.1515/math-2017-0029.
17. Liang, Y.C.; Cuevas Juarez, J.R. A self-adaptive virus optimization algorithm for continuous optimization problems. *Soft Computing* **2020**, *24*, 13147–13166.
18. Olamaei, J.; Moradi, M.; Kaboodi, T. A new adaptive modified firefly algorithm to solve optimal capacitor placement problem. 18th Electric power distribution conference. IEEE, 2013, pp. 1–6.
19. Li, X.; Yin, M. Modified cuckoo search algorithm with self adaptive parameter method. *Information Sciences* **2015**, *298*, 80–97.
20. Li, X.; Yin, M. Self-adaptive constrained artificial bee colony for constrained numerical optimization. *Neural Computing and Applications* **2014**, *24*, 723–734.
21. Cui, L.; Li, G.; Zhu, Z.; Wen, Z.; Lu, N.; Lu, J. A novel differential evolution algorithm with a self-adaptation parameter control method by differential evolution. *Soft Computing* **2018**, *22*, 6171–6190.
22. de Barros, J.B.; Sampaio, R.C.; Llanos, C.H. An adaptive discrete particle swarm optimization for mapping real-time applications onto network-on-a-chip based MPSoCs. Proceedings of the 32nd Symposium on Integrated Circuits and Systems Design, 2019, pp. 1–6.
23. Cruz-Salinas, A.F.; Perdomo, J.G. Self-adaptation of genetic operators through genetic programming techniques. Proceedings of the Genetic and Evolutionary Computation Conference, 2017, pp. 913–920.
24. Kavoosi, M.; Dulebenets, M.A.; Abioye, O.F.; Pasha, J.; Wang, H.; Chi, H. An augmented self-adaptive parameter control in evolutionary computation: A case study for the berth scheduling problem. *Advanced Engineering Informatics* **2019**, *42*, 100972.
25. Nasser, A.B.; Zamli, K.Z. Parameter free flower algorithm based strategy for pairwise testing. Proceedings of the 2018 7th international conference on software and computer applications, 2018, pp. 46–50.
26. Zhang, L.; Chen, H.; Wang, W.; Liu, S. Improved Wolf Pack Algorithm for Solving Traveling Salesman Problem. FSDM, 2018, pp. 131–140.
27. Soto, R.; Crawford, B.; Olivares, R.; Carrasco, C.; Rodriguez-Tello, E.; Castro, C.; Paredes, F.; de la Fuente-Mella, H. A reactive population approach on the dolphin echolocation algorithm for solving cell manufacturing systems. *Mathematics* **2020**, *8*, 1389.
28. Karimi-Mamaghan, M.; Mohammadi, M.; Meyer, P.; Karimi-Mamaghan, A.M.; Talbi, E.G. Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art. *European Journal of Operational Research* **2022**, *296*, 393–422.
29. Gómez-Rubio, Á.; Soto, R.; Crawford, B.; Jaramillo, A.; Mancilla, D.; Castro, C.; Olivares, R. Applying Parallel and Distributed Models on Bio-Inspired Algorithms via a Clustering Method. *Mathematics* **2022**, *10*, 274. doi:10.3390/math10020274.
30. Caselli, N.; Soto, R.; Crawford, B.; Valdivia, S.; Olivares, R. A Self-Adaptive Cuckoo Search Algorithm Using a Machine Learning Technique. *Mathematics* **2021**, *9*, 1840. doi:10.3390/math9161840.
31. Soto, R.; Crawford, B.; Molina, F.G.; Olivares, R. Human Behaviour Based Optimization Supported With Self-Organizing Maps for Solving the S-Box Design Problem. *IEEE Access* **2021**, *9*, 84605–84618. doi:10.1109/access.2021.3087139.

32. Valdivia, S.; Soto, R.; Crawford, B.; Caselli, N.; Paredes, F.; Castro, C.; Olivares, R. Clustering-Based Binarization Methods Applied to the Crow Search Algorithm for 0/1 Combinatorial Problems. *Mathematics* **2020**, *8*, 1070. doi:10.3390/math8071070.
33. Maturana, J.; Lardeux, F.; Saubion, F. Autonomous operator management for evolutionary algorithms. *Journal of Heuristics* **2010**, *16*, 881–909.
34. dos Santos, J.P.Q.; de Melo, J.D.; Neto, A.D.D.; Aloise, D. Reactive search strategies using reinforcement learning, local search algorithms and variable neighborhood search. *Expert Systems with Applications* **2014**, *41*, 4939–4949.
35. Khan, A.; Cao, X.; Xu, B.; Li, S. Beetle Antennae Search: Using Biomimetic Foraging Behaviour of Beetles to Fool a Well-Trained Neuro-Intelligent System. *Biomimetics* **2022**, *7*, 84. doi:10.3390/biomimetics7030084.
36. Zennaki, M.; Ech-Cherif, A. A new machine learning based approach for tuning metaheuristics for the solution of hard combinatorial optimization problems. *Journal of Applied Sciences(Faisalabad)* **2010**, *10*, 1991–2000.
37. Lessmann, S.; Caserta, M.; Arango, I.M. Tuning metaheuristics: A data mining based approach for particle swarm optimization. *Expert Systems with Applications* **2011**, *38*, 12826–12838.
38. Liang, X.; Li, W.; Zhang, Y.; Zhou, M. An adaptive particle swarm optimization method based on clustering. *Soft Computing* **2015**, *19*, 431–448.
39. Harrison, K.R.; Ombuki-Berman, B.M.; Engelbrecht, A.P. A parameter-free particle swarm optimization algorithm using performance classifiers. *Information Sciences* **2019**, *503*, 381–400.
40. Dong, W.; Zhou, M. A supervised learning and control method to improve particle swarm optimization algorithms. *IEEE Transactions on Systems, Man, and Cybernetics: Systems* **2016**, *47*, 1135–1148.
41. Kurek, M.; Luk, W. Parametric reconfigurable designs with machine learning optimizer. 2012 International Conference on Field-Programmable Technology. IEEE, 2012, pp. 109–112.
42. Al-Duoli, F.; Rabadi, G. Data mining based hybridization of meta-RaPS. *Procedia Computer Science* **2014**, *36*, 301–307.
43. Wang, G.; Chu, H.E.; Zhang, Y.; Chen, H.; Hu, W.; Li, Y.; Peng, X. Multiple parameter control for ant colony optimization applied to feature selection problem. *Neural Computing and Applications* **2015**, *26*, 1693–1708.
44. Seyyedabbasi, A.; Aliyev, R.; Kiani, F.; Gulle, M.U.; Basyildiz, H.; Shah, M.A. Hybrid algorithms based on combining reinforcement learning and metaheuristic methods to solve global optimization problems. *Knowledge-Based Systems* **2021**, *223*, 107044.
45. Sadeg, S.; Hamdad, L.; Remache, A.R.; Karech, M.N.; Benatchba, K.; Habbas, Z. Qbso-fs: A reinforcement learning based bee swarm optimization metaheuristic for feature selection. Advances in Computational Intelligence: 15th International Work-Conference on Artificial Neural Networks, IWANN 2019, Gran Canaria, Spain, June 12-14, 2019, Proceedings, Part II 15. Springer, 2019, pp. 785–796.
46. Sagban, R.; Ku-Mahamud, K.R.; Bakar, M.S.A. Nature-inspired parameter controllers for ACO-based reactive search. *Research Journal of Applied Sciences, Engineering and Technology* **2015**, *11*, 109–117.
47. Nijimbere, D.; Zhao, S.; Gu, X.; Esangbedo, M.O.; Dominique, N. Tabu search guided by reinforcement learning for the max-mean dispersion problem. *Journal of Industrial and Management Optimization* **2020**, *17*, 3223–3246.
48. Reyes-Rubiano, L.; Juan, A.; Bayliss, C.; Panadero, J.; Faulin, J.; Copado, P. A biased-randomized learnheuristic for solving the team orienteering problem with dynamic rewards. *Transportation Research Procedia* **2020**, *47*, 680–687.
49. Kusy, M.; Zajdel, R. Stateless Q-learning algorithm for training of radial basis function based neural networks in medical data classification. Intelligent Systems in Technical and Medical Diagnostics. Springer, 2014, pp. 267–278.
50. Kalaiselvi, B.; Pushparani, M. A novel impulsive genetic fuzzy C-means for task scheduling and hybridization of improved Fire Hawk optimizer and enhanced Deep Q-Learning algorithm for load balancing in cloud computing. *Journal of Data Acquisition and Processing* **2023**, *38*, 1091.
51. Agrawal, P.; Abutarboush, H.F.; Ganesh, T.; Mohamed, A.W. Metaheuristic algorithms on feature selection: A survey of one decade of research (2009-2019). *Ieee Access* **2021**, *9*, 26766–26791.
52. Dokeroglu, T.; Deniz, A.; Kiziloz, H.E. A comprehensive survey on recent metaheuristics for feature selection. *Neurocomputing* **2022**.

53. Ren, K.; Zeng, Y.; Cao, Z.; Zhang, Y. ID-RDRL: a deep reinforcement learning-based feature selection intrusion detection model. *Scientific Reports* **2022**, *12*. doi:10.1038/s41598-022-19366-3.
54. Priya, S.; Kumar, K. Feature Selection with Deep Reinforcement Learning for Intrusion Detection System. *Computer Systems Science & Engineering* **2023**, *46*.
55. Barrera-García, J.; Cisternas-Caneo, F.; Crawford, B.; Gómez Sánchez, M.; Soto, R. Feature Selection Problem and Metaheuristics: A Systematic Literature Review about Its Formulation, Evaluation and Applications. *Biomimetics* **2024**, *9*, 9.
56. Mengash, H.A.; Alruwais, N.; Kouki, F.; Singla, C.; Abd Elhameed, E.S.; Mahmud, A. Archimedes Optimization Algorithm-Based Feature Selection with Hybrid Deep-Learning-Based Churn Prediction in Telecom Industries. *Biomimetics* **2023**, *9*, 1. doi:10.3390/biomimetics9010001.
57. Jiang, Y.; Wu, Q.; Zhu, S.; Zhang, L. Orca predation algorithm: A novel bio-inspired algorithm for global optimization problems. *Expert Systems with Applications* **2022**, *188*, 116026.
58. Sutton, R.S.; Barto, A.G. *Reinforcement learning: An introduction*; MIT press, 2018.
59. Wang, L.; Pan, Z.; Wang, J. A review of reinforcement learning based intelligent optimization for manufacturing scheduling. *Complex System Modeling and Simulation* **2021**, *1*, 257–270.
60. Sun, H.; Yang, L.; Gu, Y.; Pan, J.; Wan, F.; Song, C. Bridging locomotion and manipulation using reconfigurable robotic limbs via reinforcement learning. *Biomimetics* **2023**, *8*, 364.
61. Zhu, K.; Zhang, T. Deep reinforcement learning based mobile robot navigation: A review. *Tsinghua Science and Technology* **2021**, *26*, 674–691.
62. Azar, A.T.; Koubaa, A.; Ali Mohamed, N.; Ibrahim, H.A.; Ibrahim, Z.F.; Kazim, M.; Ammar, A.; Benjdira, B.; Khamis, A.M.; Hameed, I.A.; others. Drone deep reinforcement learning: A review. *Electronics* **2021**, *10*, 999.
63. Alavizadeh, H.; Alavizadeh, H.; Jang-Jaccard, J. Deep Q-learning based reinforcement learning approach for network intrusion detection. *Computers* **2022**, *11*, 41.
64. Zhang, L.; Tang, L.; Zhang, S.; Wang, Z.; Shen, X.; Zhang, Z. A Self-Adaptive Reinforcement-Exploration Q-Learning Algorithm. *Symmetry* **2021**, *13*, 1057. doi:10.3390/sym13061057.
65. Jang, B.; Kim, M.; Harerimana, G.; Kim, J.W. Q-learning algorithms: A comprehensive classification and applications. *IEEE access* **2019**, *7*, 133653–133667.
66. Wang, H.n.; Liu, N.; Zhang, Y.y.; Feng, D.w.; Huang, F.; Li, D.s.; Zhang, Y.m. Deep reinforcement learning: a survey. *Frontiers of Information Technology & Electronic Engineering* **2020**, *21*, 1726–1744.
67. Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A.A.; Veness, J.; Bellemare, M.G.; Graves, A.; Riedmiller, M.; Fidjeland, A.K.; Ostrovski, G.; others. Human-level control through deep reinforcement learning. *nature* **2015**, *518*, 529–533.
68. Diekmann, N.; Walther, T.; Vijayabaskaran, S.; Cheng, S. Deep reinforcement learning in a spatial navigation task: Multiple contexts and their representation. *2019 Conference on Cognitive Computational Neuroscience* **2019**. doi:10.32470/ccn.2019.1151-0.
69. Schaul, T.; Quan, J.; Antonoglou, I.; Silver, D. Prioritized Experience Replay. *CoRR* **2015**.
70. Ramicic, M.; Bonarini, A. Correlation minimizing replay memory in temporal-difference reinforcement learning. *Neurocomputing* **2020**, *393*, 91–100. doi:10.1016/j.neucom.2020.02.004.
71. Liu, F.; Viano, L.; Cevher, V. Understanding Deep Neural Function Approximation in Reinforcement Learning via ϵ-Greedy Exploration. *[Proceedings of NeurIPS 2022]* **2022**.
72. Nguyen, B.H.; Xue, B.; Zhang, M. A survey on swarm intelligence approaches to feature selection in data mining. *Swarm and Evolutionary Computation* **2020**, *54*, 100663.
73. Pudjihartono, N.; Fadason, T.; Kempa-Liehr, A.W.; O'Sullivan, J.M. A Review of Feature Selection Methods for Machine Learning-Based Disease Risk Prediction. *Frontiers in Bioinformatics* **2022**, *2*. https://doi.org/10.3389/fbinf.2022.927312.
74. Kaur, S.; Kumar, Y.; Koul, A.; Kumar Kamboj, S. A Systematic Review on Metaheuristic Optimization Techniques for Feature Selections in Disease Diagnosis: Open Issues and Challenges. *Archives of Computational Methods in Engineering* **2022**, *30*, 1863–1895. doi:10.1007/s11831-022-09853-1.
75. Li, J.; Cheng, K.; Wang, S.; Morstatter, F.; Trevino, R.P.; Tang, J.; Liu, H. Feature selection: A data perspective. *ACM computing surveys (CSUR)* **2017**, *50*, 1–45.
76. Hoque, N.; Bhattacharyya, D.K.; Kalita, J.K. MIFS-ND: A mutual information-based feature selection method. *Expert Systems with Applications* **2014**, *41*, 6371–6385.

77. Peng, Y.; Wu, Z.; Jiang, J. A novel feature selection approach for biomedical data classification. *Journal of Biomedical Informatics* **2010**, *43*, 15–23.
78. Ma, L.; Li, M.; Gao, Y.; Chen, T.; Ma, X.; Qu, L. A novel wrapper approach for feature selection in object-based image classification using polygon-based cross-validation. *IEEE Geoscience and Remote Sensing Letters* **2017**, *14*, 409–413.
79. Kaur, S.; Kumar, Y.; Koul, A.; Kumar Kamboj, S. A systematic review on metaheuristic optimization techniques for feature selections in disease diagnosis: open issues and challenges. *Archives of Computational Methods in Engineering* **2023**, *30*, 1863–1895.
80. Tan, F.; Yan, P.; Guan, X. Deep reinforcement learning: from Q-learning to deep Q-learning. Neural Information Processing: 24th International Conference, ICONIP 2017, Guangzhou, China, November 14–18, 2017, Proceedings, Part IV 24. Springer, 2017, pp. 475–483.
81. Ramaswamy, A. Theory of Deep Q-Learning: A Dynamical Systems Perspective. *ArXiv* **2020**, *abs/2008.10870*.
82. Hong, Z.W.; Su, S.Y.; Shann, T.Y.; Chang, Y.H.; Lee, C.Y. A Deep Policy Inference Q-Network for Multi-Agent Systems. *ArXiv* **2017**, *abs/1712.07893*.
83. Hu, X.; Chu, L.; Pei, J.; Liu, W.; Bian, J. Model complexity of deep learning: a survey. *Knowledge and Information Systems* **2021**, *63*, 2585–2619. doi:10.1007/s10115-021-01605-0.
84. Fan, J.; Wang, Z.; Xie, Y.; Yang, Z. A theoretical analysis of deep Q-learning. Learning for dynamics and control. PMLR, 2020, pp. 486–489.
85. Bartz-Beielstein, T.; Preuss, M. Experimental research in evolutionary computation. Proceedings of the 9th annual conference companion on Genetic and evolutionary computation. ACM, 2007, GECCO07. doi:10.1145/1274000.1274102.
86. Ravelo, C.; Olivares, R. Biomimetic Orca Predator Algorithm improved by Deep Reinforcement Leaning for Feature Selection, 2024. doi:10.6084/M9.FIGSHARE.25126043.