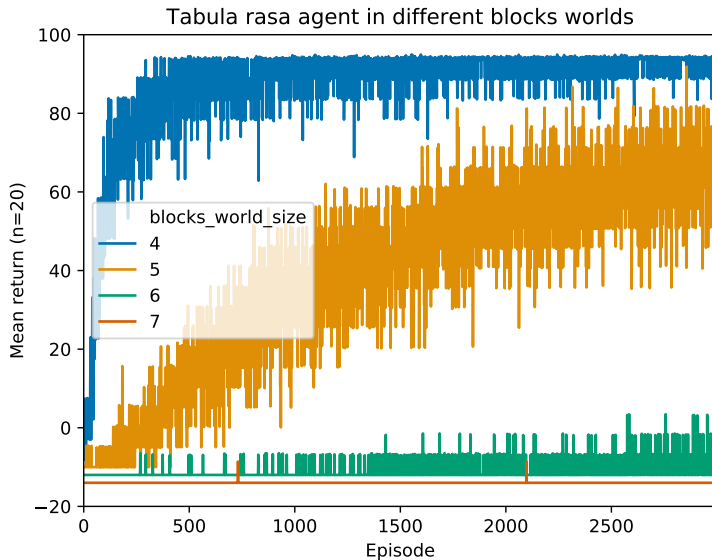


Experiment 1

To reproduce the results from the bachelor's thesis

Experiment 1a



Experiment 1a

Remarks

- ▶ Results are as expected.
- ▶ Bigger blocks worlds need much more episodes to train.

Experiment 1b



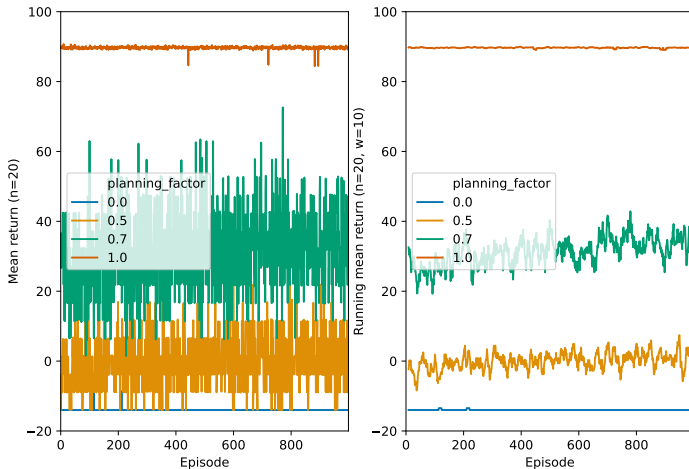
Experiment 1b

Remarks

- ▶ Adding a planning component clearly speeds up training.
- ▶ Already small planning horizons cause significant increase in training speed (compared to the previous experiment).
- ▶ Bigger planning horizons speed up training even more.

Experiment 1c

Planning agent (at random times, $ph=12$) in 7-blocks world



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

Experiment 1c

Remarks

- ▶ Planning at random times speeds up training as well.
- ▶ However, even with near-perfect planning horizons, planning at random times is not as effective as planning for new states.
 - ▶ ...except when `planning_factor=1.0`, i.e. when the planner is used at every timestep - too expensive!

Conclusion

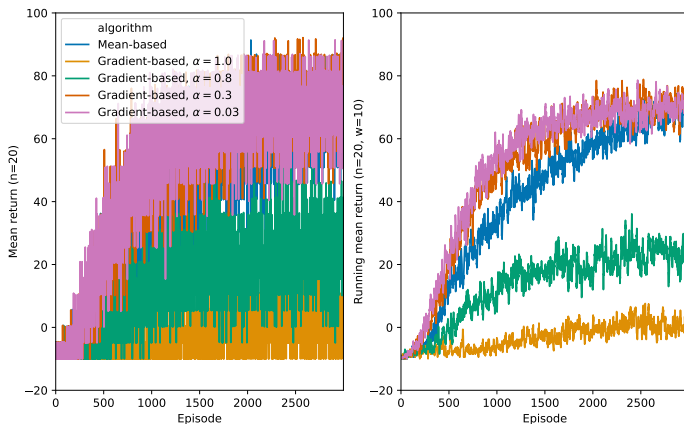
- ▶ Overall, the results seem to be in line with that of the bachelor's thesis.
- ▶ Planning clearly improves the training process.

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.

Experiment 2a

Tabula rasa agent in 5-blocks world



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

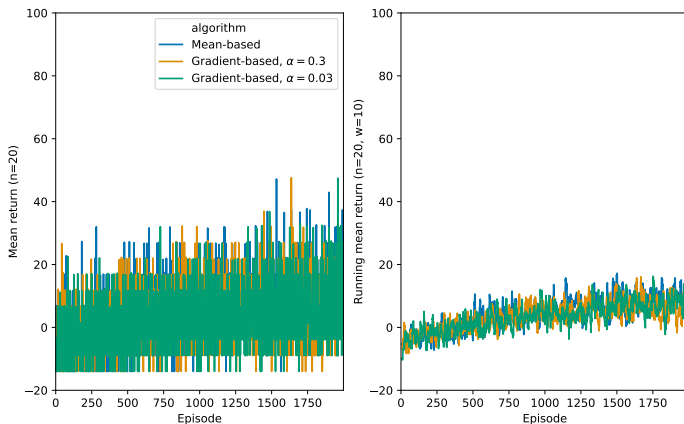
Experiment 2a

Remarks

- ▶ There is no clear difference between the mean-based and the gradient-based algorithm at its best.
 - ▶ It may look like the gradient-based algorithm is better, but this is just a coincidence. Re-running the experiment may cause the mean-based algorithm to perform better.
 - ▶ *Note* Probably a good idea to re-run this experiment with a higher sample size.
- ▶ Both $\alpha = 0.3$ and $\alpha = 0.03$ seem to be good parameter choices.
- ▶ There seems to be a decrease in performance for higher values ($\alpha \geq 0.8$), which is expected.

Experiment 2b

Planning agent (for new states, $ph=5$) in 7-blocks world



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

Experiment 2b

- ▶ Planning still speeds up training, also for the gradient-based algorithm.
- ▶ 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

This experiment is for profiling the framework and identifying possible bottlenecks.

Experiment 3a

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

Fri Sep 4 12:14:58 2020 exp3a_profile_raw.txt

1164209 function calls (1156493 primitive calls) in 6.990 seconds

Ordered by: internal time

List reduced from 2708 to 20 due to restriction <20>

ncalls	totttime	percall	cumtime	percall	filename:lineno(function)
2251	2.993	0.001	2.993	0.001	{method 'ground' of 'clingo.Control' objects}
4501	1.252	0.000	1.252	0.000	{method 'load' of 'clingo.Control' objects}
8852	0.536	0.000	0.536	0.000	{method 'add' of 'clingo.Control' objects}
150	0.324	0.002	5.735	0.038	MonteCarlo.py:49(generate_episode)
2100	0.323	0.000	5.007	0.002	markov_decision_procedure.py:31(transition)
301064	0.301	0.000	0.301	0.000	blocksworld.py:88(<genexpr>)
1	0.199	0.199	0.644	0.644	blocksworld.py:88(<listcomp>)
39883	0.162	0.000	0.162	0.000	{method 'symbols' of 'clingo.Model' objects}
2251	0.070	0.000	0.070	0.000	{method 'solve' of 'clingo.Control' objects}
51/50	0.069	0.001	0.071	0.001	{built-in method _imp.create_dynamic}
257	0.051	0.000	0.051	0.000	{built-in method marshal.loads}
1	0.039	0.039	0.039	0.039	{built-in method mkl._py_mkl_service.get_version}
257	0.038	0.000	0.044	0.000	<frozen importlib._bootstrap_external>:914(get_data)
1391	0.032	0.000	0.032	0.000	{built-in method posix.stat}
4530	0.025	0.000	0.039	0.000	posixpath.py:338(normpath)
2467	0.023	0.000	0.039	0.000	inspect.py:613(cleandoc)
150	0.022	0.000	0.360	0.002	markov_decision_procedure.py:13(__init__)
150	0.020	0.000	0.338	0.002	markov_decision_procedure.py:91(_compute_available_actions)
1061/970	0.015	0.000	0.059	0.000	{built-in method builtins.__build_class__}
16318/16314	0.014	0.000	0.025	0.000	{method 'join' of 'str' objects}

Experiment 3b

Planning agent (for new states, $ph=5$) in a 7-blocks world, 150 episodes

Fri Sep 4 12:15:22 2020 exp3b_profile_raw.txt

1332591 function calls (1324875 primitive calls) in 22.789 seconds

Ordered by: internal time

List reduced from 2712 to 20 due to restriction <20>

ncalls	totttime	percall	cumtime	percall	filename:lineno(function)
3532	11.088	0.003	11.088	0.003	{method 'ground' of 'clingo.Control' objects}
3532	3.833	0.001	3.833	0.001	{method 'solve' of 'clingo.Control' objects}
8422	2.027	0.000	2.027	0.000	{method 'load' of 'clingo.Control' objects}
1359	1.871	0.001	15.146	0.011	planner.py:17(suggest_next_action)
150	1.308	0.009	21.602	0.144	MonteCarlo.py:49(generate_episode)
13976	0.818	0.000	0.818	0.000	{method 'add' of 'clingo.Control' objects}
2022	0.335	0.000	4.764	0.002	markov_decision_procedure.py:31(transition)
301064	0.292	0.000	0.292	0.000	blocksworld.py:88(<genexpr>)
1	0.209	0.209	0.647	0.647	blocksworld.py:88(<listcomp>)
41164	0.175	0.000	0.175	0.000	{method 'symbols' of 'clingo.Model' objects}
257	0.048	0.000	0.048	0.000	{built-in method marshal.loads}
7093	0.041	0.000	0.062	0.000	posixpath.py:338(normpath)
51/50	0.040	0.001	0.042	0.001	{built-in method _imp.create_dynamic}
1	0.039	0.039	0.039	0.039	{built-in method mkl._py_mkl_service.get_version}
2467	0.022	0.000	0.038	0.000	inspect.py:613(cleandoc)
21443/21439	0.021	0.000	0.037	0.000	{method 'join' of 'str' objects}
150	0.021	0.000	0.321	0.002	markov_decision_procedure.py:91(_compute_available_actions)
150	0.020	0.000	0.342	0.002	markov_decision_procedure.py:13(_init_)
7202	0.019	0.000	0.026	0.000	posixpath.py:75(join)
1391	0.019	0.000	0.019	0.000	{built-in method posix.stat}

Experiment 3

Observations

- ▶ Let's compare these results to the last implementation:
 - ▶ The tabular rasa agents runtime decreased from 17.9s to 7.0s (−60%)
 - ▶ The planning agent runtime decreased from 34.5s to 22.7s (−34%)
- ▶ The tabula-rasa agent spent ~5s in transition to next states and ~1s iterating all 7-blocks world states.
- ▶ The planning agent spent ~5s in transition, ~1s iterating all states and ~15s planning.