GymPN: a Python Library for Automated Decision-Making in Process Management Systems

Riccardo Lo Bianco[1,*], Willem van Jaarsveld[1] and Remco Dijkman[1]

[1]Eindhoven University of Technology, Eindhoven, The Netherlands

Abstract

GymPN is a Python library for modeling and solving business process decision-making problems using Deep Reinforcement Learning (DRL). Built on top of SimPN, it integrates Petri Net-based simulation with DRL training pipelines, enabling users to define, train, and evaluate decision policies with minimal configuration. GymPN supports both heuristic and learning-based approaches, and includes features for modeling partially observable processes as well as decisions at different steps of the process. The library is open-source, easy to use, and has been validated in a wide range of business process scenarios.

Keywords

Petri Net, Deep Reinforcement Learning, Optimization

1. Introduction

Petri Nets (PNs) represent an established mathematical formulation for the simulation of business processes [1]. In recent years, an extension of the PN formalism, namely, Action-Evolution Petri Nets (A-E PNs), was proposed to enable modeling and solving optimization problems in business processes [2]. As an example, let us consider the problem in Figure 1, which represents a simple dynamic task assignment problem.

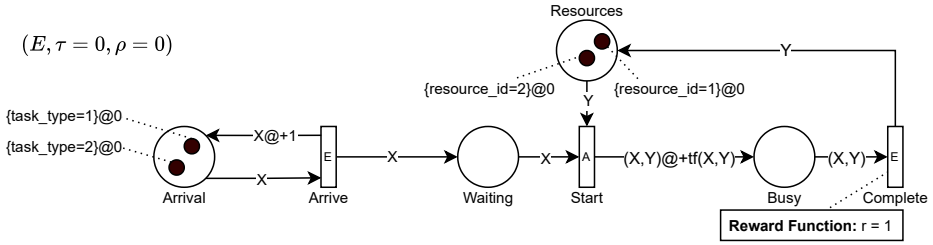


Figure 1: A simple dynamic task assignment policy expressed through A-E PN.

Two cases, each composed of a single task, enter the system at every time unit. Two resources are present, with different proficiencies on each task type. This is modeled through the *tf* function so that resources whose *resource_id* is the same as the *task_id* they are assigned to take 1 time unit to complete the task, 2 time units otherwise. A reward of 1 is produced every time a case is completed. Thus the objective is to minimize the cycle time of cases (equivalent to maximizing the reward). The goal of the optimization is to find a policy function that maps each observation (a given marking of the PN) to a combination of task and resource to maximize the reward over an episode.

The A-E PN formalism was conceived to combine elements of simulation with elements of sequential decision-making, with a particular focus on Deep Reinforcement Learning (DRL). However, the theoretical foundations of A-E PN have not yet been implemented in a well-structured, easy-to-use

Proceedings of the Best BPM Dissertation Award, Doctoral Consortium, and Demonstrations & Resources Forum co-located with 23rd International Conference on Business Process Management (BPM 2025), Seville, Spain, August 31st to September 5th, 2025.

*Corresponding author.

r.lo.bianco@tue.nl (R. Lo Bianco); w.l.v.jaarsveld@tue.nl (W. van Jaarsveld); r.m.dijkman@tue.nl (R. Dijkman)

0000-0002-0898-3959 (R. Lo Bianco); 0000-0003-3620-4067 (W. van Jaarsveld); 0000-0003-4083-0036 (R. Dijkman)

software package. GymPN provides such a tool in the form of a Python package. Based on the recently published SimPN library [3], GymPN maintains the same design principles as SimPN, integrating all elements of A-E PNs in a seamless fashion. This tight integration allows for using most of SimPN's features, such as reporting, support for higher-level modeling languages, and visualizations. At the same time, it greatly simplifies the adoption for SimPN users.

The remainder of this work is structured as follows: section 2 discusses alternative optimization packages, presenting the advantages that GymPN offers; section 3 presents the modeling, simulation and optimization features of the library; section 4 discusses the maturity of the library.

2. Related work

Several DRL-based approaches for automated decision-making in process management systems have been proposed [4, 5]. However, all these approaches are tailored to a single problem instance, and they do not offer utilities to model new problems. By contrast, GymPN integrates the theory of A-E PN [2, 6] with the formal syntax of SimPN [3]. Moreover, GymPN extends the capabilities of A-E PN, adding support for partial observability and multi-decision processes [7].

GymPN can be considered a software package for sequential decision-making. However, compared to other publicly available Python packages, like Gymnasium [8] (internally used in GymPN) or OR-Gym [9], it offers unique utilities to model the desired problem using business process notation. Moreover, it does not require the user to define optimization-specific elements (except for the objective), allowing one to focus on the problem definition instead.

3. Features

GymPN uses the same syntax as SimPN, integrating the necessary elements to define a business process decision-making problem, i.e., what decisions need to be made (action transitions), and a measure of goodness of such decisions (rewards). To display GymPN's capabilities, we model a simple task assignment problem. We then proceed by showing how to train a DRL agent and compare it with a heuristic policy.

3.1. Modeling

In this section, we implement the problem presented in Figure 1 in GymPN.

```
agency = GymProblem()
arrival = agency.add_var("arrival", var_attributes=['task_type'])
waiting = agency.add_var("waiting", var_attributes=['task_type'])
busy = agency.add_var("busy", var_attributes=['task_type', 'resource_id'])
arrival.put({'task_type': 0})
arrival.put({'task_type': 1})
resources = agency.add_var("resources", var_attributes=['resource_id'])
resources.put({'resource_id': 0})
resources.put({'resource_id': 1})

def arrive(a):
    return [SimToken(a, delay=1), SimToken(a)]
agency.add_event([arrival], [arrival, waiting], arrive)

def start(c, r):
    if c['task_type'] == r['resource_id']:
        return [SimToken((c, r), delay=1)]
    else:
        return [SimToken((c, r), delay=2)]
agency.add_action([waiting, resources], [busy], behavior=start, name="start")
```

```
def complete(b):
    return [SimToken(b[1])]
agency.add_event([busy], [resources], complete, name='complete', reward_function=lambda x: 1)

agency.training_run(length=10, args_dict=default_args)
```

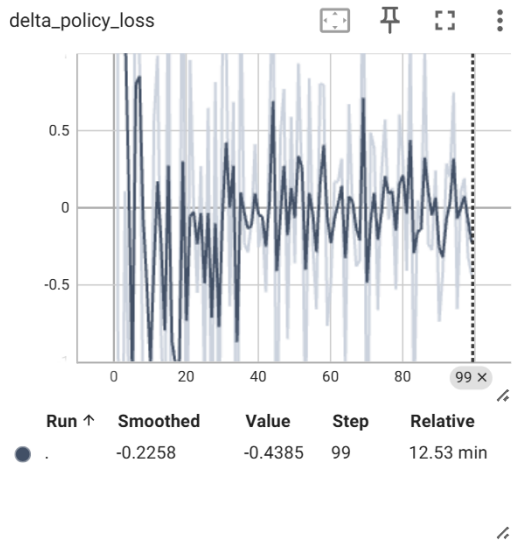
The main class in GymPN is `GymProblem` (an extension of SimPN's `SimProblem`), which can be seen as an initially empty Petri Net. Places are added to the PN using the `add_var` method. The initial marking is defined by inserting tokens into the places through the `put` function. GymPN assumes that the attributes of the tokens are expressed as dictionaries, and it is necessary to specify the attributes of the tokens in every place via the `var_attributes` parameter. This is necessary to ensure that the observations for DRL are derived correctly from the marking. Evolution transitions (the classical non-deterministic transitions of Timed Colored Petri Nets), such as the arrival transition, are added to the problem via the `add_event` function, the same way as they are handled in SimPN. Action transitions (transitions for which a policy chooses the tokens to be used for firing) are defined in a similar fashion via the `add_action` function. For any transition, it is possible to specify a reward function that is called every time the transition is fired. In the example, only transition `done` produces a reward of 1. This means that, over 10 time units, the maximum cumulative reward is 20.

3.2. Training Policies

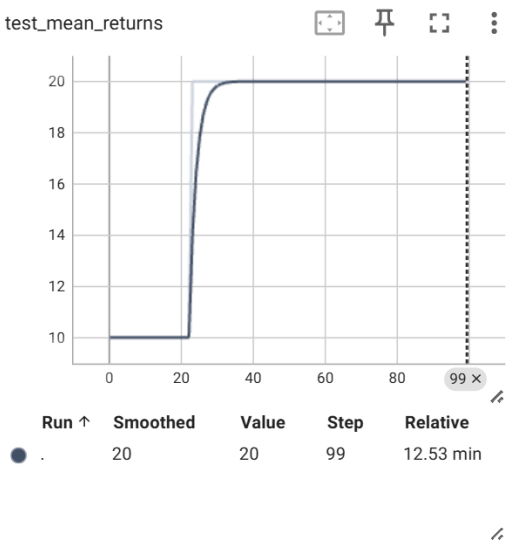
To train a DRL policy on the provided example, it is sufficient to run the `training_run` function:

```
agency.training_run(length=10)
```

The `training_run` function runs a given number of training epochs (by default 100) on the Proximal Policy Optimization (PPO) algorithm. The default hyperparameters and other configuration variables can be modified by passing a dictionary to the `args_dict` parameter in the `training_run` function. By default, running the `training_run` function will open a TensorBoard dashboard that displays the advancement of the training process. In Figure 2, we report two representative graphs from the dashboard.



(a) A decreasing policy loss indicates that the learning is progressing.



(b) The (deterministic) policy is tested during training, showing convergence.

Figure 2: Two of the graphs produced by the `training_run` function.

3.3. Testing Policies

The `GymSolver` class is responsible for mapping a PN marking to a suitable observation and returning an action (i.e., what tokens to use to fire an action transition) according to a policy function. Having defined the solver, the `testing_run` function is used to simulate an episode using the desired DRL policy, by retrieving the weights of the trained neural network.

```
solver = GymSolver(weights_path='./weights.pth', metadata=agency.make_metadata())
res = agency.testing_run(length=10, solver=solver)
```

In the example, the DRL policy produces a reward of 20 over 10 time units, which is the maximum. This is in line with the results observed during the training phase, as reported in Figure 2b.

A specialized solver, namely, `HeuristicSolver`, is available to implement heuristic policies. In the presented example, we can develop a simple heuristic policy that always assigns the resource with a given `resource_id` to a task with the same `task_type`. The heuristic policy function takes two parameters representing, respectively, the observable parts of the Petri Nets and the list of all possible combinations of tokens that can be used to fire an action transition.

```
def perfect_heuristic(observable_net, tokens_comb):
    for k, el in tokens_comb.items():
        for binding in el:
            task = binding[0][1].value
            resource = binding[1][1].value
            if task['task_type'] == resource['resource_id']:
                return {k: binding}

solver = HeuristicSolver(perfect_heuristic)
res = agency.testing_run(length=10, solver=solver)
```

It is easy to see that this is the optimal policy, and it also produces a reward of 20 over 10 time units.

Similarly, we can use the `RandomSolver` class to implement a random assignment policy.

```
solver = RandomSolver()
res = agency.testing_run(length=10, solver=solver)
```

The random assignment policy is clearly suboptimal, producing an average reward of 15 over 10 time units.

For logging, the `testing_run` function is compatible with the SimPN `Reporter` class. Moreover, the library provides an interactive graphical visualization of the behavior of the policy, as shown in Figure 3.

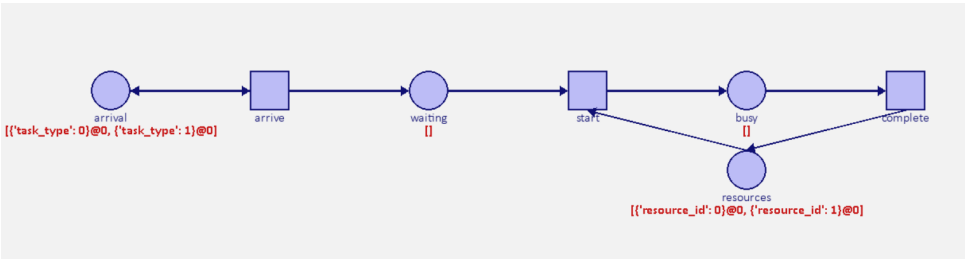


Figure 3: Interactive visualization of the behavior of the policy.

4. Tool Maturity

GymPN was made available as part of a publication [7] that introduced support for multiple action types and partial observability in the A-E PN framework. This publication included a set of eight example problems that cover basic business process management patterns. In the documentation, more complex

examples and features are discussed, and a method to automatically derive GymPN simulations from real-world process logs is being developed. At the time of writing, the repository is open-source and actively in development.

Acknowledgments. This project was developed as part of the AI (Artificial Intelligence) Planner of The Future research program, an initiative of the European Supply Chain Forum (ESCF).

Declaration on Generative AI. During the preparation of this work, the author(s) used Grammarly and Microsoft Copilot in order to: Grammar and spelling check. After using these tool(s)/service(s), the author(s) reviewed and edited the content as needed and take(s) full responsibility for the publication's content.

References

[1] K. Jensen, L. M. Kristensen, L. Wells, Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems, International Journal on Software Tools for Technology Transfer 9 (2007) 213–254. URL: https://link.springer.com/10.1007/s10009-007-0038-x. doi:10.1007/s10009-007-0038-x.

[2] R. Lo Bianco, R. Dijkman, W. Nuijten, W. Van Jaarsveld, Action-evolution Petri Nets: A Framework for Modeling and Solving Dynamic Task Assignment Problems, volume 14159, Springer Nature Switzerland, Cham, 2023, pp. 216–231. doi:10.1007/978-3-031-41620-0_13, series Title: Lecture Notes in Computer Science.

[3] R. Dijkman, Simpn: A python library for modeling and simulating timed, colored petri nets, in: BPM-D 2024, CEUR Workshop Proceedings, CEUR-WS.org, 2024, pp. 71–75. Publisher Copyright: © 2024 Copyright for this paper by its authors.; Best Dissertation Award, Doctoral Consortium, and Demonstration and Resources Forum at 22nd International Conference on Business Process Management, BPM-D 2024 ; Conference date: 01-09-2024 Through 06-09-2024.

[4] C. Shyalika, T. Silva, A. Karunananda, Reinforcement learning in Dynamic Task Scheduling: A Review, SN Computer Science 1 (2020). Publisher: Springer Science and Business Media LLC.

[5] R. Lo Bianco, W. v. Jaarsveld, J. Middelhuis, L. Begnardi, R. Dijkman, Automated decision-making for dynamic task assignment at scale, 2025. doi:10.48550/arXiv.2504.19933, arXiv:2504.19933 [cs].

[6] R. Lo Bianco, R. Dijkman, W. Nuijten, W. Van Jaarsveld, A universal Approach to Feature Representation in Dynamic Task Assignment Problems, in: A. Marrella, M. Resinas, M. Jans, M. Rosemann (Eds.), Business process Management Forum, volume 526, Springer Nature Switzerland, Cham, 2024, pp. 197–213. doi:10.1007/978-3-031-70418-5_12, series Title: Lecture Notes in Business Information Processing.

[7] R. L. Bianco, W. van Jaarsveld, R. Dijkman, Gympn: A library for decision-making in process management systems, 2025. URL: https://arxiv.org/abs/2506.20404. arXiv:2506.20404.

[8] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. D. Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J. Tai, H. Tan, O. G. Younis, Gymnasium: a Standard Interface for Reinforcement Learning Environments, 2024. ArXiv:2407.17032 [cs].

[9] C. D. Hubbs, H. D. Perez, O. Sarwar, N. V. Sahinidis, I. E. Grossmann, J. M. Wassick, OR-Gym: A Reinforcement Learning Library for Operations Research Problems, 2020. ArXiv:2008.06319 [cs].

A. Online Resources

- GymPN is available on GitHub at the address https://github.com/bpogroup/gympn
- An introductory video is available on YouTube at the address https://youtu.be/2N7DW67NxqI