PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

NODE SELECTION TECHNIQUE ON BRANCH & BOUND ALGORITHM ASSISTED BY REINFORCEMENT LEARNING

Alen David Figueroa Mandujano

AVANCE DE SEMINARIO DE TESIS
PROGRAMA DE DOCTORADO EN INGENIERÍA INFORMÁTICA

Mayo, 2021

PONTIFICIA UNIVERSIDAD CATÓLICA DE VALPARAÍSO
FACULTAD DE INGENIERÍA
ESCUELA DE INGENIERÍA INFORMÁTICA

NODE SELECTION TECHNIQUE ON BRANCH & BOUND ALGORITHM ASSISTED BY REINFORCEMENT LEARNING

Alen David Figueroa Mandujano

Ignacio Araya Zamorano
Supervisor de Tesis

Mayo, 2021

Abstract

Mixed integer nonlinear programming (MINLP) is a generalization of classic optimization with nonlinearities in the objective function or their constraints, and can have continuous or integer variables. Various real-life problems can be approached as a MINLP problem, such as on chemical engineering, production planning, networks design and many more.

State-of-the-art solvers use the branch-and-bound algorithm, an exact method that, in combination with a diverse mixture of heuristics, can tackle a fair range of practical problems. This algorithm sequentially partitions the search space using linear relaxations, thus creating a search tree. The exploration ends only when a solution, together with its proof of optimality, is found. The tree's size can vary dramatically depending on the approach that is used to create it and explore it.

One of the components in the algorithm is the node selection policy, a criterion used to select which node or sub-region will be processed next. Currently, there is a wide variety of node selection policies, and some may get better performance than others in most cases.

The present work aims to explore a line of research in the intersection of global optimization and machine learning. Specifically, we make steps in the direction of discovery of a node selection policy through deep reinforcement learning techniques. Our goal is to achieve data-driven acceleration of the tree search.

Keywords: Optimization Problem · Branch and Bound · Reinforcement Learning · Reward Functions

Contents

List of Figures . . . . . **i**

1 Introduction **1**
1.1 Research Questions 1
1.2 Hipothesis 2
1.3 Objectives 2
1.3.1 Main Objective 2
1.3.2 Specific Objectives 2
1.4 Research Methodology 3

2 Related Work **4**
2.1 Branch & Bound 4
2.2 Upperbounding 4
2.2.1 Pruning 5
2.2.2 Branching 5
2.3 Machine Learning 6
2.3.1 Imitation Learning 6
2.4 Reinforcement learning 6
2.4.1 Q-Learning 7
2.5 Deep reinforcement learning 8
2.5.1 Deep Q-learning 8
2.6 Machine learning on optimization 10
2.6.1 Hybrid Branch & Bound techniques 10
2.6.2 Machine learning on Branch & Bound on solving NCOP . . . 11

3 Proposal **12**
3.1 Rewards 13

List of Figures

1 General workflow of a deep Q-Learning algorithm. 8
2 Scheme of a deep Q-learning architechture 9
3 Proposal Workflow of communication 13

1 Introduction

Mixed integer nonlinear programming (MINLP) is a generalization of classic optimization with nonlinearities in the objective function or their constraints, and can have continuous or integer variables. Various real-life problems can be approached as a MINLP problem, such as on chemical engineering, production planning, networks design and many more.

MINLP are considered NP-hard. And state-of-the-art solvers use a technique called Branch & Bound to solve them. Although this algorithm was intended to be used on MILP, they achieved good results in this context. This approach subdivides the search space using linear relaxations, making a search tree in a sequential fashion. The search is finished only when a solution is found and its optimality is proven. From this, the resources, such as time and memory space, that requires is dependant on the heuristics that are used to build the strategy.

There are various components that influence the Branch & Bound algorithm efficiency, such as the branching policy, the pruning method or the node selection policy. Each one of them have been explored and optimized using different approaches, such as rules, mathematical proof and, most recently, using machine and deep learning models.

One of the most important components on the Branch & Bound algorithm is the node selection policy. This technique defines which node has to be analyzed next, so it defines the path that follows the search algorithm. Currently, there is no fully efficient node selection policy for all the cases that may occur, and due to the nature of the problem they tackle, it is complex to design a policy that complies with this characteristic.

Machine learning can be an effective solution to this problem. Thanks to its generalization capabilities, they allow the use of policies that could not be thought of by a researcher because of their complexity.

Machine learning is a field that include algorithms and statistical models to make predictions or infer data by using only a sample of the total population. These techniques are able to make generalizations over the showed data, recognize patterns and make predictions. These models have entered most recently on the task of searching better algorithms to optimize. Their generalization capabilities allow the use of policies that could not be thought of by a researcher because of their complexity.

This work explores the use of machine learning techniques to develop a new node selection policy, inspired on the state of the art for this task. We focus on the task of solving MINLP problems. Currently, the few proposals that tackle the node selection policy trough these type of branches is

1.1 Research Questions

Through this research, the questions to be answered are the following:

- How to develop reinforcement learning techniques for node selection on Branch & Bound algorithms on MINLP problems?
- Is is possible that reinforcement learning techniques for node selection improve the performance of Branch & Bound algorithms on MINLP problems?
- How the reward function affects the performance of reinforcement learning techniques for Branch & Bound algorithms on MINLP problems?

1.2 Hipothesis

On this section the hypotheses for this work will be presented. Two hypotheses will be proposed, with their null hypotheses.

- The use of reinforcement learning-based node selection policies improve the performance of Branch & Bound techniques for solving MINLP problems.
- $\mathbf{H_0}$: The use of reinforcement learning-based node selection policies **do not** improve the performance of Branch & Bound techniques for solving MINLP problems
- The reward function selection for reinforcement learning-based node selection policies affect the performance of Branch & Bound techniques for solving MINLP problems.
- $\mathbf{H_0}$: The reward function selection for reinforcement learning-based node selection policies **do not** affect the performance of Branch & Bound techniques for solving MINLP problems.

1.3 Objectives

Through the research questions and hypotheses of this work, the main and specific objectives will be presented in this section.

1.3.1 Main Objective

To develop a reinforcement learning-based node selection policy for the Branch & Bound technique to solve MINLP problems. Performing in turn, different adjustments, such as reward functions, metrics and state representations.

1.3.2 Specific Objectives

- To study the state of the art on node selection policy for Branch & Bound techniques.
- To study the sate of the art on machine learning-assisted techniques on Branch & Bound components.

- To propose different components for modelling reinforcement learning-based node selection policy for Branch & Bound techniques.
- Compare the results of the different models proposed with the actual state of the art for node selection policy on Branch & Bound techniques.

1.4 Research Methodology

The research methodology for this work is composed by a quantitative approach and explanatory scope. This is because there is a comparison between existing techniques and the proposed ones.

2 Related Work

As in [1], A numerical constrained optimization problem is defined as a constraint system $S = (C, x, \mathbf{x})$, where x is a vector of n variables. Considering a real function $f_{obj} : \mathbb{R}^n \rightarrow \mathbb{R}$. The numerical constrained optimization problem, related to the objective function f_{obj} and the constraint system S, consist of finding: $x* = \underset{x \epsilon \mathbf{x}}{min} f_{obj}(x)$ s.t. x satisfies the constraints in C

2.1 Branch & Bound

Branch and Bound methods are a solution approach that can work on this type of problem. In the context of a NCOP, our objective is to find an optimal solution $x* = \underset{x \epsilon \mathbf{x}}{min} f_{obj}(x)$ s.t. x satisfies the constraints in C. To solve these problems, Branch and Bound algorithms iteratively generate a search tree of subregions of the search space. The basic algorithm for this search is shown on algorithm 1. It start by defining an upper bound $\hat{x}$, then, at each iteration it selects a non explored node from a list of unexplored regions. if a solution $\hat{x}'$ is found on the selected node, and their objective value is better than the actual $\hat{x}$, their value is updated. On the other hand, if the objective value of $\hat{x}'$ is not better than the actual global optimal $\hat{x}$, the node is pruned and the region becomes a leaf node. Finally, if is not possible to discard all the region on the node, the region is subdivided into more substets (usually 2) that then are added to the list of unexplored regions. The algorithm is executed until there are no regions left on the list and the value of $\hat{x}$ is the global minimum. [2]

Algorithm 1 Branch & Bound basic algorithm

```
1:  L ← [X]
2:  x̂ ← inf
3:  while L ≠ ∅ do
4:      S ← select_node_from(L)
5:      x̂' ← estimate_lb(S)
6:      if x̂' ≠ null and f_obj(x̂') < f_obj(x̂) then
7:          x̂ ← x̂'
8:      else
9:          S ← prune(S)
10:         if S ≠ ∅ then
11:             S_list ← partition(S)
12:             insert(L, S_list)
```

2.2 Upperbounding

The procedure to find and discard nodes on the search is called upper bounding. This process consists of finding feasible solutions with lower cost than the most optimal solution found yet, called upper bound (UB). This procedure implies a reduction of the search space, so it is a crucial component of the Branch and Bound algorithm. The

capacity of making a good upper bounding relies on a series of key components which can drastically affect its performance [3].

2.2.1 Pruning

The first of the key components is the Pruning, which consists of filtering the solutions within a box to the constraints or comparing to the solutions found on previous iterations of the algorithm. This action is made by a contractor, which will apply mathematical policies to the solution discarding process.

2.2.2 Branching

The second is the branching, a procedure that aims to make the subdivisions of the region, thus creating the nodes that will be explored. This task has two approaches: Binary branching and Wide branching. The result of Binary Branching techniques are the division of a node into two mutually exclusive regions. On the other hand, wide branching creates sub-regions which not necessarily are mutually-exclusive, so a solution can be found in multiple paths, making the search redundant at times.

The branching policies have 2 sub tasks: variable selection and value selection. For variable selection, several approaches exists, such as the use of the round robin algorithm, which can be really expensive if there is a bad order of variables at start, Largest-First, which select the variable with the largest domain in the box in order to reduce it on early stages of the solver. These approaches tend to be effective in various scenarios, but there is space to improvements related to more specific instances. Another task that can be improved is node selection. For NCOPs, a good selection of nodes can lead to a significant reduction of the search space. This can be explained by the tree fashion of these techniques; discarding large nodes on early stages of the process implies a significant reduction of the search space [3]. There are 4 main policies for node selection used among the different implementations of Branch and Bound techniques: Depth first search, breadth first search, best first search and feasible diving.

- **Depth First Search**: is a classical strategy for traversing graph-like structures. On the Branch and Bound algorithm, this policy implies that the node that was most recently added to the list of unexplored regions will be always selected [4].

- **Breath Fisrst Search**: a policy which aims to explore all sub-regions that are at some fixed distance from the root before starting exploring more deeper nodes. This kind of strategy is really efficient at finding solutions that are closer from the first regions, which can be helpful in instances of problems that generate more imbalanced trees, but overall it costs too much to be used broadly [5].

- **Best first search**: is a strategy which aims to prioritize the exploring of regions that are potentially more promising, from a measuring via some heuristic. Normally, on Branch and Bound, the lower bound (lb) which corresponds to the best possible solution on the node. The main advantage of using this approach is that, because of the aim to always be trying to improve the UB, which in most cases achieve good results on finding the best solution on more early stages of the algorithm [6]. However, one important drawback of this technique is that it can become way less efficient in some contexts where there are too many nodes where their heuristic value is the same. In these cases, BFS becomes slower than other strategies [7].

- **Feasible diving**: is the state-of-the-art technique for node selection. It aims to solve the problems of BFS and DFS by combining them. For it to work, a new rule is added to the node selection of BFS; the depth of the node in the search tree. With this addition, the algorithm selects the nodes in a greedy-DFS fashion, which has the objective dive into a promising region in order to prove their optimality or discard the best region [2][1].

2.3 Machine Learning

Machine learning is a field of computer science and artificial intelligence that define a series of algorithms that can make inferences about relationships between data in order to make predictions. This area can be subdivided into 3 main areas of learning paradigms: supervised learning, unsupervised learning and reinforcement learning. Supervised learning techniques require use of example data with already computed results, namely labeled data. The tasks that can make this type of technique are prediction of values or classification. Unsupervised learning is a paradigm that is used when the main objective is to find relationships between unlabeled data, allowing finding representative characteristics of them.

2.3.1 Imitation Learning

Imitation learning is a subclass of supervised learning algorithms. Its operation is based on the learning of a given policy, called guru, on which the model is intended to be able to approximate or imitate its behavior. The policy can be simple, such as a function to be calculated, or complex, such as an overly expensive algorithm on which it is sought to reduce its complexity.

2.4 Reinforcement learning

Reinforcement learning is an area of machine learning which orients the learning process in a reward-driven fashion. For it to work, it is important to have a correct definition and use of the terms of a policy, a value function, function reward and environment model.

- In a reinforcement learning context, a policy is defined as a strategy followed by the learning agent in order to make a decision from the environment. It can be defined as a markov decision process (S, A, R, P), where S contain the internal states of the agent, A is defined as the set of actions that it can make, P is the transition matrix of the markov process, and R is the reward function of the process.

- The value function is the estimation of how good can be an action given the current state. Their estimation is defined as the expected rewards that can be achieved from the execution of the policy from the current state to a final state. It differs from the reward in the fact that the value function aims to measure the future outcome of the policy, in contrast with the reward.

- A reward is the objective function of the agent. It translates the state-action pair to a defined value.

- Finally, the environment model is a representation, or simulation, of the actual environment. In some cases the learning over a real environment is impossible (due to ethical or technical reasons) so the use of an environment can depend on the task from which to learn.

2.4.1 Q-Learning

Q-Learning **QLearning** is a Reinforcement Learning method that attacks the problem of learning to control autonomous agents, through trial and error interactions with a dynamic environment, which provides reinforcement signals for each action performed. Q-Learning allows solving sequential decision problems in which the utility of an action depends on a sequence of decisions and where there is uncertainty about the dynamics of the environment in which the agent is located.

Q-Learning contains different aspects that are the basis of its operation. First, this technique requires a structure called Q-Table, which will be a map where the relationship between a state, an action and its reward. This reward needs to be computed, which is done via the bellman equation:

$$\underbrace{\text{New}Q(s,a)}_{\substack{\text{Nuevo}\\ \text{Q-Valor}}} = Q(s,a) + \underbrace{\alpha}_{\substack{\text{Factor de}\\ \text{Aprendizaje}}} [\, \underbrace{R(s,a)}_{\substack{\text{Recompensa}\\ \text{Obtenida}}} + \underbrace{\gamma}_{\substack{\text{Tasa de descuento}}} \overbrace{\max Q'(s',a')}^{\substack{\text{Recompensa máxima predicha, dado}\\ \text{un nuevo estadoy todas las acciones posibles}}} - Q(s,a)]$$

The values that evaluate the reward are given by the Q-Tables. There are some difficulties with this technique: In domains where the states are of high dimensionality,

performing the training is too costly, since traversing each state is infeasible. Within this problem, there is also an associated resource cost, since it is necessary to store the Q-Table to be able to make decisions, even when it is no longer in the learning phase.

2.5 Deep reinforcement learning

Deep RL is a method that aims to combine reinforcement learning techniques with deep learning models. These techniques incorporate deep neural network models to make decisions on unstructured data, which allows them to be useful to operate in environments where there is too much information to take into account to make a decision. Among its proposals is Deep Q-Learning, which seeks to replace the computation of the Q function of this model with a deep learning model, which approximates its operation. **DQN**

2.5.1 Deep Q-learning

Deep Q-learning (DQN) is based on the use of imitation learning for the approximation of the Q-Table, so that not only the cost of storing this data is reduced, but also this model would be able to generalize about the actions, so it will not have to go through all the possible states to be able to know its expected reward.

The paradigm on which it operates is presented in figure 1. It starts with the network being initialized with random weights. As shown in figure 2, the network will have as input the state of the agent. As output, there will be one neuron for each action it can take, the value of each of these being the Q-value that approximates the network. From these values, the expected value is calculated using the Bellman equation, to finally use it in the algorithm for adjusting the weights of the network.

Figure 1: General workflow of a deep Q-Learning algorithm.

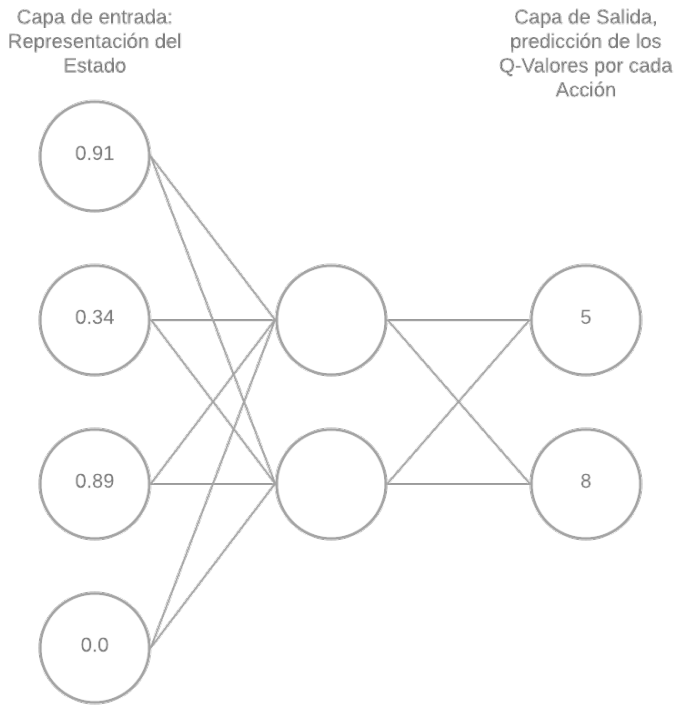

Figure 2: Scheme of a deep Q-learning architechture

For the correct operation of this model, it is necessary to separate the learning into 2 networks, which must have the same architechture, but differ only on the associated weights on each of them. These will be called the main network and the target network, respectively. During the learning process, the weights of the main network will be transferred to the target network. This transfer will be performed in a certain number of iterations, determined by the researcher. Finally, to replace the use of the Q-Table in the Bellman equation, the values predicted by the target network are considered.

De esta manera, el procedimiento sería el siguiente:

1. From a state, an action is predicted through the main network.
2. With the objective network, the reward its computed.
3. The weights on the main network are updated.
4. Every n iterations, the weights on the main network are copied to the objective network.

2.6 Machine learning on optimization

Artificial intelligence-based methods have supported more traditional approaches of optimization in various ways. From the use of neural networks to approximate a complex function in order to use it as a solver for specific settings [8], the use of unsupervised learning techniques to affect the parameter tuning over metaheuristics [9] and complete approaches [10], or even to make use of neural networks to support a local minimum escape algorithm over a gradient descent solver [11]. These approaches have been helpful in the sense of reduction of computational time and, in some cases, computational power needed to perform them. In the last subject, there has to be addressed the implication in the use of more sophisticated techniques such as deep learning, which tend to be computationally demanding.

2.6.1 Hybrid Branch & Bound techniques

Intelligent agents have been incorporated in Branch and Bound methods in various forms. The most explored approach has been in the approximation of computationally-heavy policies in order to reduce the time employed to evaluate them. On this subject, [9] analyzed how machine learning models can be used to learn a more optimal policy for variable selection. They concluded that, even when improvements are achieved, there are still challenges in proposing a technique that can be less affected when the instances of a problem are from other application domains. [8] presented a more generic approach through the use of imitation learning to approximate a specific policy called strong branching, a branching strategy which aims to consider all branching candidates before selecting one. Because of this, their implementation is computationally intensive, so the use of imitation learning to approximate the policy behaviour was a good approach. To achieve the approximation effectively, the same parameters used on the computation of the policy were used as input parameters to the model, and the result of the heuristic was used as the objective value for the model to achieve. The results showed that the integration of the agent achieved a good approximation of the policy, but overcoming the heavy computational cost of the original one. [6] Focused on improving the performance of machine learning models to learn branching policies. Machine learning is heavily dependent on the input data representation, so they stated that the correct selection of variables that can represent the actual state of the branching algorithm can be an important influence on the performance of these techniques. They proposed two matrix representations to the branching candidates and the state of the search tree in an effort to capture the dynamics of the Branch and Bound process. This proposal generated a significant improvement compared to the previous technique regarding these techniques, and effectively generated a more resilient model to tackle problem instances on different application domains.

On the other hand, Reinforcement learning has been another approach focused on the creation of more independent policies. Because of their characteristics, they do not require the use of a traditional policy from which to learn to imitate. Instead, Reinforcement learning only requires a good state representation and a computation of

a reward to learn its own policy, and because of that, they can achieve even better results if their representations are correctly proposed. On this subject, [3] presented the use of Deep Q-learning model as a variable selection policy. Their objective was to create a branching policy whose objective is to maintain every sub-tree on their minimal size, to assist a Depth First Search policy of node selection. For this, their reward was stated as the tree size, calculating the inverse size of it. To the state representation, they included static variables, such as the parameters of each instance of the problem, and dynamic parameters, such as the depth of the node to branch, distance of the node to the borders of the domain and the ramification state. Their results showed that their proposal can improve or be comparable to traditional techniques, but being more versatile to be adaptable to more variable problems.

2.6.2 Machine learning on Branch & Bound on solving NCOP

On Machine learning models supporting the Branch and Bound techniques to solving numerical constrained optimization problems, there have been implementations to solve problems related to wireless network planning. [13] Proposed the use of imitation learning to train a SVM model on a classification of the decision making. Specifically, this approach aims to replace the policy to decide if a node has to be pruned or branched with the machine learning model which will approximate the policy behaviour. The proposal achieved better results but implying a reduction of global optimal approximation. These reductions are expected when we are using these techniques, because of their limitations as inferential tools. [14] Proposed the same policy, but replacing the use of SVM with a Deep Learning approach using a multilayer perceptron, with the objective to reduce the reduction of the distance between the found solutions and the global optimum taking advantage of the better learning characteristics of the deep learning models.

3 Proposal

This proposal focuses on create a Deep Reinforcement Learning-based policy for node selection on Branch & bound techniques for solving numerically constrained optimization problems.

Feasible diving is the state of the art approach of node selection for Branch & Bound algorithms. This strategy merges depth first search, also called exploitation and best-first search, called exploration. This technique selects the next node to be evaluated using a heuristic related to the lower bound of each node. The node with the lowest lower bound will be the next to be selected, then, a depth first search strategy is followed until a leaf node is reached and the process is repeated. The improvement point is from the exploitation phase. Here, there is not an early stop policy over the DFS. The objective of this proposal is to find if this early stop can be achieved without compromising the quality of the solutions. For this, a reinforcement learning model will be used in order to predict when the conditions to change between DFS and BFS meet.

In the figure 3, the basic workflow of the proposal is exposed. The Branch & Bound main loop is the main algorithm that performs the search, just like the algorithm 1. When its the time to select the next node to be processed, the node selection algorithm, known as feasible diving, is called. The search parameters are passed to be fed to the reinforcement learning model. In this proposal, these parameters are the instance parameters and the current upper bound. With the search data and the one of the preciously selected node, specifically the depth of the node and its lower bound, the reinforcement learning algorithm will predict if is convenient keep exploiting the zone or stop and begin exploring others. With this decision, feasible diving will select a node and return it to the Branch & Bound algorithm to process it.

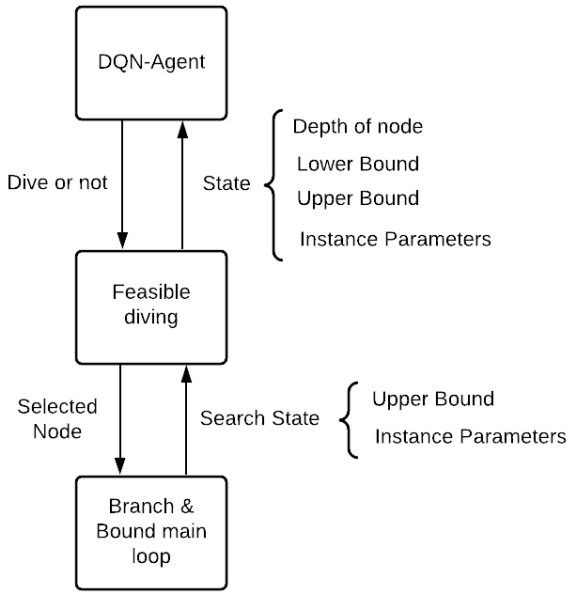

Figure 3: Proposal Workflow of communication

3.1 Rewards

The rewards on Reinforcement Learning models are a fundamental piece for the effective functioning of it, and because of that, the literature proposes various ways to calculate them. The reward chosen determines how well a decision is from the consequences that it caused to the environment. And because of that, the reward is a key component on the proposal.

By the propositions made on [12], when a DFS strategy for node selection is made, the Branch & Bound tree minimization is achieved when every sub-tree is a minimal size. By this statement, one of the rewards that will be tested is computed with the nodes generated by the algorithm.

The reward function is the component on reinforcement learning that defines which objective will follow the model. If some concrete action is desired to be followed, then the reward has to get a bigger value when this type of decision is made.

Because of the nature of this problem, a reward can be computed in different times. In this regard, we decided to classify the rewards in two categories: a online, or instantaneous reward, and a offline, or delayed reward. For the offline method, as shown in algorithm 2, the weights on the DQN model will be updated once one instance is solved

using the amount of nodes generated as the reward. First, a random instance is selected to be solved with a epsilon-greedy method using the DQN model. Once the problem is solved, the experience from the resulting tree is stored with its reward. Finally, the weights of the DQN model are updated and another instance is selected to continue the training.[12]

Algorithm 2 Offline training algorithm

```
R ← initialize_DQN()
for t = 0, 1...N do
    i ← random_instance()
    e ←solve_problem(i, R)
    collect_experience(e)
    update_DQN(R)
```

For the online method, the reward has to be instantaneous, in contrast with the offline one. For this, the most instant reward is to check if the upper bound is affected by the action taken. The problem of the feasible diving technique comes when the DFS section cant discard nor confirm that is not necessary to keep exploiting some region. Because of that, when the algorithm is not capable of improve the upper bound, it can be time to explore other regions. Finally, because of that, the DQN model comes useful on a online way by computing a reward via checking if the upper bound is improved with the action taken or not. This can be computed in two ways: through a binary way, if the upper bound is improved or not, or via measuring how much the upper bound is improved. On this work, we will be testing both ways in a effort to find the best way to reward the DQN model.

To balance these rewards, a key point is to make, in contrast, negative rewards. These will keep the DQN model from making inefficient improvements just to obtain more rewards in exchange. To limit this behaviour, a good solution is to punish the DQN model when it takes too much iterations to finish the optimization. From this, knowing that the model will be trained over benchmark instances, there can be a insight of the amount of iterations is the limit to achieve. The use of this value is only necessary on the training process and will be calculated as the equation 2. Finally, the reward will be computed as in the equation 3

$$R(\hat{x}, \hat{x}') = \begin{cases} \frac{\hat{x}-\hat{x}'}{\hat{x}} & \hat{x} > \hat{x}' \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$P(I_{\text{expected}}, I_{\text{current}}) = \begin{cases} I_{\text{current}} - I_{\text{expected}} & I_{\text{expected}} \leq I_{\text{current}} \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

$$R^{\pi}(\hat{x}, \hat{x}', I_{\text{expected}}, I_{\text{current}}) = R(\hat{x}, \hat{x}') - P(I_{\text{expected}}, I_{\text{current}}) \tag{3}$$

Algorithm 3 online training algorithm

procedure SOLVE_PROBLEM(i, R) ▷ i is the instance, R is the DQN model
  while i is not solved **do**
    $S \leftarrow$ feasible_diving_DQN(L, R) ▷ Here the DQN selects the node
    $\hat{x}' \leftarrow$ estimate_lb(S)
    if $\hat{x}' \neq$ *null* and $f_{obj}(\hat{x}') < f_{obj}(\hat{x})$ **then**
      update_DQN($R, R^{\pi}(\hat{x}, \hat{x}', I_{\text{expected}}, I_{\text{current}})$)
      $\hat{x} \leftarrow \hat{x}'$
    else
      update_DQN($R, R^{\pi}(\hat{x}, \hat{x}', I_{\text{expected}}, I_{\text{current}})$)
      $S \leftarrow$ prune(S)
      if $S \neq \varnothing$ **then**
        $S_{list} \leftarrow$ partition(S)
        insert(L, S_{list})

References

[1] Bertrand Neveu, Gilles Trombettoni, and Ignacio Araya. "Node selection strategies in interval branch and bound algorithms". In: *Journal of Global Optimization* 64.2 (2016), pp. 289–304.

[2] Ignacio Araya and Victor Reyes. "Interval Branch-and-Bound algorithms for optimization and constraint satisfaction: a survey and prospects". In: *Journal of Global Optimization* 65.4 (2016), pp. 837–866.

[3] Tobias Achterberg and Roland Wunderling. "Mixed Integer Programming: Analyzing 12 Years of Progress". In: *Facets of Combinatorial Optimization* (2013), pp. 449–481. DOI: 10.1007/978-3-642-38189-8_18. URL: https://link.springer.com/chapter/10.1007/978-3-642-38189-8_18.

[4] Robert Tarjan. "Depth-First Search and Linear Graph Algorithms". In: *SIAM Journal on Computing* 1.2 (1972), pp. 146–160. DOI: 10.1137/0201010. eprint: https://doi.org/10.1137/0201010. URL: https://doi.org/10.1137/0201010.

[5] David R Morrison, Sheldon H Jacobson, Jason J Sauppe, et al. "Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning". In: *Discrete Optimization* 19 (2016), pp. 79–102.

[6] Andrea Cassioli, David Di Lorenzo, Marco Locatelli, et al. "Machine learning for global optimization". In: *Computational Optimization and Applications* 51.1 (2012), pp. 279–303.

[7] E. Sewell and Sheldon Jacobson. "A Branch, Bound, and Remember Algorithm for the Simple Assembly Line Balancing Problem". In: *INFORMS Journal on Computing* 24 (July 2012), pp. 433–442. DOI: 10.1287/ijoc.1110.0462.

[8] Artur M Schweidtmann and Alexander Mitsos. "Deterministic global optimization with artificial neural networks embedded". In: *Journal of Optimization Theory and Applications* 180.3 (2019), pp. 925–948.

[9] Diego Tapia, Broderick Crawford, Ricardo Soto, et al. "A Q-Learning Hyperheuristic Binarization Framework to Balance Exploration and Exploitation". In: *Applied Informatics*. Ed. by Hector Florez and Sanjay Misra. Cham: Springer International Publishing, 2020, pp. 14–28. ISBN: 978-3-030-61702-8.

[10] Elias B. Khalil, Pierre Le Bodic, Le Song, et al. "Learning to Branch in Mixed Integer Programming". In: *Proceedings of the 30th AAAI Conference on Artificial Intelligence*. 2016. URL: files/papers/KhaLebSonNemDil16.pdf.

[11] Haotian Zhang, Jianyong Sun, and Zongben Xu. *Learning to be Global Optimizer*. 2020. arXiv: 2003.04521 [cs.LG].

[12] Marc Etheve, Zacharie Alès, Côme Bissuel, et al. "Reinforcement learning for variable selection in a branch and bound algorithm". In: *International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer. 2020, pp. 176–185.