PUI, Probabilistic Planner

David Otgonsuren Rico
otgondav@fel.cvut.cz

***Abstract*—This document describes the implementation of three probabilistic planners. Their implementation, evaluation and discussion of the results. The three planners are FF-replan, Value iteration(VI) asynchronous version, and Monte Carlo Tree Search(MCTS).**

***Index Terms*—FF-replan, VI, MCTS**

I. Implementation

All planners are implemented to work with the dataset of the mazes given. As described in the assignment the reward for stepping on an empty field and staying on the same field is -1, delay field -50 and goal field 200. We want to maximize the reward of the path taken.
The probability of going from field A to a neighbouring field B is 70% and sideways in each direction 15%. The probabilities and rewards are fixed for every run and every strategy.
All of the planners and utility functions associated with them are implemented in C++.

A. FF-replan

The algorithm is implemented from the assignment description with some changes.
First we assume the "Most-likely-outcome determinization". No sideways movement is taken into account when choosing a path.
Second step is finding the path with the highest cost to the goal node. In the assignment description an example to perform this is the Dijkstra's algorithm but for our purposes Breadth First Search is a better solution since we care only about the path that will lead us to the single goal state and not every other field on the map.
Third step is to execute the plan but now with the according probabilities. If it agrees we continue executing. If it differs we plan again. One small change I made to this is when we move sideways but bumped into the wall and stayed on the same place we don't need to replan and can keep using the same plan.

B. Value Iteration asynchronous

The value iteration technique was implemented according to the pseudocode from the lecture on probabilistic planning. I chose the asynchronous version for it's efficiency as it updates the V and π values immediately.
The cost(s,a) function in state s taking action a is computed as

$$\sum_{s' \in S} Pr(s'|s,a) * R(s'|s) \tag{1}$$

Major change is also that in the Bellman update function the V value is updated as a maximum Q value over applicable actions and π as argmax over Q values over applicable actions.
This is because we want to maximize the reward and not minimize it.The threshold is set to 1.0 for all mazes.
This gets us the policy for every state. Following the policy is trivial as for every state we appear in we take the policy according to that state unti we get to the goal state.

C. MCTS

Implemented according to the tutorial slides for MCTS. Repeated selection, expansion, simulation and backpropagation until resource or time runs out and then choosing the child node with the best UCT value and then repeating the whole process until the goal node is reached. In the beginning the time limit for the tree search was set for 0.5 second. This caused problems since in some instances the time limit was unnecessarily large and in some cases not enough at all. It was changed so the maximum depth of the tree is 3 following the best child nodes. Resource constraint scaled better than the time constraint.
The major challenge was to choose the best rollout policy for the simulation step. Choosing purely random moves was not a viable option as it would take an extremely long time before it reached the goal state. Limiting the number of random moves solved the time problem but did not retrieve any valuable information. Using FF-replan as a rollout policy did retrieve valuable information but took too long to finish. Having a fast simulation phase is essential to expand the tree. In the end the rollout policy was set as following a precomputed VI policy. Fast execution and valuable information.

II. Evaluation

All of the strategies were run 1000 times with different seeds on mazes 7x7 to 51x51. Only exception might be some 51x51 mazes that took a lot of time to run by some strategies and were run only 100 times. Mazes 101x101 took especially long time to run and were not added to the evaluation but they were ran at least once.
One of the things measured was runtime of different strategies. Computing runtime for FF-replan is simple but how to compute runtime for VI. I measured getting the policy and following it separately. Since the policy is not affected by the number of runs and can computed once and then followed on different seeds it made sense to measure them separately. Similarly for MCTS which uses the VI policy the measurement for the policy was not included. In figures 1 and 2 it can be seen that obtaining the VI policy costs the most amount of time but happens only once while the other strategies are averages of runtimes on 1000

runs. While the range of times in the two figures is vastly different the structure stays quite similar. This is true for other runtime comparison graphs. They can be found in the plotting/runtimes directory. These graphs were generated by a Python script.

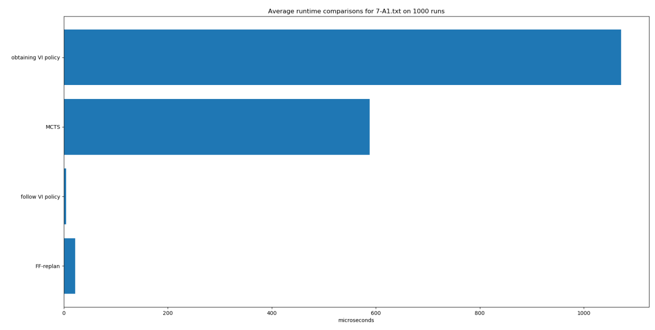


Fig. 1. Run-time comparisons for 7A1

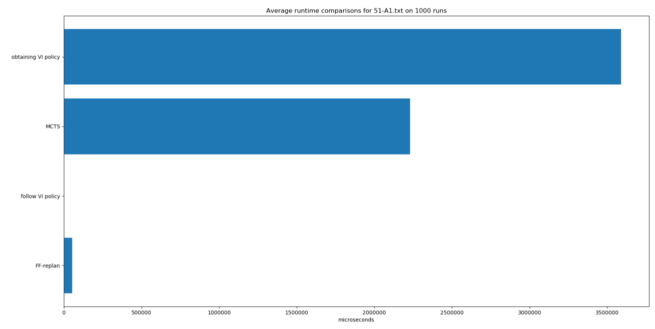


Fig. 2. Run-time comparisons for 51A1

Second measurement was for rewards accumulated for the whole path taken. The rewards are compared in boxplots grouped by the types of mazes. Examples can be seen in figures 3 and 4. These boxplots were generated by a python script and can be found in the plotting/rewards directory along with graphs for the other gropus as well. It is obvious that MCTS performance is much worse than FF-replan and VI. More to this in the discussion part.

Additional information about the performance of different strategies can be viewed from the path taken in the maze. In the output/¡strategy¿ directories we can find path for each maze. The fields visited are marked by letter T, otherwise the maze has the same structure as the input file. Examples can be seen in figures 7, 8, 9 and can be compared to the original 5. Lastly the VI policy can be viewed in similar fashion where for every field, L-left, R-right, U-up, R-right is put in place. Policies for every maze are in the output/policy directory except for the 101E maze which is the only policy for which the policy didn't converge even after a large amount of time.

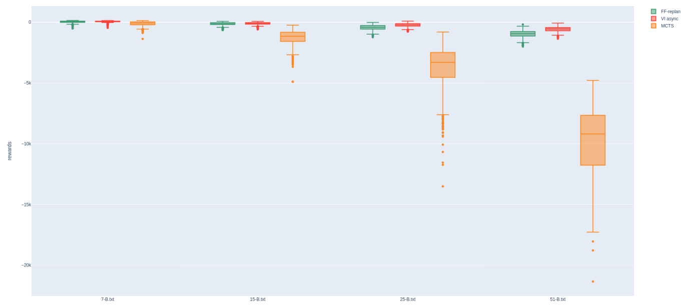


Fig. 3. Rewards for B type mazes

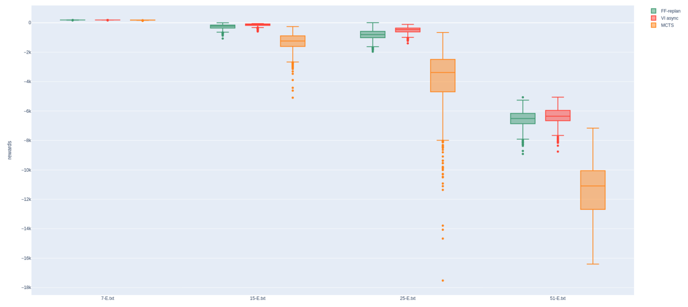
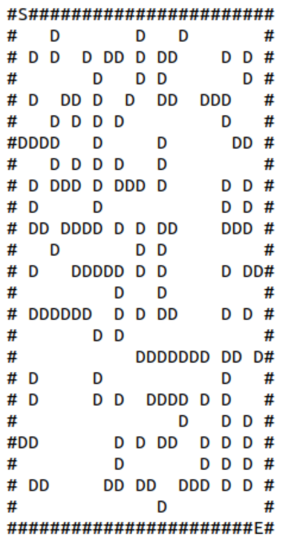


Fig. 4. Rewards for E type mazes

III. Discussion

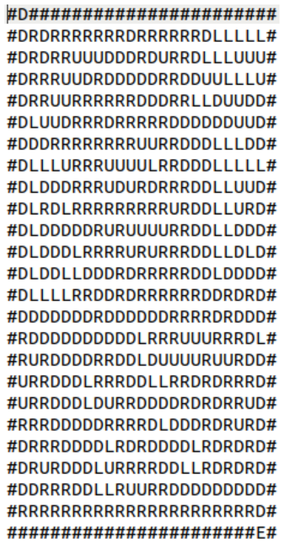
The FF-replan and Value Iteration strategies work quite well. Their rewards are quite comparable for almost every type of maze. The difference between these is in runtime. Which is faster? Depends. If we were to have only one run of the maze then FF-replan would be much faster than VI as can be seen from the runtime graphs. If we were to run them 1000 times on the maze then VI would be much faster. This is because the VI policy can be computed only once for the 1000 runs and following the VI policy is much faster than FF-replan.
Why is MCTS performing so much worse then the other two strategies? When looking at the rewards graph and path taken by MCTS it is visible that it is not choosing the best path and seems like it often gets lost before finding the goal. My reasoning is that during the simulation phase when following the VI policy it can for a field closer to the goal accumulate a reward far worse then in a field further from a goal due to the randomness of taking actions. And with a limit depth of tree it can choose a child node further from a goal. With time it finds the goal but it needs a lot more steps. How can this be improved? Higher limit on the tree depth, in the simulation phase following the policy multiple times and averaging the rewards or some heuristic function that performs well for these types of mazes. Due to the fact that the evaluation of MCTS

already took quite a long time I did not try to implement these potential reward improvements. For the reasons above I think these mazes are not well suited for MCTS.
I enjoyed implementing the different strategies. This concludes my report.



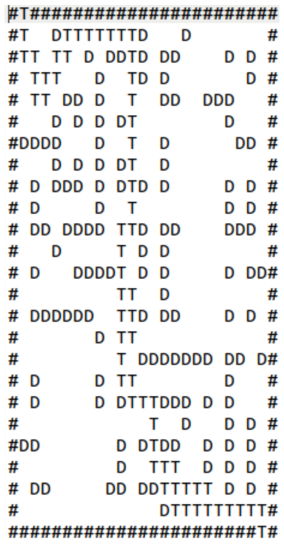
```
#S#######################
#   D       D   D       #
# D D  D DD D DD    D D #
#       D   D D       D #
# D  DD D  D  DD  DDD   #
#   D D D D         D   #
#DDDD   D     D      DD #
#   D D D D   D         #
# D DDD D DDD D     D D #
# D     D           D D #
# DD DDDD D D DD    DDD #
#   D       D D         #
# D   DDDDD D D     D DD#
#         D   D         #
# DDDDDD  D D DD    D D #
#       D D             #
#           DDDDDDD DD D#
# D     D           D   #
# D     D D  DDDD D D   #
#               D   D D #
#DD       D D DD  D D D #
#         D       D D D #
# DD     DD DD  DDD D D #
#             D         #
#######################E#
```

Fig. 5. Input maze 25B



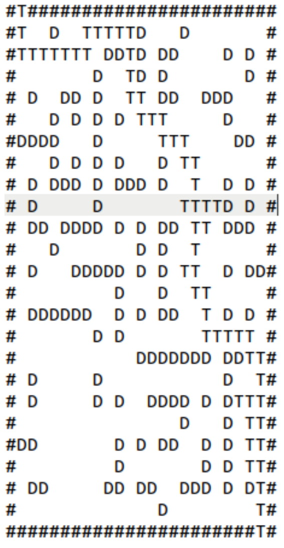
```
#D#######################
#DRDRRRRRRRDRRRRRRDLLLLL#
#DRDRRUUUDDDRDURRDLLLUUU#
#DRRRUUDRDDDDDRRDDUULLLU#
#DRRUURRRRRRDDDRRLLDUUDD#
#DLUUDRRRDRRRRRDDDDDDUUD#
#DDDRRRRRRRRUURRDDDLLLDD#
#DLLLURRRUUUULRRDDDLLLLL#
#DLDDDRRRUDURDRRRDDLLUUD#
#DLRDLRRRRRRRRRURDDLLURD#
#DLDDDDDRURUUUURRDDLLDDD#
#DLDDDLRRRRURURRRDDLLDLD#
#DLDDLLDDDRDRRRRRDDLDDDD#
#DLLLLRRDDRDRRRRRRDDRDRD#
#DDDDDDDRDDDDDDDRRRRDRDDD#
#RDDDDDDDDDDDLRRRUUURRRDL#
#RURDDDDRRDDLDUUUURUURDD#
#URRDDDLRRRDDLLRRDRDRRRD#
#URRDDDLDURRDDDDRDRDRRUD#
#RRRDDDDDRRRRDLDDDRDRURD#
#DRRRDDDDLRDRDDDDLRDRDRD#
#DRURDDDLURRRRDDLLRDRDRD#
#DDRRRDDLLRUURRDDDDDDDDD#
#RRRRRRRRRRRRRRRRRRRRRRD#
#######################E#
```

Fig. 6. VI policy for 25B



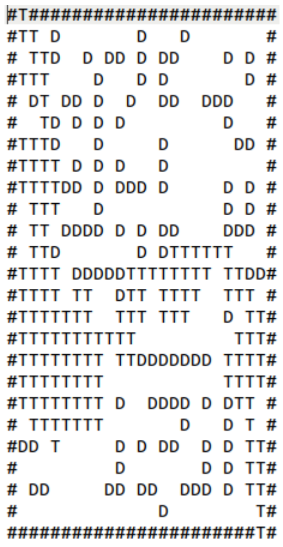
```
#T#######################
#T  DTTTTTTTD   D       #
#TT TT D DDTD DD    D D #
# TTT   D  TD D       D #
# TT DD D  T  DD  DDD   #
#   D D D DT        D   #
#DDDD   D  T  D      DD #
#   D D D DT  D         #
# D DDD D DTD D     D D #
# D     D  T        D D #
# DD DDDD TTD DD    DDD #
#   D     T D D         #
# D   DDDDT D D     D DD#
#         TT  D         #
# DDDDDD  TTD DD    D D #
#       D TT            #
#         T DDDDDDD DD D#
# D     D TT        D   #
# D     D DTTTDDD D D   #
#            T  D   D D #
#DD       D DTDD  D D D #
#         D  TTT  D D D #
# DD     DD DDTTTTT D D #
#             DTTTTTTTTT#
#######################T#
```

Fig. 7. Path taken by FF-replan on 25B



```
#T#######################
#T  D  TTTTTD   D       #
#TTTTTTT DDTD DD    D D #
#       D  TD D       D #
# D  DD D  TT DD  DDD   #
#   D D D D TTT     D   #
#DDDD   D     TTT    DD #
#   D D D D   D TT      #
# D DDD D DDD D  T  D D #
# D     D       TTTTD D #
# DD DDDD D D DD TT DDD #
#   D       D D  T      #
# D   DDDDD D D TT  D DD#
#         D   D  TT     #
# DDDDDD  D D DD  T D D #
#       D D       TTTTT #
#           DDDDDDD DDTT#
# D     D           D  T#
# D     D D  DDDD D DTTT#
#               D   D TT#
#DD       D D DD  D D TT#
#         D       D D TT#
# DD     DD DD  DDD D DT#
#             D        T#
#######################T#
```

Fig. 8. Path taken by VI on 25B



```
#T#######################
#TT D       D   D       #
# TTD  D DD D DD    D D #
#TTT    D   D D       D #
# DT DD D  D  DD  DDD   #
#  TD D D D         D   #
#TTTD   D     D      DD #
#TTTT D D D   D         #
#TTTTDD D DDD D     D D #
# TTT   D           D D #
# TT DDDD D D DD    DDD #
# TTD       D DTTTTTT   #
#TTTT DDDDDTTTTTTTT TTDD#
#TTTT TT  DTT TTTT  TTT #
#TTTTTTT  TTT TTT   D TT#
#TTTTTTTTTTT         TTT#
#TTTTTTTT TTDDDDDDD TTTT#
#TTTTTTTT           TTTT#
#TTTTTTTT D  DDDD D DTT #
# TTTTTTT       D   D T #
#DD T     D D DD  D D TT#
#         D       D D TT#
# DD     DD DD  DDD D TT#
#             D        T#
#######################T#
```

Fig. 9. Path taken by MCTS on 25B