Date of publication DEC-31, 2023, date of current version DEC-21, 2023
www.computingonline.net/ computing@computingonline.net

Print ISSN 1727-6209
Online ISSN 2312-5381
DOI 10.47839/ijc.22.4.3351

Markov Combinatorial Processes for Reinforcement Learning and Combinatorial Optimization Problems

FRANCESCA GUERRIERO[1], FRANCESCO PAOLO SACCOMANNO [2]

[1]Department of Mechanical, Energy and Management Engineering, University of Calabria, 87036 Rende, Italy (e-mail: francesca.guerriero@unical.it)
[2]Department of Mechanical, Energy and Management Engineering, University of Calabria, 87036 Rende, Italy (e-mail: francescopaolo.saccomanno@unical.it)

Corresponding author: Francesca Guerriero (e-mail: francesca.guerriero@unical.it).

"We acknowledge financial support from: PNRR MUR project PE0000013-FAIR."

ABSTRACT Solving combinatorial optimization problems is a crucial challenge in many real-world applications. These problems require the optimal choice of combinations from a large set of possibilities, subject to the specific constraints of the problem under consideration, in order to maximize or minimize an objective function. In recent years, Reinforcement Learning (RL) has attracted considerable attention as a potential innovative tool for tackling these complex tasks. The main challenge, to solve combinatorial optimization problems using RL, is related to the need of overcoming the sequential nature of the Markov Decision Processes (MDP) model, on which the solution algorithms are based. In this work, we present an extension of the MDP, that enables software agents to learn from a model, that better reflects the non-sequential nature of these problems. The results demonstrate that, for the first time, a software agent can provide optimal results or, at the very least, solutions with minimal deviation from the optimal values, in the majority of the benchmark instances used in the computational study.

KEYWORDS markov decision process; combinatorial optimization; reinforcement learning; bin packing problem;

I. INTRODUCTION

SOLVING combinatorial optimization (CO) problems represents a fundamental challenge in a wide range of real-world applications, where the main goal is to find an optimal combination of elements, from a discrete set of possibilities, subject to solution feasibility constraints, in order to maximize or minimize a desired objective function.

In recent years, Reinforcement Learning (RL), a branch of machine learning, has gained considerable attention as a possible innovative approach to address CO problems ([1], [2], [3]). This approach is based on the ability of an agent to learn through interaction with an environment, that models the problem under consideration, in order to maximize a cumulative reward appropriately linked to the desired objective function.

In the theoretical framework of RL, Markov Decision Processes (MDPs) model play a central role, serving as the formal representation of several decision problems. However, transposing this model to CO problems presents significant challenges. The main difficulty arises from the sequential nature of MDPs, which results in the generation of large search spaces.

Building upon the aforementioned considerations, this paper aimed at developing an extension of MDPs, that is more suitable for modeling and solving CO problems. The key idea is to train the agent to build the MDPs, deciding which states to accept or not and their value.

The proposed strategy is used to solve the Bin Packing Problem (BPP). The defined approach enables to limit the size of the search space, facilitating the determination of optimal results for the majority of tested instances, and near-optimal results for the remaining ones.

The rest of the paper is organized as follows. Section II provides an overview of the RL framework, describes the underlined MDPs model, and introduces the BPP, to which the proposed approach is applied. Section III offers

a review of the use of MDPs in solving RL approaches, with a specific focus on the BPP. Section IV includes a general description of the proposed approach, along with a version tailored for solving the BPP. Section V discusses the computational results obtained by applying the proposed approach to benchmark instances of the BPP. Finally Section VI reports the main conclusions of our work and a description of possible future developments.

II. PROBLEM DESCRIPTION

In the RL framework, a software agent (i.e., the algorithm) learns through a mechanism of rewards and punishments, by interacting with the environment ([4], [3]). The main goal of this agent is to maximize the total reward, expressed as:

$$Gt = \sum_{k=0}^{H} \gamma^k r_{t+k+1}, \tag{1}$$

where t is a generic step, γ is the discount factor and H is the horizon, that can be infinite.

This environment is typically modeled mathematically as a MDP ([4], [3]) In these scenarios, probabilities and rewards are typically unknown or only partially known, and the agent learns them through interactions with the environment.

The main components of the MDP are represented by the tuple (S, A, P, R, γ) where:

- S is a finite set of states, represented by $s \in S$.
- A is a finite set of actions, represented by $a \in A$.
- $P(s'|s, a)$ denotes the state transition probabilities, which represent the probability of transitioning from state s to state s' when taking action a. It is defined as $P(s'|s, a) = Pr(S_{t+1} = s'|S_t = s, A_t = a)$.
- $r(s', s, a)$ represents the expected reward for transitioning from state s to state s' by taking action a. It is defined as $r(s', s, a) = E[R_{t+1}|S_{t+1} = s', S_t = s, A_t = a]$, where R_{t+1} is the reward obtained after the transition.

The agent learns to make decisions through direct interaction with its environment. It takes actions, receives feedback in the form of rewards or punishments, and gradually refines its decision-making process. This training process involves learning a strategy, often referred to as a policy denoted as π, which enables the agent to maximize its return G ([4], [5], [6], [3], [7]).

Several contributions addressing CO problems using RL techniques, have been published in the scientific literature tackles various problems, including but not limited to the traveling salesman problem, set covering, and BPP ([3], [6], [8], [1]). In this work, we focus on the BBP, an NP-Hard problem ([9]), whose maim aim is to insert a predetermined number of elements of different weights into the smallest possible number of containers ([10], [11], [12], [13]).

Formally, the BPP involves the task of packing $\mathtt{n}$ items, each with size $\mathtt{I}_i$ $(0 \leq i < \mathtt{n})$, into the minimum number of bins. This must be done while ensuring that the capacity $\mathtt{c}$ of each bin is not exceeded.

The BPP arises in several real-life applications. For this reason, both exact approaches and ad-hoc heuristics have been developed. The first exact method, that is a branch & bound approach, has been proposed by Martello et al. in [10]. This method has been improved by the same authors in [14]. A different approach has been developed in [15], where the Bin Completion procedure is proposed. In this approach, rather considering the different containers into which each item can be placed, the authors consider the different ways each container can be filled. This work was subsequently improved by the same authors in [16].

DeLorme et al, in [17], review the most important exact algorithms developed for solving the BPP.

Due to its NP-hard nature, numerous heuristic approaches have been devised to address the BPP ([18], [19], [20]). The first and most well-known heuristics are Next-Fit (NF), First-Fit (FF) and Best-Fit (BF) methods, and their derivatives ([21], [22]).

We refer the readers to the survey [23] for approximation approaches for the classical BPP and its generalizations. For metaheuristic approaches, we recommend the review paper [24].

III. STATE OF ART

MDPs find application in a wide spectrum of CO problems. However, they suffer from what is known as the "curse of dimensionality", i.e. the size of the model grows exponentially compared to the size of the problem to be treated, and therefore the use of the MDPs becomes intractable both in terms of time and storage complexity ([25], [26]).

In the literature there are several approaches to overcome the "curse of dimensionality", which can be divided into two classes: those based on "Approximation", which try to approximate the search space, for example by aggregating states or building heuristic versions of exact algorithms ([27], [28], [29], [30]), and those based on "Simulation", where the model is built using simulations and neural networks ([31], [32], [4], [33], [34], [35]).

Effective methods for solving MDPs include the Value Iteration method, the Policy Iteration method, regular LP interior-point algorithms, and the Vavasis-Ye algorithm, to cite a few ([36], [37], [38]).

The Value Iteration method is an iterative approach used to find the optimal value function which describes how good is to be in a specific state s when following a certain policy π:

$$V_\pi(s) = E[G_t|S_t = s].$$

In particular, $V_\pi(s)$ represents the expected return (i.e., the expected cumulative discounted reward) when starting from state s and following policy π. The process begins with an initial estimate or a random value for the value function and iteratively updates it until convergence. During

each iteration, it computes the new value for each state by considering the expected immediate rewards (r) and the estimated values of the successor states ($V_\pi(s')$). It is guaranteed to converge to the optimal value function and policy ([4]).

The Policy Iteration is another iterative method for solving MDPs, that finds the optimal policy by iteratively evaluating and improving the policy itself ([4], [7]).

Interior-point algorithms are a class of linear programming approaches used to solve MDPs. In this framework, MDPs are formulated as linear programs, and interior-point algorithms, such as the primal-dual interior-point method, are applied to find the optimal solution. These approaches represent MDPs as sets of linear equations and inequalities. One notable advantage of these methods is their ability to efficiently handle large-scale MDPs while offering strong theoretical properties ([37], [38]).

The Vavasis-Ye algorithm ([38], [39]) is a specialized algorithm designed for solving linear complementarity problems. It is also applied to handle MDPs, leveraging the structural characteristics of MDPs to efficiently find the optimal policy, by solving linear complementarity problems.

The choice of these methods often depends on the MDPs characteristics and the specific features of the optimization problem to be solved, including its size and structure. Each method has its advantages and may be preferred in different situations.

Recently the development of neural networks has extended the use of RL for solving MDPs, simultaneously improving efficiency. In particular, innovative ways of representing the input, through ad-hoc state encoders, have made it possible to manage very complex CO problems ([6], [40], [3], [1]).

These encoders adapt instances of CO problems to make them suitable for neural network processing. For example, the Pointer Network ([40]), encode the input in such a way to capture relevant features and reduce dimensionality.

Numerous RL-based approaches have been designed for addressing the BPP, with a majority of them focusing on solving the 2D and 3D versions of the problem

In particular, for the 3D BPP, one of the earliest RL-based approaches can be found in [41], where the authors apply a trained model on three categories of instances containing 8, 10 and 12 elements, demonstrating that the proposed method, which uses pointer networks as encoders of the positions of the elements, manages to overcome the reference heuristics obtaining results better than 5%.

In [42], the authors attempt to solve both 2D and 3D packing problems, using a recurrent neural network (RNN) to embed the states and then adopting an attention mechanism ([43]) to handle packing problems of different sizes. Results show that their model achieves a lower gap ratio in both 2D and 3D packing compared to existing heuristic methods and learning approaches.

In the context of the one-dimensional problem setting, only a limited number of papers have been published that explore the application of RL to the BPP. In [44], the authors consider a Proximal Policy Optimization approach [5] to train an agent and compare its performance with the BF and Sum of Squares (SS) heuristics [45] for the online version of the BPP. Their approach was tested not only on BPP, but also on Newsvendor and Vehicle Routing problems, showing that their RL policy outperforms or is competitive with baseline algorithms.

In [7], an RL agent is trained to mimic the BF heuristic. The authors demonstrate its ability to outperform the reference algorithm in all the considered instances. Unlike this last contribution, the approach proposed in this work aims to train an agent that receives a reward proportional to the feasibility and the quality of the solution it builds.

IV. THE PROPOSED APPROACH

One of the main challenges in solving CO problems with RL is the need of appropriately defining the corresponding MDPs. In the previous section, we have underlined that the main difficulty in constructing an MDP linked to a CO problem stems from the exponential growth of the model's dimension as the problem's size increases. Therefore, several approaches have been proposed in the scientific literature to address these challenges, aimed at approximating the search space.

In this work, we propose an alternative approach for building MDP models, well tailored for addressing CO problems. The related models, referred in the sequel as Markov Combinatorial Processes (MCPs), mantain the same structure and all the mathematical results as the MPDs; the difference lies in the construction method. Unlike MDPs, MCPs are not defined a-priori, based on the characteristics of the problem to be solved; instead, the agent itself learns how to construct them.

The proposed procedure can be viewed as a hybrid approach, integrating key aspects from the methods aimed at approximating MDPs and those that try to simulate them in order to handle the "curse of dimensionality". In this context, approximation involves the utilization of RNNs, through which the agent observes the state and determines the progression to subsequent states. Meanwhile, the simulation of the optimization problem is used to assign a value to the final states at each step.

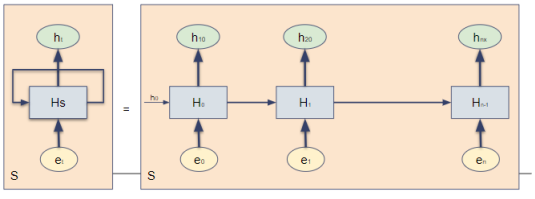


Figure 1. Graphical Representation of the Unrolling of the Markov Combinatorial Processes

The key idea draws inspiration from the existing works on RNNs ([40], [43]). More specifically, a single cell generates

an output h_t at time t, but, in addition to this output, the cell also maintains a state, H_s, which is recursively reused at time $t+1$ to compute the new output (see Fig. 1).

In other words, the outputs h_t depend on the state inherited from the previous step, and the current input e_t.

In our model, the H state of the RNNs becomes the set of the h states of the MDPs in a given iteration, the output is one of these h states, the one with the best QValue (see below), and the input is a vector containing aggregate information on the next element to be inserted (item in the case of BPP, customer and/or vehicle in the TSP, etc.).

Fig. 2 gives a graphical representation of the construction process in details. The MDPs states are identified with the h vectors inside the box wich represent the state H of the RNN at each step, while the input with the e vectors. Both are embedding vectors, meaning they contain pooled information that is then used by the agent to learn and construct the MCP appropriately.

In particular, the e-th vector is defined as follows:

$$e_t = [I_t \; E_t \; V_t \; Q_t],$$

where I_t are the values associated with the next item to be inserted (in the case of BBP it will be the weight of the item); E_t contains embedding information on the items (minimum, average and maximum values of the items and maximum capacity of the bin, for the BPP): empirically it has been seen that the information can accelerate the agent's training times, whereas it has no effect on the final value; V_t is a mask that contains information on the items already inserted: they are values that indicate how many times the i-th item is used (for example, in the case of BPP it is obviously 0 or 1); Q_t indicates the QValue of each i-th item, as we will see later, this value is used to define the overall value of the solutions found in each state.

At each step t, the MCP contains a certain number X of state-vectors h_x, where the x-th h vector contains the following information:

$$h_x = [e_t \; h_{t-1} \; h_{t-1} \oplus e_t \; r],$$

where e_t and h_t contains values from input and previous state, $h_{t-1} \oplus e_t$ are values calculated by aggregating the values of e_t with the values of the previous $h_t - 1$ state using the operator $\oplus$, a permutation invariant aggregation operator that can accept an arbitrary number of inputs (e.g., element-wise sum, mean, or max). In addition, r is the reward obtained at time t.

Thus, at each step, for each state-vector h_x, the agent decides whether to extend the state H with a new MDP state h, using the state-vector h_x and the vector e as observations. In this sense, the input e at each step t can be interpreted as the action e_{th}, which either extends or does not extend the single MDP state h_x.

In the positive case, the values are combined and new state-vectors are generated from h_x and the action-vector e_{th}. Each new state-vector h represents a new state of the MDPs.

For example in Fig. 3, at time 1 the state vector h_{10} is combined with the action-vector e_1, to generate the state-vector h_{20} (step 1), which, in this step, represents also the optimal solution, since it has the biggest QValue.

In step 2, starting from h_{10} and h_{20}, the state-vectors h_{30} and h_{34} are generated by the combination with e_2. The state vector h_{34} is crossed out in the figure, because the agent has decided to exclude it from the process. At the final step $(n-1)$, all possible state-vectors are generated, among which certain ones are considered optimal (highlighted in green in the figure), representing those with the highest QValue.

In general, it is clear that in each step, there can exist multiple optimal states.

A. MARKOV COMBINATORIAL PROCESSES FOR THE BPP

The proposed approach has been tailored to address the BPP in the following way. First of all, the problem is divided into subproblems, each of which deals with filling a single bin in the best possible way, i.e. minimizing the remaining free space after the items are inserted.

Fig.3 gives a graphical representation of the results obtained by using MCPs on a single subproblem S_x, with a single bin. The final solution of this step S_x contains the optimal solutions only for the current bin. In the considered example, we have only h_{nx2}, which is the state of MDPs with the best QValue in S_x.

The process continues in a similar way, building solutions for the other bins, until all the items have been considered (see Fig.4). Since, solving the single subproblems separately and combining the related solutions does not ensure optimality for the original problem, the vector of embeddings $[C \; I \; V \; Q]$ has been used, where C represents the capacity of the bins, I contains the weights of the items, V is the mask indicating whether items have been inserted (1 if already inserted, 0 otherwise), and Q represents the QValue of the items. These values are updated after each Sx step is completed and its best solution is selected.

For agent learning, we employ the Proximal Policy Optimization (PPO) learning algorithm ([5], [46]), with an actor-critic network architecture ([5], [47]). At each step, the agent receives observations from the environment derived from the embedding vectors e_t and h_x. In particular, during each training iteration, the agent observes each state h_x belonging to H_t using an attention mechanism ([40], [43]) and then it learns whether to expand the MDPs with the new state h, aiming to achieve the highest cumulative reward.

The pseudocode reported in Algorithm 1 shows how the agent is used to generate the MCPs for a single subproblem, i.e. for a single bin.

A bin is opened and it is set as the current bin. Then Algorithm 1 is executed and until an optimal solution is found for the current bin, i.e. the bin is fullfileed, or all items are processed, the vector e_t is created for each item, and then if it has not been used, the agent decides whether

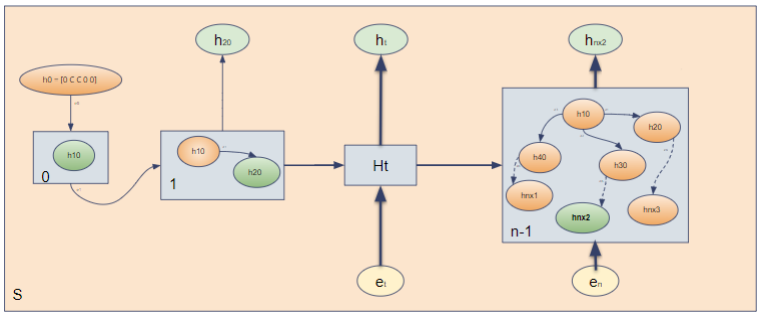


Figure 2. Details of the Unrolling Process

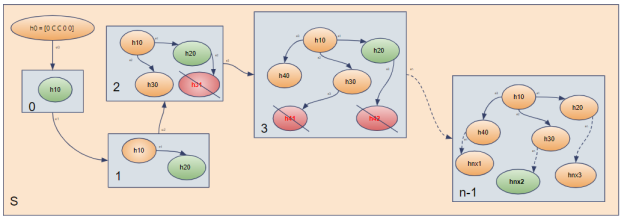


Figure 3. Markov Combinatorial Processes

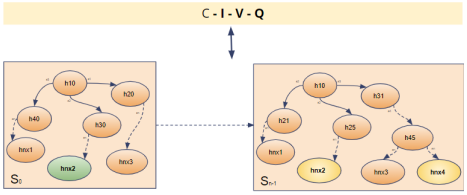


Figure 4. MCPs for BPP

Algorithm 1: Agent Heuristic

Input: Embedding Vector $[C, I, V, Q]$
Output: Best h_x states of H

while optimal $\neq True$ or all $items$ are processed **do**
  for *each item* **do**
    $e_t \leftarrow [\mathrm{I}_t\ E_t\ V_t\ Q_t]$;
    if $V[i] \neq 0$ **then**
      /* the item is already placed in a bin */
      continue with next item;
    for *each* h_x *in* H_t **do**
      if *AgentCreateNewMDPState*(h_x, e_t) == *True* **then**
        $h_{new} \leftarrow [\mathrm{e}_t\ h_{t-1}\ h_{t-1} \bigoplus e_t\ r]$;
        $H \leftarrow H + \langle h_{new} \rangle$;
    optimal $\leftarrow$ `waste == 0`;
return *Best* h_x *states of* H

or not to create a new state h_{new} starting from the state h_x. In positive case, h_{new} is added to the set H of states of current MDPs. The best h_x state is used to insert the items in the current bin and this process is repeated with a new opened bin until all the items are placed, i.e. all $V[i]$ are equal to 1.

V. COMPUTATIONAL RESULTS

The results in this work were obtained using the Colab shared platform, a web-based tool provided by Google ($https\ :\ //colab.research.google.com/$). This platform offers access to essential libraries and hardware resources for training neural networks, but it doesn't provide detailed execution time data due to shared hardware usage. Nevertheless, this is not a concern, as our goal is to demonstrate the agent's ability to effectively solve BPPs.

Table 1. Instances Characteristics

Instance	Opt Range	Instances Number	Bin capacity	Elements Number
Falkenauer U	[46, 52]	20	150	120
Falkenauer U	[99, 106]	20	150	250
Falkenauer U	[196, 207]	20	150	500
Falkenauer U	[393, 411]	20	150	1000
Falkenauer T	20	20	1000	60
Falkenauer T	40	20	1000	120
Falkenauer T	83	20	1000	249
Falkenauer T	167	20	1000	501

The computational experiments have been carried on a set of istances considered in [18] and [48]. In particular, there are 160 instances divided into two classes. The first one 'Falkenauer U'contains four groups with 120, 250, 500 and 1000 elements respectively, whose size is uniformly distributed between 20 and 100, while the capacity of the bins is 150. The second class 'Falkenauer T' consists of four groups containing 60, 120, 249 and 501 elements, whose sizes distributed between 25 and 50 and the bin capacity

equal to 100. These latter instances are considered the most challenging.

The main characteristics of the problem instances are given in Table 1, where the first column indicates the type of instance considered, the second the number of bins (min and max) in the optimal solutions, the third column the number of instances for each type, and the last column specifies the capacity and the number of elements to be placed.

Table 2. FU Results

Type	U120		U250		U500		U1000	
Instance Number	**Bin**	**GAP**	**Bin**	**GAP**	**Bin**	**GAP**	**Bin**	**Gap**
0	49	1	99	0	198	0	399	0
1	49	0	100	0	201	0	406	0
2	46	0	102	0	202	0	411	0
3	50	1	100	0	204	0	412	1
4	50	0	101	0	206	0	397	0
5	48	0	102	1	206	0	400	1
6	48	0	102	0	208	1	395	0
7	51	2	104	0	205	1	404	0
8	51	0	107	2	196	0	399	0
9	46	0	101	0	202	0	397	0
10	52	0	105	0	200	0	400	0
11	50	1	101	0	200	0	401	0
12	48	0	106	0	199	0	393	0
13	49	0	103	0	196	0	396	0
14	50	0	100	0	204	0	395	1
15	48	0	106	1	201	0	402	0
16	52	0	97	0	202	0	404	0
17	53	1	100	0	198	0	404	0
18	49	0	100	0	202	3	399	3
19	50	1	102	0	2	0	400	0

Table 3. FT Results

Type	T60		T120		T249		T501	
Instance Number	**Bin**	**GAP**	**Bin**	**GAP**	**Bin**	**GAP**	**Bin**	**Gap**
0	21	1	41	1	84	1	168	1
1	21	1	41	1	84	1	168	1
2	20	0	41	1	84	1	169	2
3	21	1	41	1	84	1	168	1
4	21	1	41	1	84	1	169	2
5	21	1	41	1	84	1	168	1
6	21	1	41	1	84	1	168	1
7	21	1	41	1	84	1	168	1
8	21	1	41	1	84	1	168	1
9	21	1	41	1	84	1	168	1
10	21	1	41	1	84	1	168	1
11	21	1	41	1	84	1	168	1
12	21	1	41	1	84	1	168	1
13	21	1	41	1	84	1	169	2
14	21	1	41	1	84	1	168	1
15	21	1	41	1	84	1	169	2
16	21	1	41	1	84	1	169	2
17	21	1	41	1	84	1	168	1
18	21	1	41	1	84	1	168	1
19	21	1	41	1	84	1	169	2

A first phase of the testing phase has been devoted to the evaluation of the effectiveness of the learning algorithm, in terms of training speed and expected reward. To this aim, we compare the rewards attained by an untrained agent (i.e., a random agent) with those obtained by the agent trained using the PPO approach. In Fig. 5, the reward obtained on the first instance of the Falkenauer U problem class, with 120 items is depicted. Similar results have been collected on the other instances.

As expected, Fig. 5 underlines that when a random agent is used, it is not possible to have a high average reward nor to improve it over time.

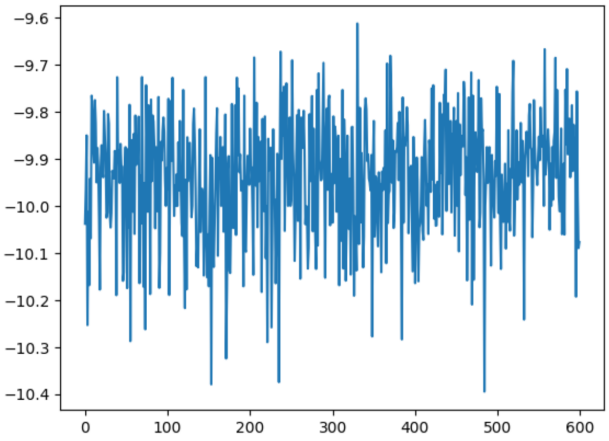


Figure 5. Rewards obtained by using a random agent on the first instance of tje set Falkenauer U.

Using the training algorithm instead (see Fig. 6), the agent learns until it obtains the maximum expected reward. In particular, each time the agent performs a wrong action, it receives a negative reward if it accepts a Markov state that is not feasible for the problem (e.g. for BPP if it exceeds the available space) and the episode ends. Otherwise, the agent receives a reward proportional to the quality of the built solution (i.e. the filling grade of the bin).

The use of this agent in the proposed algorithm allows finding optimal or close to optimal solutions for the considered BPP instances.

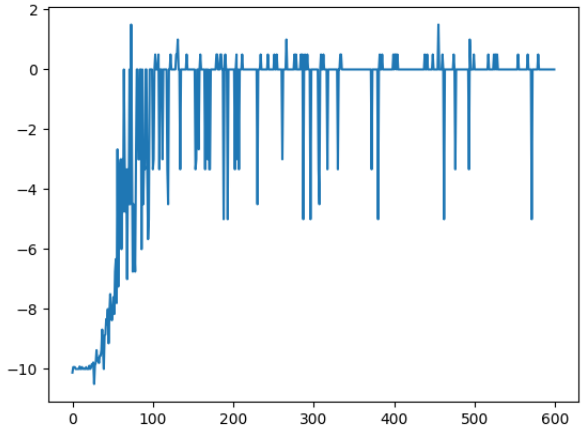


Figure 6. Training Process of the agent

Tables 2 and 3 provide detailed results for each instance type, including the number of bins required and the gap compared to the number of bins used in the known optimal

solution. The first table refers to instances of type U, while the second one to those of type T.

As observed in the U-type instances, the approach achieves near-optimal solutions for most cases. In instances where it falls short of finding the optimal solution, the discrepancy in the number of bins compared to the known best solution is typically only one bin. However, in the case of instance 18, containing 500 elements, the deviation amounts to 3 bins.

In the case of instances of type T, the most difficult ones, the algorithm is unable to find the optimal solution, except in instance 2 with 60 elements. However, the GAP with respect to the best known solution is only one bin.

Table 4. Results for each set of instances

Sets	Bin C	Opt	BFD	RLC	MCPH	Delta Ref	Delta Opt
U120	150	49.15	49.8	50.0	49.50	-0.5	0.3
U250	150	101.7	103.1	103.15	101.9	-1.2	0.2
U500	150	201.2	203.9	203.4	201.3	**-2.1**	0.1
U1000	150	400.6	405.4	403.9	400.7	**-3.2**	0.2
T60	100	20.0	23.2	21.0	20.95	**-0.05**	0.95
T120	100	40.0	45.8	41.1	41.00	**-0.15**	1.0
T249	100	83.0	95.0	84.8	84.0	**-0.8**	1.0
T501	100	167.0	190.1	169.4	168.3	**-1.05**	1.3
	AVG	**132.8**	**139.5**	**134.6**	**133.5**	**-1.1**	**0.6**

Table 4 shows the results of the proposed approach, referred to as MCPH, compared with the Best Fit Decreasing (BFD) algorithm and the RLC approach ([7]), also based on RL, but in which the agent did not directly learn to develop a solution, but rather it is trained to mimic a given heuristic.

The BFD, RLC and MCPH columns report the number of bins used by the corresponding algorithm, while Delta Ref and Delta Opt columns give the difference in the number of bins used by the MCPH algorithm compared to the reference heuristic (RLC) and to the optimal number of bins (Opt column).

From the results reported in Table 4 it is evident that the MCPH algorithm outperforms both BFD and RLC in terms of the number of bins used on all the instances. The average number is 133.5 bins compared to 134.6 bins for RLC and 139.5 for BFD. The delta compared to RLC is -1.1 bins, and compared to the optimal solution, it is 0.6 bins higher.

We can conclude that the presented approach is valid because it provides results very close to the optimal solution. It also improves upon the baseline algorithm from the literature and outperforms the RLC algorithm, which is currently the only available RL-based algorithm for BPPs.

VI. CONCLUSION

This paper introduces an innovative approach, that combines Reinforcement Learning (RL) with a novel method of constructing Markov Decision Processes (MDPs) to solve combinatorial optimization problems. The approach involves creating new Markov Combinatorial Processes (MCPs) by training an agent, enhancing its effectiveness in tackling these optimization challenges

We then applied this model to an NP-hard problem (i.e., the Bin Packing Problem) often used as a comparison baseline in the literature and we analyzed the results obtained in depth.

The computational results are very encouraging. They demostrate the effectiveness and validity of the approach. Indeed, it was possible to ascertain the ability of the algorithm to arrive at optimal solutions for different instances of the BPP, surpassing the RL solutions existing in the literature.

Nevertheless, this achievement represents only the starting point to explore the opportunities arising from the integration of RL into solution approaches to handle combinatorial optimization problems.

A crucial aspect in future developments concerns the improvement of the embedding vector, which represents one of the central elements of the algorithm. Optimizing the learning capacity of the algorithm, through the development of efficient and informative representations, is fundamental to obtain more accurate solutions and to improve the overall computational efficiency.

References

[1] H. Dai, E. B. Khalil, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," arXiv preprint arXiv:1704.01665, 2017.

[2] A. Nowak, S. Villar, A. S. Bandeira, and J. Bruna, "A note on learning algorithms for quadratic assignment with graph neural networks," arXiv preprint arXiv:1706.07450, 2017.

[3] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev, "Reinforcement learning for combinatorial optimization: A survey," Computers & Operations Research, vol. 134, pp. 539–548, 2021.

[4] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction. MIT press, 2018.

[5] J. Schulman, F. W. P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," arXiv preprint arXiv:1707.06347, 2017.

[6] Y. Bengio, A. Lodi, and A. Prouvost, "Machine learning for combinatorial optimization: A methodological tour d'horizon," European Journal of Operational Research, 2021.

[7] F. Guerriero and F. P. Saccomanno, "A machine learning approach for the bin packing problem," in Proceeding of 12th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications. 7-9 September 2023, 2023.

[8] Z. Li, Q. Chen, and V. Koltun, "Combinatorial optimization with graph convolutional networks and guided tree search," Advances in Neural Information Processing Systems, pp. 539–548, 2018.

[9] S. Basu, Design methods and analysis of algorithms. PHI Learning Pvt. Ltd., 2013.

[10] S. Martello and P. Toth, "Knapsack problems: Algorithms and computer implementations," Wiley, 1990.

[11] P. Schaus, J. Regin, R. V. Schaeren, W. Dullaert, and B. Raa, "Cardinality reasoning for bin-packing constraint: Application to a tank allocation problem," Lecture Notes in Computer Science, 2012.

[12] L. Wei, Z. Luo, R. Baldacci, and A. Lim, "A new branch-and-price-and-cut algorithm for one-dimensional bin-packing problems," INFORMS Journal on Computing, 2019.

[13] M. Delorme, M. Iori, and S. Martello, "Bin packing and cutting stock problems: Mathematical models and exact algorithms," European Journal of Operational Research, vol. 255, pp. 1–20, 2016.

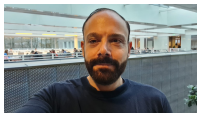
[14] S. Martello and D. Vigo, "Exact solution of the two-dimensional finite bin packing problem," Management Science, 1997.

[15] R. E. Korf, "A new algorithm for optimal bin packing," in Eighteenth National Conference on Artificial Intelligence. USA: American Association for Artificial Intelligence, 2002, p. 731–736.

[16] S. E. L. and R. E. Korf, "Improved bin completion for optimal bin packing and number partitioning." Proceedings of the Twenty-Third international joint conference on Artificial Intelligence, 2013.

[17] M. Delorme, M. Iori, and S. Martello, "Bin packing and cutting stock problems: Mathematical models and exact algorithms," European Journal of Operational Research, vol. 255, no. 1, pp. 1–20, 2016.
[18] E. Falkenauer, "A New Representation and Operators for Genetic Algorithms Applied to Grouping Problems," Evolutionary Computation, vol. 2, no. 2, pp. 123–144, 06 1994.
[19] F. Brandao and J. Pedroso, "Bin packing and related problems: General arc-flow formulation with graph compression," Computers & Operations Research, 2016.
[20] F. Guerriero and F. P. Saccomanno, "A hierarchical hyper-heuristic for the bin packing problem," Soft Comput., vol. 27, no. 18, p. 12997–13010, may 2022.
[21] D. S. Johnson, A. J. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham, "Worst-case performance bounds for simple one-dimensional packing algorithms," SIAM J. Comput., vol. 3, no. 4, pp. 299–325, 1974.
[22] D. S. Johnson, "Fast algorithms for bin packing," SIAM J. Comput., vol. 8, no. 3, p. 272–314, 1974.
[23] E. G. Coffman Jr., J. Csirik, G. Galambos, S. Martello, and D. Vigo, Bin Packing Approximation Algorithms: Survey and Classification. New York, NY: Springer New York, 2013, pp. 455–531. [Online]. Available: https://doi.org/10.1007/978-1-4419-7997-1_35
[24] C. Munien and A. E. Ezugwu, "Metaheuristic algorithms for one-dimensional bin-packing problems: A survey of recent advances and applications," Journal of Intelligent Systems, vol. 30, no. 1, pp. 636–663, 2021.
[25] J. Filar and K. Vrieze, Competitive Markov Decision Processes. Berlin, Heidelberg: Springer-Verlag, 1996.
[26] Y. Ye, "A new complexity result on solving the markov decision problem," Mathematics of Operations Research, vol. 30, no. 3, pp. 733–749, 2005. [Online]. Available: http://www.jstor.org/stable/25151680
[27] V. Gabillon, M. Ghavamzadeh, and B. Scherrer, "Approximate dynamic programming finally performs well in the game of tetris," in Advances in Neural Information Processing Systems, C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, Eds., vol. 26. Curran Associates, Inc., 2013.
[28] D. P. Bertsekas, "Feature-based aggregation and deep reinforcement learning: A survey and some new implementations," 2018.
[29] W. Powell, Approximate Dynamic Programming: Solving the Curses of Dimensionality: Second Edition. United States: Wiley-Blackwell, Sep. 2011, publisher Copyright: © 2011 by John Wiley Sons, Inc. All rights reserved.
[30] M. Hutter, "Extreme state aggregation beyond markov decision processes," Theoretical Computer Science, vol. 650, pp. 73–91, 2016, algorithmic Learning Theory.
[31] R. Meshram and K. Kaza, "Simulation based algorithms for markov decision processes and multi-action restless bandits," 2020.
[32] H. S. Chang, J. Hu, M. C. Fu, and S. I. Marcus, Simulation-Based Algorithms for Markov Decision Processes, 2nd ed. Springer Publishing Company, Incorporated, 2013.
[33] D. Silver, C. J. M. Aja Huang, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," Nature, no. 529, p. 484–489, 2016.
[34] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. P. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," ArXiv, vol. abs/1712.01815, 2017.
[35] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," Science, vol. 362, no. 6419, pp. 1140–1144, 2018.
[36] M. L. Littman, T. L. Dean, and L. P. Kaelbling, "On the complexity of solving markov decision problems," CoRR, vol. abs/1302.4971, 2013. [Online]. Available: http://arxiv.org/abs/1302.4971
[37] M. G. Robert Givan, Thomas Dean, "Equivalence notions and model minimization in markov decision processes," Artificial Intelligence, vol. 147, no. 1, pp. 163–223, 2003.
[38] Y. Mansour and S. Singh, "On the complexity of policy iteration," 2013.
[39] R. D. C. Monteiro and T. Tsuchiya, "A variant of the vavasis–ye layered-step interior-point algorithm for linear programming," SIAM Journal on Optimization, vol. 13, no. 4, pp. 1054–1079, 2003.
[40] O.Vinyals, M.Fortunato, and N.Jaitly, "Pointer networks," Advances in Neural Information Processing Systems, p. 2692–2700, 2015.
[41] H. Hu, X. Zhang, X. Yan, L. Wang, and Y. Xu, "Solving a new 3d bin packing problem with deep reinforcement learning method," 2017.
[42] D. Li, Z. Gu, Y. Wang, C. Ren, and F. C. M. Lau, "One model packs thousands of items with recurrent conditional query learning," CoRR, vol. abs/2111.06726, 2021. [Online]. Available: https://arxiv.org/abs/2111.06726
[43] W. Kool and M. Welling, "Attention solves your tsp," arXiv preprint arXiv:1803.08475, 2018.
[44] B. Balaji, J. Bell-Masterson, E. Bilgin, A. Damianou, P. M. Garcia, A. Jain, A. Luo, A. Maggiar, B. Narayanaswamy, and C. Ye, "Reinforcement learning benchmarks for online stochastic optimization problems," 2019. [Online]. Available: https://openreview.net/forum?id=HklKjpeCiE
[45] M. Bender, B. Bradley, G. Jagannathan, and K. Pillaipakkamnatt, "Sum-of-squares heuristics for bin packing and memory allocation," ACM Journal of Experimental Algorithmics, vol. 12, 01 2007.
[46] J. B. Diederik P. Kingma, "Adam: A method for stochastic optimization," 3rd International Conference for Learning Representations, 2015.
[47] R. S. Sutton and A. G. Barto, Reinforcement learning: An introduction (2nd Edition). MIT press Cambridge, 2017.
[48] E. Falkenauer, "A hybrid grouping genetic algorithm for bin packing," Journal of Heuristics, vol. 2, no. 1, pp. 5–30, Jun 1996.

FRANCESCA GUERRIERO is Full Professor of Operations Research at University of Calabria. She is currently the Dean of the Department of Mechanical, Energy and Management Engineering, University of Calabria and she is the Vice-President of the Italian Operations Research Society (AIRO). Her main research interests are in the area of network optimization, logistics and distribution, revenue management, project management, optimization and big data. She is co-author of more than 150 papers published in prestigious journal. She has been and is member of the scientific committee of several International Conferences and she is a member of the editorial board of scientific journals. She supervises master and PhD research projects. She has been and is involved in several research projects as either project leader or member of the research unit.



FRANCESCO PAOLO SACCOMANNO is currently a PhD candidate in Computer Engineering at the University of Calabria and holds a Master of Science in Electrical Engineering from the University of Rome 'La Sapienza'. His primary research focuses on the development of reinforcement learning methodologies to address combinatorial optimization problems.
Additionally, he has keen interest in parallel algorithm development for GPUs, quantum computing, and machine learning applications.