Reinforcement Learning for Dynamic Memory Allocation

Arnab Mandal, Suryansh Rohil
https://github.com/Eros483/rl_memory_
https://rl-mem-alloc-am-sr.streamlit.app/

ABSTRACT

In recent years, reinforcement learning (RL) has gained popularity and has been applied to a wide range of tasks. One such popular domain where RL has been effective is resource management problems in systems. We look to extend existing work done on leveraging machine learning, to solve existing memory allocation problems.

I. INTRODUCTION

Reinforcement learning is a form of machine learning wherein an agent learns to make decisions in an environment to maximise cumulative rewards.

In the context of memory allocation, the decision to be made is the choice of where to allocate memory with respect to existing memory blocks (created due to the deallocation of previously stored processes).

The environment includes the memory stored, the processes that are to be allocated memory, and the potential for fragmentation.

Rewards are given to the agent, incentivising successful allocations, penalising unsuccessful allocations, and discouraging allocation that leads to external fragmentation.

There are four primary pre-existing memory allocation methods, i.e. first-fit, best-fit, second-fit and worse-fit.

First-fit involves allocating the process to the first memory block that can accommodate the memory requirements of the process.

Second-fit is the same as first-fit but instead, allocation happens to the second memory block that can accommodate the memory requirements.

Best-fit involves allocating the process to a memory block so that any memory left over in the same block is minimised. This typically reduces the risk of external fragmentation.

Worst-fit involves doing the opposite as the best-fit algorithm, with processes being allocated to a memory block such that the leftover memory is maximised. This is a round-about manner, which also works at reducing external fragmentation.

The objective was to train a reinforcement learning model that would reduce fragmentation to a much greater degree compared to the existing algorithms, and thus be a far more suitable alternative.

II. METHODS AND MATERIAL

Using Python as the programming language of our choice, we trained a reinforcement model based on processes of random sizes. We represented memory as a numpy array of 0s, with 0s being replaced by the process ID that is allocated to that block of memory.

We leveraged a DQN agent, as was suggested in previous publications, working inside a QNN and training it over 1000 episodes.

We then customised a memory filled with 'holes' so as to best stimulate a real-life scenario.

This memory was then passed to the existing 4 algorithms and our trained agent.

User-defined processes would be subjected to each algorithm, and after allocation, the memory was once again rendered to observe differences between each algorithm.

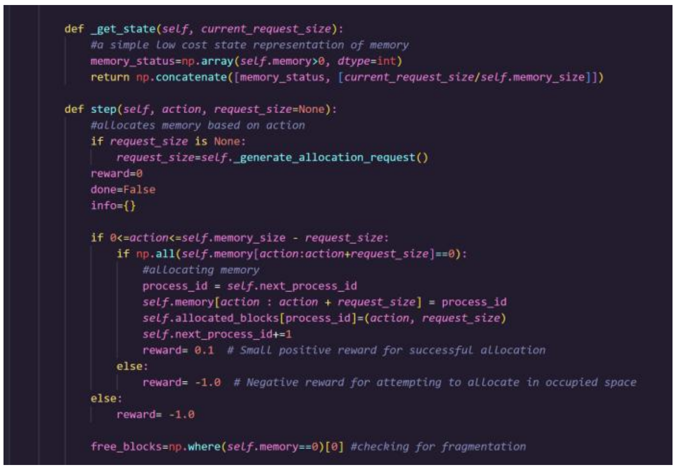
A definition to determine the degree of external fragmentation was created, and the same was measured for each algorithm.

The results for all of the above were uploaded on the above-mentioned website, creating a simple front-end leveraging Streamlit.

III. RESULTS AND DISCUSSION

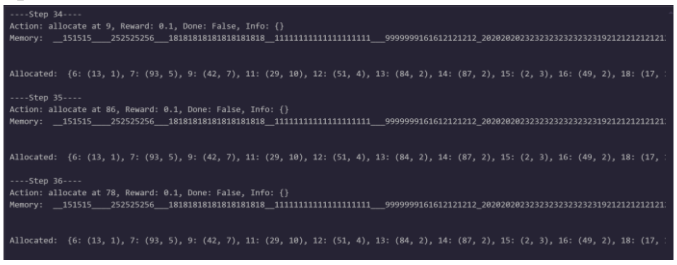
Existing libraries such as gym, gymnasium and MP3 proved to be insufficient for this project.

A user-defined environment class proved to be necessary, with several methods such as reset, step, render and deallocation functions being necessary to initialise.



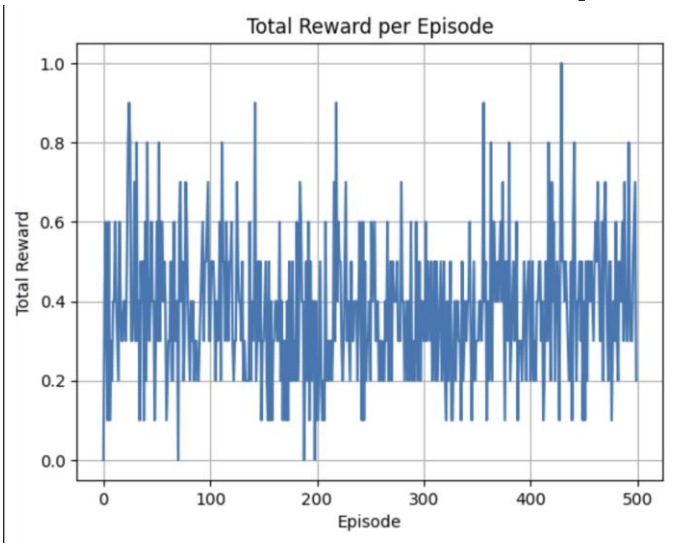
```python
def _get_state(self, current_request_size):
    #a simple low cost state representation of memory
    memory_status=np.array(self.memory>0, dtype=int)
    return np.concatenate([memory_status, [current_request_size/self.memory_size]])

def step(self, action, request_size=None):
    #allocates memory based on action
    if request_size is None:
        request_size=self._generate_allocation_request()
    reward=0
    done=False
    info={}

    if 0<=action<=self.memory_size - request_size:
        if np.all(self.memory[action:action+request_size]==0):
            #allocating memory
            process_id = self.next_process_id
            self.memory[action : action + request_size] = process_id
            self.allocated_blocks[process_id]=(action, request_size)
            self.next_process_id+=1
            reward= 0.1  # Small positive reward for successful allocation
        else:
            reward= -1.0  # Negative reward for attempting to allocate in occupied space
    else:
        reward= -1.0

    free_blocks=np.where(self.memory==0)[0] #checking for fragmentation
```

Before training an agent leveraging QNNs, the existing environment was first tested with several random sets of processes, utilising the created render function to observe memory allocation over several hundred episodes.

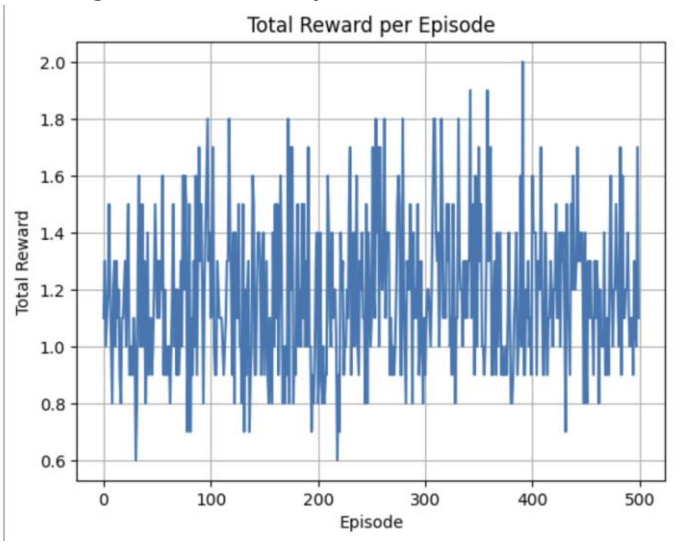


```
----Step 34----
Action: allocate at 9, Reward: 0.1, Done: False, Info: {}
Memory:  __151515____252525256___1818181818181818181818__11111111111111111111___9999999161612121212_2020202023232323232323231921212121212121


Allocated:  {6: (13, 1), 7: (93, 5), 9: (42, 7), 11: (29, 10), 12: (51, 4), 13: (84, 2), 14: (87, 2), 15: (2, 3), 16: (49, 2), 18: (17, 

----Step 35----
Action: allocate at 86, Reward: 0.1, Done: False, Info: {}
Memory:  __151515____252525256___1818181818181818181818__11111111111111111111___9999999161612121212_2020202023232323232323231921212121212121


Allocated:  {6: (13, 1), 7: (93, 5), 9: (42, 7), 11: (29, 10), 12: (51, 4), 13: (84, 2), 14: (87, 2), 15: (2, 3), 16: (49, 2), 18: (17, 

----Step 36----
Action: allocate at 78, Reward: 0.1, Done: False, Info: {}
Memory:  __151515____252525256___1818181818181818181818__11111111111111111111___9999999161612121212_2020202023232323232323231921212121212121


Allocated:  {6: (13, 1), 7: (93, 5), 9: (42, 7), 11: (29, 10), 12: (51, 4), 13: (84, 2), 14: (87, 2), 15: (2, 3), 16: (49, 2), 18: (17, 
```

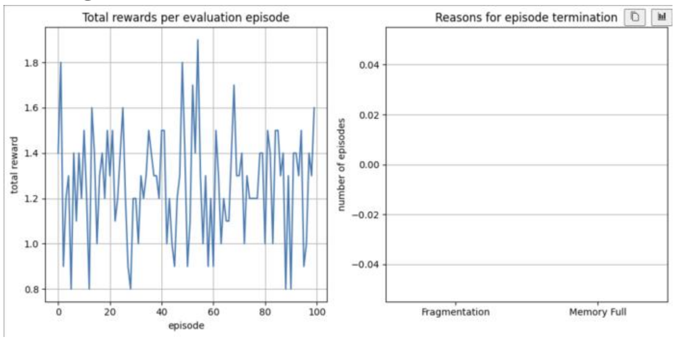
After this, a DQN agent was trained at a learning rate of 0.001 and a starting gamma value of 0.99. These values were chosen after intensive hyper-parametrization as available resources permitted. A training loop was carried out over 1000 episodes.



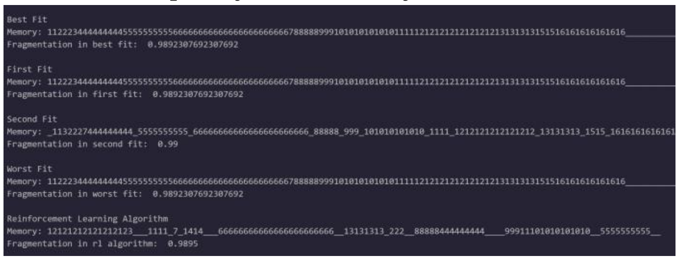
The lack of a proper trend indicates that the model is exploring out of bounds and needs to be trained further. Overall, however, the rewards showed a positive result, demonstrating that the agent was carrying out successful, low-fragmentation memory allocations.



The same was observed after disabling exploration for the model but with even increased rewards. However, insufficient system resources prevented us from further training the model.

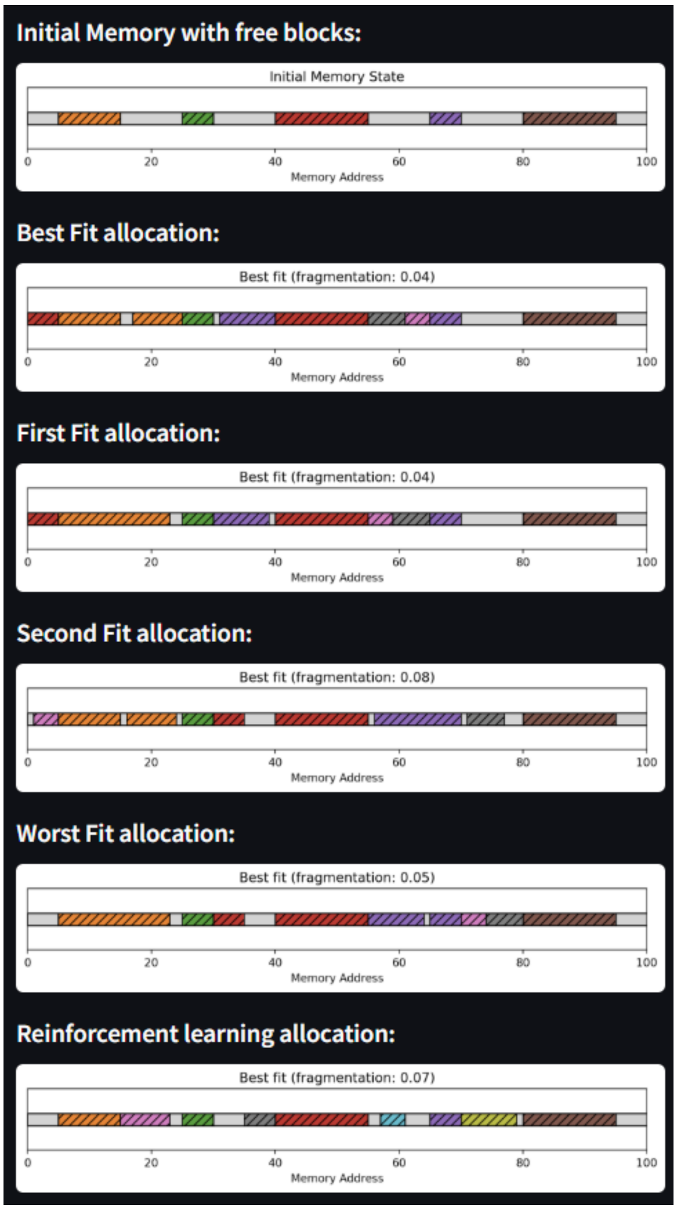


We proceeded with implementing the existing algorithms for first-fit, best-fit, worst-fit and second-fit. We observed differences in each algorithm, which were clearly present only if processes had been previously allocated and deallocated from a fresh memory rather than on a completely reset memory.



```
Best Fit
Memory: 1122234444444444555555555556666666666666666666666666678888899910101010101011111212121212121212131313131515161616161616________
Fragmentation in best fit:  0.9892307692307692

First Fit
Memory: 1122234444444444555555555556666666666666666666666666678888899910101010101011111212121212121212131313131515161616161616________
Fragmentation in first fit:  0.9892307692307692

Second Fit
Memory: _1132227444444444_5555555555_666666666666666666666666_88888_999_101010101010_1111_1212121212121212_13131313_1515_1616161616161
Fragmentation in second fit:  0.99

Worst Fit
Memory: 1122234444444444555555555556666666666666666666666666678888899910101010101011111212121212121212131313131515161616161616________
Fragmentation in worst fit:  0.9892307692307692

Reinforcement Learning Algorithm
Memory: 1212121212121212123___1111_7_1414___666666666666666666666666__13131313_222__88888444444444____9991110101010101010__5555555555__
Fragmentation in rl algorithm:  0.9895
```

After defining a function to determine the degree of external fragmentation, we tested each algorithm.
Trends were observed with respect to successful allocations and the degree of fragmentation.



The reinforcement learning algorithm typically led to the maximum number of successful memory allocations. Conversely, the RL algorithm typically had one of the higher fragmentation rates compared to the remaining algorithms.

Overall, the best-fit algorithm consistently had the least fragmentation degree.

IV. CONCLUSION

The results of Reinforcement learning show strong potential for its use as a memory allocation algorithm. However, resource limitations led to insufficient training of the agent used, and the model was established. Ideally, with far larger training episodes, fragmentation could be further reduced in comparison to the other allocation methods while retaining the rate of allocation of processes. Other limitations experienced include a lack of real-life data to use as training samples, which hinders the training process majorly. A more robust form of comparison between each algorithm needs to be developed and explored as well. Despite our promising results, we acknowledge existing limitations in the thoroughness and scale of our benchmark, challenges in the practicality of system-level implementation, and overall drawbacks of using hand-engineered simulators. We leave these limitations to be addressed in future work.

V. REFERENCES

1. Lim, A., & Maddukuri, A. ([Year]). *Reinforcement Learning for Dynamic Memory Allocation*. [Potentially a thesis, technical report, or conference paper. Add specific publication details if available].
2. [Authors of Various Recent Research Groups]. ([Year(s)]). *RL-based Memory Management Optimization*. [This would be a citation of a collection of works. If you are referencing a specific paper, please provide the authors, year, and publication details. Otherwise, you might summarize the trend and cite a key review or survey paper if one exists].
3. [Authors of Research on AI-driven Optimization]. ([Year]). *Learning-Based System Optimization in Memory-Constrained Environments*. [Potentially a journal article, conference paper, or book chapter. Add specific publication details if available].
4. [Authors of Comparative Studies]. ([Year]). *Benchmarking RL-based Memory Allocation with Classical Approaches*. [Likely a journal article or conference paper focused on comparative evaluation. Add specific publication details if available].