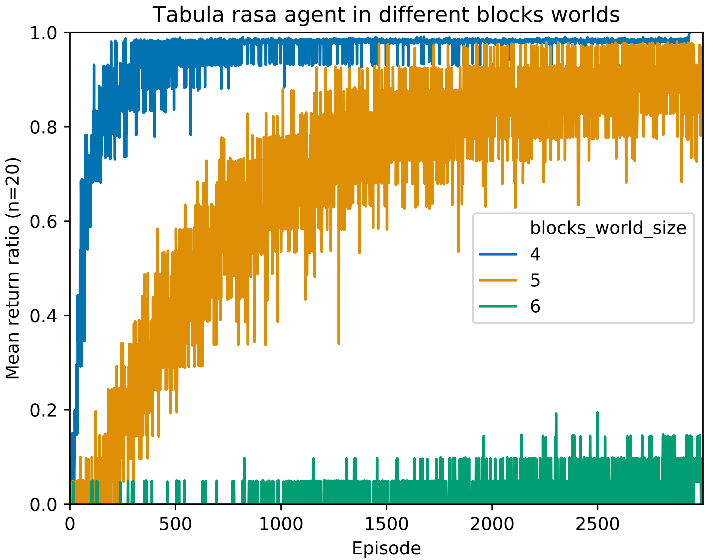
Experiment 1

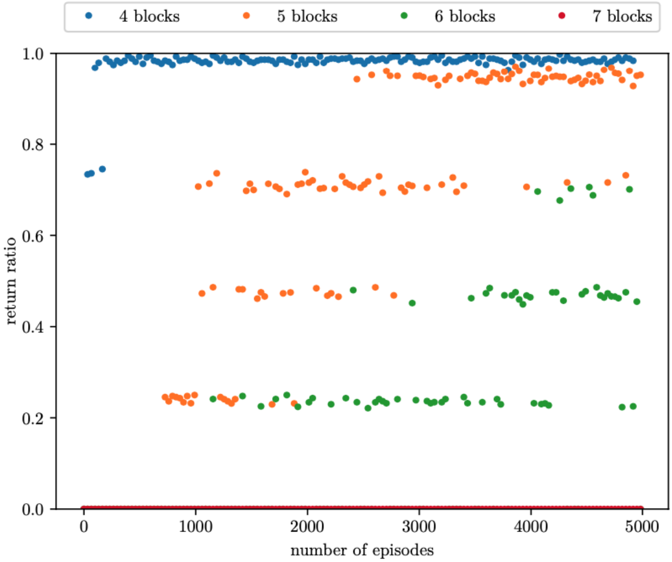
To reproduce the results from the bachelor's thesis

Experiment 1a



Experiment 1a

Comparison with bachelor's thesis



Experiment 1a

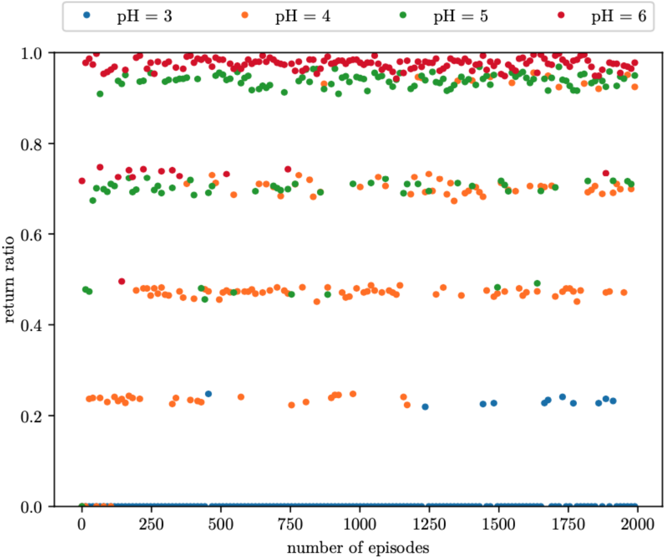
Remarks

Results are in line with those of the bachelor's thesis.

Experiment 1b

Experiment 1b

Comparison with bachelor's thesis



Experiment 1b

Remarks

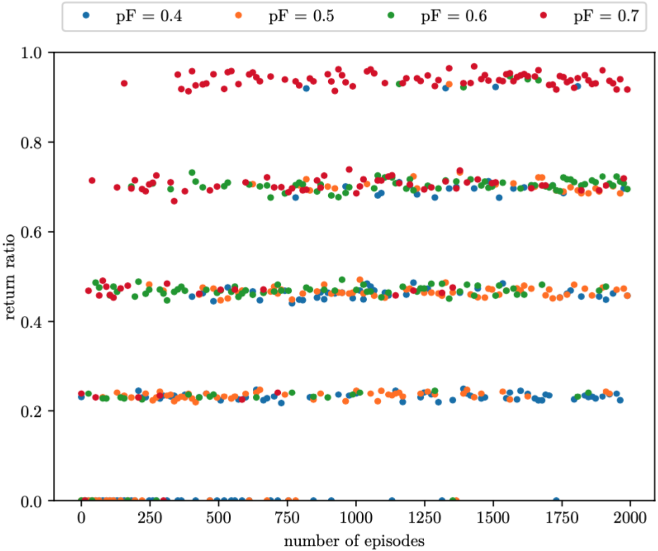
- The results seem to be shifted!
  - $pH = 3$ in the thesis seems roughly equal to $pH = 4$ in my work.
  - $pH = 4$ in the thesis seems roughly equal to $pH = 5$ in my work.
  - $pH = 5$ in the thesis seems roughly equal to $pH = 6$ in my work.
  - My results were generated automatically. Further, The results here were consistent with those in Experiment 2b. Thus, it is unlikely that I made a mistake.
  - In my experiment, the planning horizon shows clear effect. Thus, I also don't think that there is a mistake in the learning agents code.

Experiment 1c

The original output (left) is too messy. Used running average to smooth the result and make it more readable (right).

Experiment 1c

Comparison with bachelor's thesis



Experiment 1c

Remarks

Only the first 1000 episodes were rendered (vs. 2000 in the bachelor's thesis). It's hard to say because of the noise, but the results look similar to those in the thesis.

Conclusion

- Overall, the results seem to be in line with that of the bachelor's thesis.
- A low sample size ($n = 4$) may skew the results
- Generating results took ca. 4-13 hours. The blocks world size has a big impact on performance.
  - *Possible culprit:* Computing the return rate is expensive. It requires computing optimal plans for every step.
  - *Possible culprit:* Computing plans is expensive. For every state, the optimal return up to the planning horizon is computed, even if no answer set leads to a correct solution.
  - **Update** Removed some redundant ASP-solver calls, now the simulation is much faster. However, the points above are still valid.

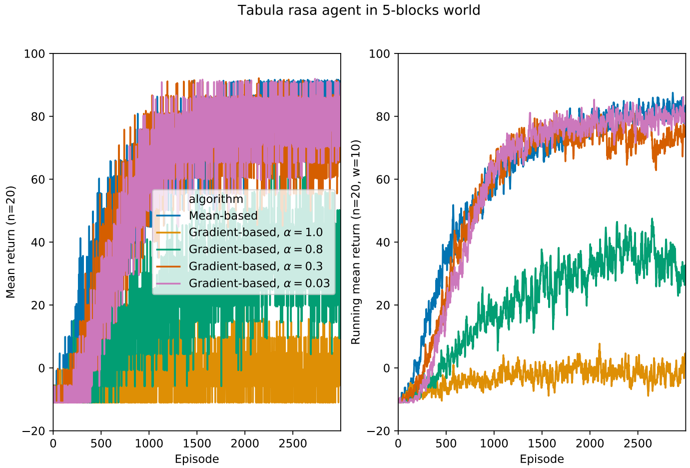
Experiment 2

To compare the mean-based First-visit Monte-Carlo method from the bachelor's thesis with a gradient-based every-visit monte-carlo method.

Note

Switched to plotting return instead of return ratio due to performance reasons.

Experiment 2a



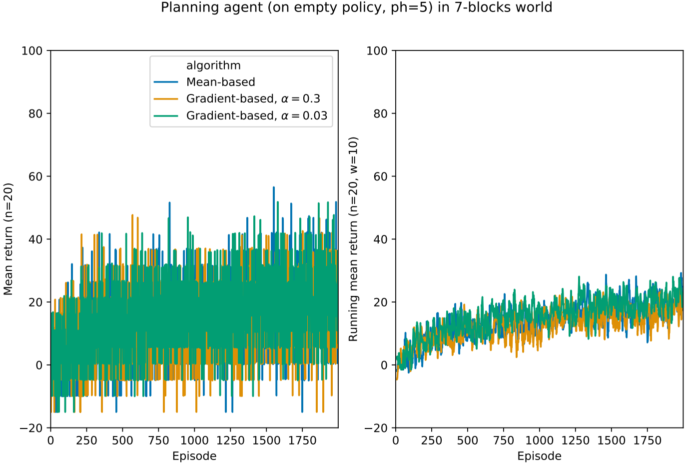
The original output (left) is too messy. Used running average to smooth the result and make it more readable (right).

Experiment 2a

Remarks

- There are subtle differences between the mean-based and the gradient-based algorithm at its best.
- I expected to see an accelerating effect when switching from first-visit MC to every-visit MC, but this seems not to be the case.
- Both $\alpha = 0.3$ and $\alpha = 0.03$ are doing well.
  - Both show similar progress in the first 1000 episodes.
  - Towards the end, $\alpha = 0.3$ gets worse again, which may indicate that α is too large to capture the subtleties needed to improve further.
- However, there seems to be a decrease in performance for higher values ($\alpha \geq 0.8$), which is expected.

Experiment 2b



The original output (left) is too messy. Used running average to smooth the result and make it more readable (right).

Experiment 2b

Remarks

- The low performance is disappointing, compared to the results of the bachelor's project (Experiment 1b, Comparison), where $pH = 5$ almost immediately achieves satisfactory results.
- Despite that, learning is clearly happening and the results are in line with thoe of Experiment 1b.
- There is no clear difference between the mean-based and gradient-based algorithms.

Conclusions

- Overall, the gradient-based method seems to work as well as the mean-based method.
- The gradient-based method handles planning as well as the mean-based method.

Experiment 3

The previous experiments took very long to complete. This experiment is for profiling the framework and identifying possible bottlenecks.

- **Update:** This experiment helped me to find redundant ASP-calls (my own fault) and increase performance 5-fold. The results below are provided with the current (improved) version of the code.

Experiment 3a

Tabula-rasa agent in a 7-blocks world, 150 episodes

Wed Aug 19 10:43:36 2020 exp3a_profile_raw.txt

```
      3637651 function calls (3629949 primitive calls) in 17.924 seconds

   Ordered by: internal time
   List reduced from 2714 to 20 due to restriction <20>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     4057    9.820    0.002    9.820    0.002 {method 'ground' of 'clingo.Control' objects}
     4057    2.046    0.001    2.046    0.001 {method 'load' of 'clingo.Control' objects}
     4057    1.212    0.000    1.458    0.000 {method 'solve' of 'clingo.Control' objects}
     4056    1.197    0.000   15.918    0.004 BlocksWorld.py:107(next_step)
    14419    0.882    0.000    0.882    0.000 {method 'add' of 'clingo.Control' objects}
   160104    0.450    0.000    0.530    0.000 BlocksWorld.py:160(parse_part_state)
     4057    0.415    0.000    0.415    0.000 ClingoBridge.py:6(__init__)
    41689    0.220    0.000    0.220    0.000 {method 'symbols' of 'clingo.Model' objects}
    18816    0.177    0.000    0.758    0.000 BlocksWorld.py:185(parse_state)
    53872    0.108    0.000    0.144    0.000 entities.py:53(<listcomp>)
    37233    0.097    0.000    0.122    0.000 BlocksWorld.py:173(parse_action)
   686190    0.097    0.000    0.097    0.000 entities.py:8(__eq__)
        1    0.081    0.081    1.152    1.152 BlocksWorld.py:42(generate_all_states)
   160111    0.059    0.000    0.081    0.000 entities.py:2(__init__)
    53872    0.052    0.000    0.204    0.000 entities.py:52(__hash__)
      254    0.044    0.000    0.044    0.000 {built-in method marshal.loads}
    51/50    0.040    0.001    0.042    0.001 {built-in method _imp.create_dynamic}
   422366    0.040    0.000    0.040    0.000 entities.py:5(__repr__)
     4057    0.038    0.000   12.201    0.003 ClingoBridge.py:29(run)
   448953    0.036    0.000    0.036    0.000 {method 'append' of 'list' objects}
```

Experiment 3b

Planning agent (on empty policy, ph=5) in a 7-blocks world, 150 episodes

Wed Aug 19 10:44:10 2020 exp3b_profile_raw.txt

```
         3844589 function calls (3836887 primitive calls) in 34.476 seconds

   Ordered by: internal time
   List reduced from 2715 to 20 due to restriction <20>

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
     4936   18.177    0.004   18.177    0.004 {method 'ground' of 'clingo.Control' objects}
     4936    7.226    0.001    7.489    0.002 {method 'solve' of 'clingo.Control' objects}
     4936    2.515    0.001    2.515    0.001 {method 'load' of 'clingo.Control' objects}
     4935    2.409    0.000   32.482    0.007 BlocksWorld.py:107(next_step)
    16974    1.051    0.000    1.051    0.000 {method 'add' of 'clingo.Control' objects}
     4936    0.517    0.000    0.517    0.000 ClingoBridge.py:6(__init__)
   166257    0.461    0.000    0.546    0.000 BlocksWorld.py:160(parse_part_state)
    42619    0.235    0.000    0.235    0.000 {method 'symbols' of 'clingo.Model' objects}
    88278    0.207    0.000    0.257    0.000 BlocksWorld.py:173(parse_action)
    18816    0.161    0.000    0.723    0.000 BlocksWorld.py:185(parse_state)
    56354    0.114    0.000    0.151    0.000 entities.py:53(<listcomp>)
        1    0.083    0.083    1.102    1.102 BlocksWorld.py:42(generate_all_states)
   522544    0.077    0.000    0.077    0.000 entities.py:8(__eq__)
   166264    0.062    0.000    0.085    0.000 entities.py:2(__init__)
    56354    0.054    0.000    0.212    0.000 entities.py:52(__hash__)
     4936    0.049    0.000   26.769    0.005 ClingoBridge.py:29(run)
   476028    0.047    0.000    0.047    0.000 entities.py:5(__repr__)
      254    0.045    0.000    0.045    0.000 {built-in method marshal.loads}
   516934    0.043    0.000    0.043    0.000 {method 'append' of 'list' objects}
    51/50    0.041    0.001    0.043    0.001 {built-in method _imp.create_dynamic}
```

Experiment 3

Observations

- For the tabula rasa agent, 16 out of the 18 seconds are spent in the `next_step` method of the `BlockWorld`.
  - This method starts the ASP-solver, so it is unsurprising that it takes time
  - Most time (10s) is spent grounding
- For the planning agent, double the time was spent in `next_step`, while the number of calls increased only by 25.