ASP-Driven Emergency Planning for Norm Violations in Reinforcement Learning

—

Technical Appendix

Contents

1 Overview **2**
1.1 Structure . . . 2
1.2 Experimental Setup . . . 2

2 Gardener **2**
2.1 Data . . . 2
2.2 Setup . . . 3
2.3 Logic Program . . . 3
2.4 Results . . . 5
2.4.1 Deterministic (1-Step) . . . 6
2.4.2 Nondeterministic (1-Step) . . . 8
2.4.3 Deterministic (2-Step) . . . 11
2.4.4 Nondeterministic (2-Step) . . . 14

3 Pacman **16**
3.1 Data . . . 16
3.2 Setup . . . 16
3.3 Logic Program . . . 17
3.4 Results . . . 18
3.4.1 Small . . . 19
3.4.2 Medium . . . 20
3.4.3 Original . . . 21

4 CCN **23**
4.1 Data . . . 23
4.2 Setup . . . 23
4.3 Logic Program . . . 24
4.4 Results . . . 25

1 Overview

1.1 Structure

The appendix is structured after the three experiments conducted. Each section contains information about the data set, how to reproduce the experiments and a comprehensive list of achieved results. All code and data used for the experiments can be found in the provided `Appendix.zip`, which has the following file structure:

```
/
├─ Code
│  ├─ ccn...  This directory holds all executable code required to link ndnSim with
│  │          our framework.  The python code required after linking ndnSim is found
│  │          in a subdirectory named framework.
│  │  └─ framework
│  ├─ gardener...  This directory holds all executable code required for gardener.
│  └─ pacman...  This directory holds all executable code required for pacman.
├─ Data
│  ├─ CCN...  This directory holds all instances of CCN.
│  ├─ Gardener
│  │  └─ instances...  This directory holds all instances of Gardener.  The
│  │                   trained RL policies of each instance are stored in a
│  │                   subdirectory named learning.
│  │     └─ learning
│  └─ Pacman
│     └─ layouts...  This directory holds all instances of pacman.  The
│                    trained RL policies of each instance are stored in a
│                    subdirectory named learning.
│        └─ learning
└─ Appendix.pdf ...  This PDF.
```

1.2 Experimental Setup

We use the ASP solver clingo (version 5.7.1) to solve the ASP program of the framework. Further code and the RL implementations are in Python.

The experiments were run on a Linux server with two Intel Xeon CPU E5-2650 v4 (12 cores @ 2.20GHz, no hyperthreading) and 256GB RAM.

We use utility-based policy fixing to prevent that the agent gets stuck by unsolvability (i.e., finds no strict k-policy fix) and ease achieving the goal. Exact weights for the constraints and rewards of each experiment can be found in their respective sections.

For Gardener and Pacman the listed results of the various parameter combinations are averages over multiple runs. The listed variances in each table compare the variance of norm violations of the baseline RL setting with the variance of norm violations over all parameter configurations using the framework within a table.

Over the conducted experiments, the variance when using our framework was smaller compared to the baseline RL, with the exception of a single case (Table 8), and in many cases it was significantly smaller; recall that for Pacman the policy *vegetarian* allows to eat blue ghosts, and in few cases (Tables 37, 44, 44) the variance of eating blue ghosts was higher than in the RL baseline. Overall, the experimental results indicate a high reliability of the framework in terms of decreasing norm violations.

2 Gardener

2.1 Data

We tested the implementation of the framework on 80 deterministic and 80 nondeterministic instances. These instances were randomly generated using the Python script `Code/gardener/gardener.py` and can be found in the data part of the appendix at `Data/Gardener`. To generate new random instances, one can use the following command:

```
python3 gardener.py -m 0 -n <n> --instSizes <sizes> --parameter <params> -- prefix <prefix>
```

where `<n>` is the number of instances (per size) that should be generated. `<sizes>` is a comma-separated string of numbers, where each number x corresponds to an instance size that is x^2. `<params>` is another comma-separated string of exactly three numbers p_w,p_p and p_f. All parameters are a number between 0 and 1 describing a percentage of cells. p_w describes the percentage of cells that should be covered in walls. Parameter p_p is the percentage of cells that should be covered in phenomena and p_f is the percentage of cells (of cells containing phenomena) that should be covered by a frog. Finally, `<prefix>` is a simple string that is prefixed to every generated instance. The optional argument `-r <seed>` fixes the random seed to a specific value. In our testing, we used the random seed 42 to generate the instances found in `Data/Gardener`.

Once the program finished creating the instance files, it further trains a RL learning algorithm using Q-learning on these files ($50000 + s^2 * 5$ episodes of learning, where s^2 is the size of the instance) and stores the learned Q-table in a `.pkl` file.

2.2 Setup

Each instance of Gardener can be executed with the following command:

```
python3 gardener.py -m <mode> --horizon <k> --radius <r> <path-to-instance>
```

with `<mode>` being a number between 1 and 3 (1 = run RL, 2 = run the framework, 3 = run the framework with telingo). `<k>` and `<r>` are the horizon and the window radius and are only considered if mode 2 or 3 is selected. The program expects that there is a pre-trained `.pkl` file in the directory `<path-to-instance>/learning` named equally to the instance. When generating instances using `gardener.py`, the RL algorithm is automatically trained and the learning files generated.

There is an optional argument `--cache <c>`, only considered in mode 2, that tells the program how many steps to take before computing a new k-policy fix (default is 1). Another optional argument, `-i`, is available in all three modes and informs the program to visualize the agent's process in a graphical interface. Finally, the optional argument `-r <seed>` fixes the random seed to a specific value. In our testing, we used the starting random seed 42 to generate a new random seed for every test.

2.3 Logic Program

The following code corresponds to the logic program `Code/gardener/program.lp`, which is the logic program used in our framework for deterministic and nondeterministic instances of the Gardener game. The logic program is grounded only once per instance. To reflect the state of the window around the agent, the external atoms are then changed dynamically in `Code/gardener/logic.py` in each step. Hence, Gardener includes all optimizations laid out in Section "Program Optimization" of the main document.

The rewards from following policy preferences and penalties from violating a norm are encoded in the logic program. In our implementation, all rewards and penalties are accounted on the same priority level. Penalties for killing a frog or plant are equal (5) and higher than any potential reward of a single action ($0 - 3$). To discourage the agent from looping in a norm-compliant trajectory, an additional penalty (10), sanctioning plans containing repeated states, is introduced. Similarly, an additional reward (5) when reaching the target is established and rewards received from following policy preferences are multiplied by an increasing number when the agent re-visits a cell. Earlier violations and norm adherence are weighted higher to discourage immediate violations or policy deviations.

```
% external constants: frogs, horizon, radius, size


%%% Windowing %%%
% This section implements MSS and the windowing technique described in section
% "Program Optimization" in the main document

% player always starts at the 0 position
pcol(0,0).
prow(0,0).

% frogs can be anywhere within the window
#external fcol(F,C,0) : F=0..frogs-1, C=-radius..radius.
#external frow(F,R,0) : F=0..frogs-1, R=-radius..radius.

% if a frog is outside the current window, it will not be considered for
% the computation
```

```
#external foutside(F) : F=0..frogs-1.

% depending on where the window is, the cells can contain walls or plants
#external wall(C,R) : C=-radius..radius, R=-radius..radius.
#external plant(C,R) : C=-radius..radius, R=-radius..radius.
#external target(C,R) : C=-radius..radius, R=-radius..radius.

% a multiplier for the reward
#external multi(M) : M=1..10.

% rewards for the actions of the agent
#external action(A, S, C, R) : A=0..3, S=0..3, C=-radius..radius, R=-radius..radius.


%%% P_gen %%%
% This section corresponds to the subprogram P_gen described in section
% "ASP Framework" in the main document

% movement of the agent
pmove(0, T) | pmove(1, T) :- T=1..horizon - 1.
prow(R + 1,T) | prow(R - 1,T) :- prow(R,T-1), pmove(0, T).
pcol(C,T) :- pcol(C,T-1), pmove(0, T).
pcol(C + 1,T) | pcol(C - 1,T) :- pcol(C,T-1), pmove(1, T).
prow(R,T) :- prow(R,T-1), pmove(1, T).

% hard restrictions (agent cannot leave window or move into a wall)
:- pcol(C,_), C = -radius - 1.
:- pcol(C,_), C = radius + 1.
:- prow(R,_), R = -radius - 1.
:- prow(R,_), R = radius + 1.
:- prow(R,T), pcol(C,T), wall(C,R).

% penalty for violating norm when stepping onto a plant
:~ prow(R,T), pcol(C,T), plant(C,R), T > 0. [(horizon - T) * 5@1]

% penalty for using the same cell twice
:~ prow(R,T), pcol(C,T), prow(R,T1), pcol(C,T1), T != T1, multi(M). [M * 10@1]

% encode which action has been taken
action_taken(0, T, C, R) :- pcol(C,T), prow(R,T), prow(R+1,T+1).
action_taken(1, T, C, R) :- pcol(C,T), prow(R,T), prow(R-1,T+1).
action_taken(2, T, C, R) :- pcol(C,T), prow(R,T), pcol(C-1,T+1).
action_taken(3, T, C, R) :- pcol(C,T), prow(R,T), pcol(C+1,T+1).

% check the reward for each time step
reward(S, T) :- action_taken(A,T,C,R), action(A,B,C,R), multi(M), S = (horizon - T) * M * B.
reward(S, horizon+1):- prow(R,T), pcol(C,T), target(C,R), multi(M), S = (horizon - T) * M * 5.


%%% P_check %%%
% This section corresponds to the subprogram P_check described in section
% "ASP Framework" in the main document

% movement of the frogs
fmove(F,0, T) | fmove(F,1, T) :- F=0..frogs-1, T=1..horizon - 1.
frow(F,R + 1,T) | frow(F,R - 1,T) :- frow(F,R,T-1), fmove(F,0, T).
fcol(F,C,T) :- fcol(F,C,T-1), fmove(F,0, T).
fcol(F,C + 1,T) | fcol(F,C - 1,T) :- fcol(F,C,T-1), fmove(F,1, T).
frow(F,R,T) :- frow(F,R,T-1), fmove(F,1, T).

% saturate if frogs make illegal move
sat :- fcol(_,C,_), C = -radius - 1.
sat :- fcol(_,C,_), C = radius + 1.
sat :- frow(_,R,_), R = -radius - 1.
sat :- frow(_,R,_), R = radius + 1.
sat :- frow(F,R,T), fcol(F,C,T), wall(C,R).

% check saturation criteria
ok(0).
ok(0,F) :- F=0..frogs-1.
sat :- ok(horizon-1).
```

```
:- not sat.

% if deterministic instance
ok(T) :- T=1..horizon - 1, frogs = 0.

% saturate to M_sat as described in section "ASP Framework -> Saturation
% technique" in the main document
fcol(F,C,T) :- C = -radius..radius, sat, T=1..horizon-1, F=0..frogs-1.
frow(F,R,T) :- R = -radius..radius, sat, T=1..horizon-1, F=0..frogs-1.
fmove(F,0,T) :- F=0..frogs-1, T=1..horizon-1, sat.
fmove(F,1,T) :- F=0..frogs-1, T=1..horizon-1, sat.

% each frog either needs to be safely away from player or give penalty
ok(T,F) :- pcol(C,T), fcol(F,C',T), C != C', ok(T-1).
ok(T,F) :- prow(R,T), frow(F,R',T), R != R', ok(T-1).
good(T,G) | bad(T,G) :- G=0..frogs-1, T=1..horizon-1.
ok(T,F) :- bad(T,F).
ok(T,F) :- foutside(F), T=1..horizon-1.
ok(T) :- {ok(T,F) : F=0..frogs-1} = frogs, ok(T-1), frogs > 0.

% penalty for violating the norm of stepping onto a frog
penalty(P, T) :- P = (horizon - T) * 5, bad(T,F).


%%% Optimization %%%
% Here (in addition with weak constraints above) we implement the utilitarian
% policy fix described in section "Policy Fixing" in the main document.

% optimize for maximum reward and minimal penalty
% internally clingo negates #maximize statements and translates them into
% #minimize statements. Hence these optimization statements amounts to a
% combined optimization that aims to maximize rewards - penalties
#minimize {S@1,T : penalty(S, T)}.
#maximize {R@1,T : reward(R,T)}.


#show action_taken/4.
```

2.4 Results

The results achieved are divided in subsections, depending on whether the tested instances contain frogs (nondeterministic) or not (deterministic). We further distinguish between the application of the framework in which we compute a new k-policy fix in every step (1-Step) or only every second step (2-Step).

Each of the tables in their respective subsections lists the results achieved computing a k-policy fix, with various parameters r and k, in comparison with a RL baseline. The value labelled "% Int." hereby describes the percentage of steps in which the framework deviates from the most preferred action by the RL policy. In the deterministic case, each row represents average values over 10 different instance of the same size but with different wall/phenomena placement. In the nondeterministic case, each instance and parameter configuration is additionally run 5 times.

In addition to these results, each table displays the variance of killed plants and frogs for the RL baseline, as well as the combined average over all applications of the framework. In the special case "Deterministic (1-Step)" we also include the computation times when solved with telingo instead of clingo.

Over all our tests in Gardener, we saw an increase in norm compliance when using our framework compared to the RL baseline. Even with $k = 1$ the norm compliance was significantly increased, though even better results were achieved with varying r and k.

Notably, when executing the first two steps of each computed k-policy fix the run time is (as expected) half, while norm compliance is not significantly worse. Using telingo instead of clingo, however, did not result in any meaningful performance gains, which is likely because the optimizations made in the logic program and detailed in Section "Program Optimization" of the main document are not applicable in this case.

2.4.1 Deterministic (1-Step)

Config	Killed Plants/ Frogs	Steps	Avg./ Startup Time (ms)	Telingo Avg./ Startup Time (ms)	% Int.
RL	1.50 / 0.00	9	0 / 0	0 / 0	0.00
r=3, k=1	0.90 / 0.00	12	29 / 38	161 / 184	2.00
r=3, k=2	0.70 / 0.00	11	30 / 45	169 / 190	1.80
r=3, k=4	0.60 / 0.00	11	43 / 101	188 / 231	2.00
r=3, k=6	0.60 / 0.00	12	106 / 242	224 / 318	2.50
r=5, k=1	0.90 / 0.00	12	61 / 72	269 / 288	2.00
r=5, k=2	0.70 / 0.00	11	63 / 80	287 / 303	1.80
r=5, k=4	0.60 / 0.00	11	79 / 144	324 / 377	2.00
r=5, k=6	0.60 / 0.00	12	181 / 370	381 / 535	2.20

Table 1: Instances *big-d-10-0x.lp*, *x=01..10*
size 10×10, 25 walls, 10 plants and 0 frogs
Variance RL (Plants / Frogs): 1.25 / 0.00
Variance FR (Plants / Frogs): 1.06 / 0.00

Config	Killed Plants/ Frogs	Steps	Avg./ Startup Time (ms)	Telingo Avg./ Startup Time (ms)	% Int.
RL	2.70 / 0.00	23	0 / 0	0 / 0	0.00
r=3, k=1	1.90 / 0.00	30	37 / 47	159 / 178	4.60
r=3, k=2	1.80 / 0.00	29	38 / 53	164 / 187	5.00
r=3, k=4	0.70 / 0.00	26	52 / 113	187 / 236	4.50
r=3, k=6	0.60 / 0.00	33	134 / 260	222 / 328	7.00
r=5, k=1	1.90 / 0.00	30	71 / 86	265 / 284	4.60
r=5, k=2	1.70 / 0.00	28	74 / 90	279 / 302	4.20
r=5, k=4	0.70 / 0.00	26	92 / 162	316 / 385	4.30
r=5, k=6	0.60 / 0.00	26	215 / 398	375 / 512	3.50

Table 2: Instances *big-d-25-0x.lp*, *x=01..10*
size 25×25, 156 walls, 63 plants and 0 frogs
Variance RL (Plants / Frogs): 2.61 / 0.00
Variance FR (Plants / Frogs): 1.86 / 0.00

Config	Killed Plants/ Frogs	Steps	Avg./ Startup Time (ms)	Telingo Avg./ Startup Time (ms)	% Int.
RL	6.60 / 0.00	46	0 / 0	0 / 0	0.00
r=3, k=1	3.90 / 0.00	60	67 / 77	164 / 189	9.30
r=3, k=2	3.20 / 0.00	57	69 / 83	173 / 191	8.80
r=3, k=4	2.90 / 0.00	64	84 / 143	190 / 252	13.00
r=3, k=6	2.90 / 0.00	72	174 / 303	223 / 339	17.20
r=5, k=1	3.90 / 0.00	60	114 / 131	283 / 315	9.30
r=5, k=2	3.40 / 0.00	58	116 / 129	292 / 335	9.00
r=5, k=4	2.60 / 0.00	59	135 / 210	327 / 392	10.60
r=5, k=6	2.50 / 0.00	58	264 / 458	385 / 519	9.80

Table 3: Instances *big-d-50-0x.lp*, *x=01..10*
size 50×50, 625 walls, 250 plants and 0 frogs
Variance RL (Plants / Frogs): 4.84 / 0.00
Variance FR (Plants / Frogs): 2.81 / 0.00

Config	Killed Plants/ Frogs	Steps	Avg./ Startup Time (ms)	Telingo Avg./ Startup Time (ms)	% Int.
RL	11.20 / 0.00	86	0 / 0	0 / 0	0.00
r=3, k=1	6.00 / 0.00	109	179 / 180	188 / 194	16.00
r=3, k=2	5.40 / 0.00	104	182 / 196	192 / 207	14.50
r=3, k=4	3.50 / 0.00	104	198 / 256	214 / 261	17.10
r=3, k=6	3.50 / 0.00	114	283 / 414	253 / 338	20.80
r=5, k=1	6.00 / 0.00	109	268 / 258	338 / 343	16.00
r=5, k=2	5.30 / 0.00	104	270 / 266	350 / 356	14.80
r=5, k=4	3.50 / 0.00	103	283 / 333	385 / 424	15.80
r=5, k=6	2.70 / 0.00	100	411 / 572	438 / 548	15.00

Table 4: Instances *big-d-100-0x.lp*, *x=01..10*
size 100×100, 2500 walls, 1000 plants and 0 frogs
Variance RL (Plants / Frogs): 9.96 / 0.00
Variance FR (Plants / Frogs): 9.02 / 0.00

Config	Killed Plants/ Frogs	Steps	Avg./ Startup Time (ms)	Telingo Avg./ Startup Time (ms)	% Int.
RL	0.80 / 0.00	9	0 / 0	0 / 0	0.00
r=3, k=1	0.60 / 0.00	11	28 / 42	160 / 195	1.30
r=3, k=2	0.60 / 0.00	10	30 / 48	165 / 186	0.60
r=3, k=4	0.20 / 0.00	10	47 / 114	191 / 232	1.10
r=3, k=6	0.30 / 0.00	9	112 / 276	225 / 324	0.80
r=5, k=1	0.60 / 0.00	11	59 / 71	270 / 300	1.30
r=5, k=2	0.60 / 0.00	10	62 / 82	281 / 319	0.60
r=5, k=4	0.20 / 0.00	9	80 / 144	318 / 373	0.90
r=5, k=6	0.30 / 0.00	9	196 / 391	375 / 518	0.50

Table 5: Instances *small-d-10-0x.lp*, *x=01..10*
size 10×10, 25 walls, 5 plants and 0 frogs
Variance RL (Plants / Frogs): 0.96 / 0.00
Variance FR (Plants / Frogs): 0.54 / 0.00

Config	Killed Plants/ Frogs	Steps	Avg./ Startup Time (ms)	Telingo Avg./ Startup Time (ms)	% Int.
RL	1.00 / 0.00	24	0 / 0	0 / 0	0.00
r=3, k=1	0.50 / 0.00	26	37 / 48	159 / 182	1.40
r=3, k=2	0.40 / 0.00	26	39 / 58	165 / 189	1.30
r=3, k=4	0.20 / 0.00	26	51 / 116	187 / 241	1.70
r=3, k=6	0.30 / 0.00	29	127 / 264	221 / 325	3.60
r=5, k=1	0.50 / 0.00	26	72 / 83	264 / 289	1.40
r=5, k=2	0.40 / 0.00	26	75 / 92	277 / 306	1.30
r=5, k=4	0.20 / 0.00	26	89 / 156	310 / 357	1.70
r=5, k=6	0.20 / 0.00	25	200 / 391	368 / 505	1.40

Table 6: Instances *small-d-25-0x.lp*, *x=01..10*
size 25×25, 156 walls, 32 plants and 0 frogs
Variance RL (Plants / Frogs): 0.80 / 0.00
Variance FR (Plants / Frogs): 0.27 / 0.00

Config	Killed Plants/ Frogs	Steps	Avg./ Startup Time (ms)	Telingo Avg./ Startup Time (ms)	% Int.
RL	2.90 / 0.00	44	0 / 0	0 / 0	0.00
r=3, k=1	2.10 / 0.00	52	67 / 77	164 / 188	4.50
r=3, k=2	1.80 / 0.00	51	68 / 84	171 / 193	4.60
r=3, k=4	0.60 / 0.00	48	82 / 144	188 / 239	4.70
r=3, k=6	0.50 / 0.00	51	157 / 320	231 / 335	6.20
r=5, k=1	2.10 / 0.00	52	112 / 122	280 / 305	4.50
r=5, k=2	1.70 / 0.00	51	114 / 134	293 / 339	4.40
r=5, k=4	0.60 / 0.00	47	129 / 196	324 / 389	3.90
r=5, k=6	0.50 / 0.00	48	238 / 434	377 / 502	4.40

Table 7: Instances *small-d-50-0x.lp*, *x=01..10*
size 50×50, 625 walls, 125 plants and 0 frogs
Variance RL (Plants / Frogs): 3.89 / 0.00
Variance FR (Plants / Frogs): 1.46 / 0.00

Config	Killed Plants/ Frogs	Steps	Avg./ Startup Time (ms)	Telingo Avg./ Startup Time (ms)	% Int.
RL	5.50 / 0.00	81	0 / 0	0 / 0	0.00
r=3, k=1	2.90 / 0.00	93	179 / 187	185 / 193	9.00
r=3, k=2	2.40 / 0.00	91	181 / 194	189 / 210	8.40
r=3, k=4	1.10 / 0.00	94	195 / 255	210 / 259	11.70
r=3, k=6	1.30 / 0.00	111	274 / 408	244 / 351	20.00
r=5, k=1	2.90 / 0.00	93	261 / 264	331 / 339	9.00
r=5, k=2	2.40 / 0.00	90	260 / 280	341 / 357	8.50
r=5, k=4	1.10 / 0.00	90	276 / 336	377 / 413	9.70
r=5, k=6	0.60 / 0.00	89	388 / 602	428 / 540	9.60

Table 8: Instances *small-d-100-0x.lp*, *x=01..10*
size 100×100, 2500 walls, 500 plants and 0 frogs
Variance RL (Plants / Frogs): 2.25 / 0.00
Variance FR (Plants / Frogs): 3.34 / 0.00

2.4.2 Nondeterministic (1-Step)

Config	Killed Plants/ Frogs	Steps	Avg./ Startup Time (ms)	% Int.
RL	0.70 / 0.50	10	0 / 0	0.00
r=3, k=1	0.62 / 0.60	16	31 / 46	3.64
r=3, k=2	0.46 / 0.48	14	35 / 60	2.88
r=3, k=4	0.38 / 0.56	18	72 / 163	5.04
r=3, k=6	0.48 / 0.48	29	285 / 451	11.54
r=5, k=1	0.56 / 0.58	14	64 / 85	2.84
r=5, k=2	0.42 / 0.46	15	71 / 108	3.60
r=5, k=4	0.42 / 0.42	19	134 / 263	5.50
r=5, k=6	0.42 / 0.48	21	598 / 785	6.80

Table 9: Instances *big-nd-10-0x.lp*, *x=01..10*
size 10×10, 25 walls, 5 plants and 5 frogs
Variance RL (Plants / Frogs): 0.61 / 0.11
Variance FR (Plants / Frogs): 0.43 / 0.15

Config	Killed Plants/ Frogs	Steps	Avg./ Startup Time (ms)	% Int.
RL	1.30 / 1.66	23	1 / 1	0.00
r=3, k=1	0.74 / 0.84	33	51 / 97	6.12
r=3, k=2	0.78 / 0.92	35	64 / 180	7.50
r=3, k=4	0.58 / 0.80	39	217 / 721	9.54
r=3, k=6	0.60 / 0.78	46	1162 / 2254	13.28
r=5, k=1	0.72 / 1.16	33	96 / 184	6.14
r=5, k=2	0.70 / 0.82	34	121 / 359	6.56
r=5, k=4	0.52 / 0.86	41	449 / 1389	11.64
r=5, k=6	0.50 / 1.02	42	2625 / 4912	11.24

Table 10: Instances *big-nd-25-0x.lp*, *x=01..10*
size 25×25, 156 walls, 31 plants and 32 frogs
Variance RL (Plants / Frogs): 0.41 / 0.96
Variance FR (Plants / Frogs): 0.33 / 0.51

Config	Killed Plants/ Frogs	Steps	Avg./ Startup Time (ms)	% Int.
RL	2.70 / 1.70	44	6 / 6	0.00
r=3, k=1	1.64 / 0.84	59	110 / 180	10.82
r=3, k=2	1.48 / 1.00	61	128 / 316	12.72
r=3, k=4	1.24 / 0.72	68	357 / 1087	17.52
r=3, k=6	1.22 / 0.64	87	1656 / 3009	27.30
r=5, k=1	1.52 / 0.96	60	211 / 728	11.16
r=5, k=2	1.30 / 0.82	62	313 / 2063	12.52
r=5, k=4	1.14 / 1.06	70	1655 / 12037	18.08
r=5, k=6	1.20 / 0.80	78	9004 / 52097	22.56

Table 11: Instances *big-nd-50-0x.lp*, *x=01..10*
size 50×50, 625 walls, 125 plants and 125 frogs
Variance RL (Plants / Frogs): 2.01 / 0.64
Variance FR (Plants / Frogs): 1.07 / 0.36

Config	Killed Plants/ Frogs	Steps	Avg./ Startup Time (ms)	% Int.
RL	4.30 / 4.04	89	95 / 94	0.00
r=3, k=1	3.32 / 3.02	128	393 / 466	25.06
r=3, k=2	2.74 / 2.98	128	413 / 613	26.72
r=3, k=4	2.08 / 3.22	161	659 / 1424	47.10
r=3, k=6	2.48 / 2.06	190	1915 / 3973	62.12
r=5, k=1	2.90 / 2.92	123	571 / 1092	21.44
r=5, k=2	2.52 / 2.56	131	682 / 2469	28.76
r=5, k=4	2.04 / 2.46	156	1944 / 12866	43.38
r=5, k=6	2.70 / 1.80	170	9070 / 55999	51.84

Table 12: Instances *big-nd-100-0x.lp*, *x=01..10*
size 100×100, 2500 walls, 500 plants and 500 frogs
Variance RL (Plants / Frogs): 5.61 / 2.67
Variance FR (Plants / Frogs): 2.45 / 2.08

Config	Killed Plants/ Frogs	Steps	Avg./ Startup Time (ms)	% Int.
RL	0.10 / 0.18	8	0 / 0	0.00
r=3, k=1	0.10 / 0.08	10	30 / 45	0.78
r=3, k=2	0.10 / 0.12	10	33 / 54	0.88
r=3, k=4	0.02 / 0.12	10	52 / 133	1.32
r=3, k=6	0.00 / 0.12	14	163 / 344	3.46
r=5, k=1	0.10 / 0.08	10	63 / 81	1.12
r=5, k=2	0.12 / 0.04	10	67 / 97	1.00
r=5, k=4	0.02 / 0.16	12	96 / 203	2.34
r=5, k=6	0.00 / 0.10	10	299 / 571	1.14

Table 13: Instances *small-nd-10-0x.lp*, *x=01..10*
size 10 × 10, 25 walls, 2 plants and 3 frogs
Variance RL (Plants / Frogs): 0.09 / 0.10
Variance FR (Plants / Frogs): 0.05 / 0.04

Config	Killed Plants/ Frogs	Steps	Avg./ Startup Time (ms)	% Int.
RL	0.80 / 0.52	21	0 / 0	0.00
r=3, k=1	0.22 / 0.26	25	44 / 71	2.96
r=3, k=2	0.26 / 0.20	26	49 / 103	3.60
r=3, k=4	0.20 / 0.10	30	108 / 304	5.74
r=3, k=6	0.26 / 0.26	40	452 / 781	10.76
r=5, k=1	0.20 / 0.28	26	83 / 129	3.04
r=5, k=2	0.20 / 0.28	26	94 / 193	3.80
r=5, k=4	0.20 / 0.32	30	213 / 502	5.94
r=5, k=6	0.20 / 0.20	31	989 / 1528	6.20

Table 14: Instances *small-nd-25-0x.lp*, *x=01..10*
size 25 × 25, 156 walls, 16 plants and 16 frogs
Variance RL (Plants / Frogs): 0.96 / 0.26
Variance FR (Plants / Frogs): 0.16 / 0.07

Config	Killed Plants/ Frogs	Steps	Avg./ Startup Time (ms)	% Int.
RL	2.10 / 0.94	50	3 / 3	0.00
r=3, k=1	1.22 / 0.82	62	92 / 160	8.08
r=3, k=2	0.90 / 0.46	60	103 / 296	7.44
r=3, k=4	0.64 / 0.52	71	256 / 1023	14.18
r=3, k=6	0.66 / 0.36	83	951 / 2851	19.76
r=5, k=1	1.24 / 0.60	63	157 / 362	7.92
r=5, k=2	1.14 / 0.54	63	189 / 785	8.40
r=5, k=4	0.64 / 0.32	68	581 / 3049	12.42
r=5, k=6	0.60 / 0.42	71	2957 / 11712	14.08

Table 15: Instances *small-nd-50-0x.lp*, *x=01..10*
size 50 × 50, 625 walls, 62 plants and 63 frogs
Variance RL (Plants / Frogs): 2.29 / 0.27
Variance FR (Plants / Frogs): 1.07 / 0.23

Config	Killed Plants/ Frogs	Steps	Avg./ Startup Time (ms)	% Int.
RL	4.00 / 2.80	92	47 / 46	0.00
r=3, k=1	2.02 / 1.24	116	284 / 359	15.78
r=3, k=2	1.74 / 1.14	116	297 / 505	17.16
r=3, k=4	0.96 / 0.78	128	469 / 1360	26.40
r=3, k=6	0.86 / 0.64	154	1438 / 3545	40.32
r=5, k=1	1.90 / 1.12	113	424 / 954	14.90
r=5, k=2	1.80 / 0.82	116	508 / 2312	17.12
r=5, k=4	0.90 / 0.76	127	1424 / 12592	25.94
r=5, k=6	1.02 / 0.66	134	6528 / 54842	29.56

Table 16: Instances *small-nd-100-0x.lp*, *x=01..10*
size 100 × 100, 2500 walls, 250 plants and 250 frogs
Variance RL (Plants / Frogs): 5.20 / 2.26
Variance FR (Plants / Frogs): 1.83 / 0.51

2.4.3 Deterministic (2-Step)

Config	Killed Plants/ Frogs	Steps	Avg./ Startup Time (ms)	% Int.
RL	1.50 / 0.00	9	0 / 0	0.00
r=3, k=1	*NA* / *NA*	0	*NA* / *NA*	*NA*
r=3, k=2	0.70 / 0.00	12	14 / 47	2.30
r=3, k=4	0.60 / 0.00	11	20 / 103	1.80
r=3, k=6	0.60 / 0.00	12	50 / 242	2.60
r=5, k=1	*NA* / *NA*	0	*NA* / *NA*	*NA*
r=5, k=2	0.70 / 0.00	12	30 / 81	2.30
r=5, k=4	0.60 / 0.00	11	38 / 142	1.80
r=5, k=6	0.60 / 0.00	12	86 / 369	2.10

Table 17: Instances *big-d-10-0x.lp*, *x=01..10*
size 10 × 10, 25 walls, 10 plants and 0 frogs
Variance RL (Plants / Frogs): 1.25 / 0.00
Variance FR (Plants / Frogs): 0.85 / 0.00

Config	Killed Plants/ Frogs	Steps	Avg./ Startup Time (ms)	% Int.
RL	2.70 / 0.00	23	0 / 0	0.00
r=3, k=1	*NA* / *NA*	0	*NA* / *NA*	*NA*
r=3, k=2	1.60 / 0.00	30	19 / 53	6.00
r=3, k=4	0.70 / 0.00	27	25 / 116	5.10
r=3, k=6	0.80 / 0.00	31	67 / 263	6.40
r=5, k=1	*NA* / *NA*	0	*NA* / *NA*	*NA*
r=5, k=2	1.60 / 0.00	30	36 / 91	6.00
r=5, k=4	0.80 / 0.00	27	45 / 157	4.60
r=5, k=6	0.70 / 0.00	26	106 / 396	3.70

Table 18: Instances *big-d-25-0x.lp*, *x=01..10*
size 25 × 25, 156 walls, 63 plants and 0 frogs
Variance RL (Plants / Frogs): 2.61 / 0.00
Variance FR (Plants / Frogs): 0.95 / 0.00

Config	Killed Plants/ Frogs	Steps	Avg./ Startup Time (ms)	% Int.
RL	6.60 / 0.00	46	0 / 0	0.00
r=3, k=1	NA / NA	0	NA / NA	NA
r=3, k=2	3.50 / 0.00	61	34 / 84	11.30
r=3, k=4	2.60 / 0.00	61	41 / 144	12.50
r=3, k=6	2.50 / 0.00	66	85 / 301	14.80
r=5, k=1	NA / NA	0	NA / NA	NA
r=5, k=2	3.20 / 0.00	61	58 / 132	11.30
r=5, k=4	2.60 / 0.00	60	67 / 199	12.00
r=5, k=6	2.60 / 0.00	57	131 / 462	9.60

Table 19: Instances *big-d-50-0x.lp*, *x=01..10*
size 50 × 50, 625 walls, 250 plants and 0 frogs
Variance RL (Plants / Frogs): 4.84 / 0.00
Variance FR (Plants / Frogs): 2.98 / 0.00

Config	Killed Plants/ Frogs	Steps	Avg./ Startup Time (ms)	% Int.
RL	11.20 / 0.00	86	0 / 0	0.00
r=3, k=1	NA / NA	0	NA / NA	NA
r=3, k=2	4.80 / 0.00	112	91 / 187	20.50
r=3, k=4	3.30 / 0.00	105	98 / 248	18.60
r=3, k=6	3.30 / 0.00	113	140 / 413	21.60
r=5, k=1	NA / NA	0	NA / NA	NA
r=5, k=2	4.80 / 0.00	112	134 / 264	20.60
r=5, k=4	3.60 / 0.00	104	143 / 336	17.40
r=5, k=6	3.10 / 0.00	102	203 / 569	15.50

Table 20: Instances *big-d-100-0x.lp*, *x=01..10*
size 100 × 100, 2500 walls, 1000 plants and 0 frogs
Variance RL (Plants / Frogs): 9.96 / 0.00
Variance FR (Plants / Frogs): 7.29 / 0.00

Config	Killed Plants/ Frogs	Steps	Avg./ Startup Time (ms)	% Int.
RL	0.80 / 0.00	9	0 / 0	0.00
r=3, k=1	NA / NA	0	NA / NA	NA
r=3, k=2	0.60 / 0.00	11	15 / 45	1.50
r=3, k=4	0.30 / 0.00	9	21 / 107	0.60
r=3, k=6	0.20 / 0.00	10	55 / 280	0.90
r=5, k=1	NA / NA	0	NA / NA	NA
r=5, k=2	0.60 / 0.00	11	30 / 81	1.50
r=5, k=4	0.30 / 0.00	9	38 / 151	0.70
r=5, k=6	0.30 / 0.00	9	91 / 393	0.50

Table 21: Instances *small-d-10-0x.lp*, *x=01..10*
size 10 × 10, 25 walls, 5 plants and 0 frogs
Variance RL (Plants / Frogs): 0.96 / 0.00
Variance FR (Plants / Frogs): 0.35 / 0.00

Config	Killed Plants/ Frogs	Steps	Avg./ Startup Time (ms)	% Int.
RL	1.00 / 0.00	24	0 / 0	0.00
r=3, k=1	NA / NA	0	NA / NA	NA
r=3, k=2	0.30 / 0.00	26	19 / 56	2.00
r=3, k=4	0.20 / 0.00	26	25 / 113	1.80
r=3, k=6	0.30 / 0.00	28	61 / 262	2.90
r=5, k=1	NA / NA	0	NA / NA	NA
r=5, k=2	0.30 / 0.00	26	37 / 93	2.00
r=5, k=4	0.20 / 0.00	26	44 / 155	1.90
r=5, k=6	0.20 / 0.00	25	100 / 391	1.40

Table 22: Instances *small-d-25-0x.lp*, *x=01..10*
size 25×25, 156 walls, 32 plants and 0 frogs
Variance RL (Plants / Frogs): 0.80 / 0.00
Variance FR (Plants / Frogs): 0.15 / 0.00

Config	Killed Plants/ Frogs	Steps	Avg./ Startup Time (ms)	% Int.
RL	2.90 / 0.00	44	0 / 0	0.00
r=3, k=1	NA / NA	0	NA / NA	NA
r=3, k=2	0.70 / 0.00	51	34 / 88	6.10
r=3, k=4	0.60 / 0.00	48	40 / 142	4.90
r=3, k=6	0.50 / 0.00	54	77 / 299	7.70
r=5, k=1	NA / NA	0	NA / NA	NA
r=5, k=2	0.70 / 0.00	51	57 / 141	6.10
r=5, k=4	0.60 / 0.00	48	64 / 201	4.50
r=5, k=6	0.50 / 0.00	47	121 / 436	4.30

Table 23: Instances *small-d-50-0x.lp*, *x=01..10*
size 50×50, 625 walls, 125 plants and 0 frogs
Variance RL (Plants / Frogs): 3.89 / 0.00
Variance FR (Plants / Frogs): 0.30 / 0.00

Config	Killed Plants/ Frogs	Steps	Avg./ Startup Time (ms)	% Int.
RL	5.50 / 0.00	81	0 / 0	0.00
r=3, k=1	NA / NA	0	NA / NA	NA
r=3, k=2	2.10 / 0.00	95	89 / 190	11.20
r=3, k=4	1.20 / 0.00	95	96 / 251	12.90
r=3, k=6	1.30 / 0.00	100	138 / 414	15.80
r=5, k=1	NA / NA	0	NA / NA	NA
r=5, k=2	2.10 / 0.00	95	132 / 271	11.20
r=5, k=4	1.00 / 0.00	92	138 / 336	10.90
r=5, k=6	0.60 / 0.00	89	194 / 584	9.50

Table 24: Instances *small-d-100-0x.lp*, *x=01..10*
size 100×100, 2500 walls, 500 plants and 0 frogs
Variance RL (Plants / Frogs): 2.25 / 0.00
Variance FR (Plants / Frogs): 2.36 / 0.00

2.4.4 Nondeterministic (2-Step)

Config	Killed Plants/ Frogs	Steps	Avg./ Startup Time (ms)	% Int.
RL	0.70 / 0.50	10	0 / 0	0.00
r=3, k=1	NA / NA	0	NA / NA	NA
r=3, k=2	0.46 / 0.46	15	16 / 63	3.32
r=3, k=4	0.38 / 0.50	17	34 / 165	4.60
r=3, k=6	0.44 / 0.50	28	136 / 463	11.14
r=5, k=1	NA / NA	0	NA / NA	NA
r=5, k=2	0.48 / 0.62	15	34 / 111	3.54
r=5, k=4	0.40 / 0.34	21	64 / 254	6.72
r=5, k=6	0.40 / 0.52	21	282 / 834	6.92

Table 25: Instances *big-nd-10-0x.lp*, *x=01..10*
size 10×10, 25 walls, 5 plants and 5 frogs
Variance RL (Plants / Frogs): 0.61 / 0.11
Variance FR (Plants / Frogs): 0.36 / 0.16

Config	Killed Plants/ Frogs	Steps	Avg./ Startup Time (ms)	% Int.
RL	1.30 / 1.66	23	1 / 1	0.00
r=3, k=1	NA / NA	0	NA / NA	NA
r=3, k=2	0.70 / 1.40	36	33 / 185	7.86
r=3, k=4	0.56 / 0.96	42	109 / 727	11.60
r=3, k=6	0.48 / 0.94	46	525 / 2373	13.74
r=5, k=1	NA / NA	0	NA / NA	NA
r=5, k=2	0.70 / 1.06	35	62 / 363	7.48
r=5, k=4	0.54 / 1.08	42	216 / 1969	12.24
r=5, k=6	0.56 / 0.78	45	1219 / 5058	13.20

Table 26: Instances *big-nd-25-0x.lp*, *x=01..10*
size 25×25, 156 walls, 31 plants and 32 frogs
Variance RL (Plants / Frogs): 0.41 / 0.96
Variance FR (Plants / Frogs): 0.35 / 0.67

Config	Killed Plants/ Frogs	Steps	Avg./ Startup Time (ms)	% Int.
RL	2.70 / 1.70	44	6 / 6	0.00
r=3, k=1	NA / NA	0	NA / NA	NA
r=3, k=2	1.04 / 1.18	65	67 / 323	15.02
r=3, k=4	0.96 / 0.84	73	184 / 1117	20.58
r=3, k=6	1.22 / 0.88	90	781 / 3070	29.70
r=5, k=1	NA / NA	0	NA / NA	NA
r=5, k=2	1.08 / 0.88	66	163 / 2130	15.42
r=5, k=4	1.10 / 0.86	75	827 / 12487	21.22
r=5, k=6	1.20 / 0.92	82	4347 / 53428	25.50

Table 27: Instances *big-nd-50-0x.lp*, *x=01..10*
size 50×50, 625 walls, 122 plants and 122 frogs
Variance RL (Plants / Frogs): 2.01 / 0.64
Variance FR (Plants / Frogs): 0.86 / 0.38

Config	Killed Plants/ Frogs	Steps	Avg./ Startup Time (ms)	% Int.
RL	4.30 / 4.04	89	95 / 95	0.00
r=3, k=1	NA / NA	0	NA / NA	NA
r=3, k=2	2.60 / 3.10	137	253 / 616	32.06
r=3, k=4	2.46 / 2.86	168	372 / 1487	51.60
r=3, k=6	2.28 / 2.98	202	980 / 3793	71.04
r=5, k=1	NA / NA	0	NA / NA	NA
r=5, k=2	2.50 / 2.92	141	389 / 2516	35.22
r=5, k=4	2.12 / 2.64	163	1003 / 13211	49.06
r=5, k=6	2.66 / 2.46	199	4506 / 56145	67.26

Table 28: Instances *big-nd-100-0x.lp*, *x=01..10*
size 100×100, 2500 walls, 500 plants and 500 frogs
Variance RL (Plants / Frogs): 5.61 / 2.67
Variance FR (Plants / Frogs): 2.50 / 2.96

Config	Killed Plants/ Frogs	Steps	Avg./ Startup Time (ms)	% Int.
RL	0.10 / 0.18	8	0 / 0	0.00
r=3, k=1	NA / NA	0	NA / NA	NA
r=3, k=2	0.10 / 0.12	10	16 / 56	0.90
r=3, k=4	0.00 / 0.10	12	26 / 135	1.80
r=3, k=6	0.02 / 0.14	14	85 / 873	3.78
r=5, k=1	NA / NA	0	NA / NA	NA
r=5, k=2	0.12 / 0.10	10	32 / 96	0.96
r=5, k=4	0.02 / 0.18	12	47 / 198	2.14
r=5, k=6	0.00 / 0.04	11	150 / 595	1.64

Table 29: Instances *small-nd-10-0x.lp*, *x=01..10*
size 10×10, 25 walls, 2 plants and 3 frogs
Variance RL (Plants / Frogs): 0.09 / 0.10
Variance FR (Plants / Frogs): 0.03 / 0.04

Config	Killed Plants/ Frogs	Steps	Avg./ Startup Time (ms)	% Int.
RL	0.80 / 0.52	21	0 / 0	0.00
r=3, k=1	NA / NA	0	NA / NA	NA
r=3, k=2	0.20 / 0.28	28	24 / 106	4.78
r=3, k=4	0.20 / 0.08	31	53 / 297	6.76
r=3, k=6	0.20 / 0.22	41	212 / 795	11.72
r=5, k=1	NA / NA	0	NA / NA	NA
r=5, k=2	0.22 / 0.32	28	46 / 194	4.86
r=5, k=4	0.22 / 0.34	32	104 / 524	7.16
r=5, k=6	0.20 / 0.20	34	481 / 1477	8.34

Table 30: Instances *small-nd-25-0x.lp*, *x=01..10*
size 25×25, 156 walls, 16 plants and 16 frogs
Variance RL (Plants / Frogs): 0.96 / 0.26
Variance FR (Plants / Frogs): 0.13 / 0.06

Config	Killed Plants/ Frogs	Steps	Avg./ Startup Time (ms)	% Int.
RL	2.10 / 0.94	50	3 / 3	0.00
r=3, k=1	NA / NA	0	NA / NA	NA
r=3, k=2	0.80 / 0.58	65	54 / 305	9.82
r=3, k=4	0.58 / 0.54	73	129 / 1039	15.40
r=3, k=6	0.64 / 0.32	81	482 / 2810	19.84
r=5, k=1	NA / NA	0	NA / NA	NA
r=5, k=2	0.90 / 0.56	66	99 / 801	10.64
r=5, k=4	0.68 / 0.40	73	299 / 3157	15.52
r=5, k=6	0.68 / 0.48	74	1462 / 12123	15.48

Table 31: Instances *small-nd-50-0x.lp*, *x=01..10*
size 50×50, 625 walls, 62 plants and 63 frogs
Variance RL (Plants / Frogs): 2.29 / 0.27
Variance FR (Plants / Frogs): 0.81 / 0.19

Config	Killed Plants/ Frogs	Steps	Avg./ Startup Time (ms)	% Int.
RL	4.00 / 2.80	92	47 / 46	0.00
r=3, k=1	NA / NA	0	NA / NA	NA
r=3, k=2	1.56 / 1.24	122	173 / 507	21.22
r=3, k=4	0.92 / 0.80	137	257 / 1367	32.14
r=3, k=6	0.64 / 0.82	157	727 / 3826	43.62
r=5, k=1	NA / NA	0	NA / NA	NA
r=5, k=2	1.70 / 1.08	123	280 / 2381	22.46
r=5, k=4	0.84 / 0.74	134	720 / 12983	30.02
r=5, k=6	0.98 / 0.74	143	3334 / 55767	35.60

Table 32: Instances *small-nd-100-0x.lp*, *x=01..10*
size 100×100, 2500 walls, 250 plants and 250 frogs
Variance RL (Plants / Frogs): 5.20 / 2.26
Variance FR (Plants / Frogs): 1.06 / 0.46

3 Pacman

3.1 Data

We tested the implementation of the framework on 3 different instances found as part of the Pacman Berkeley AI code (`http://ai.berkeley.edu/project_overview`). The RL algorithm uses two different feature extractors to train the RL agent. The *HungryExtractor* can distinguish between ghosts that are in a scared state, while the *SimpleExtractor* cannot. These feature extractors are implemented in `Code/pacman/featureExtractors.py` and are the same as the once used by Neufeld et. al 2022. The results of training the RL weights for both of these extractors over 1000 runs on each layout are saved as `.pkl` files in `Data/Pacman/layouts/learning`.

To retrain these weights, one can execute the command `python3 pac.py`, which restarts the training process for all layout files in the folder `layouts`.

3.2 Setup

Each instance of Pacman can be executed with the following command:

```
python3 pacman.py -m <mode> -p ApproximateQAgent -a extractor=<extractor> -x 0 -n 1
    --radius <r> --horizon <h> -l <path-to-layout>
```

with <mode> being a number between 1 and 3 (1 = run RL, 2 = run the framework with the vegan norm, 3 = run the framework with the vegetarian norm). <k> and <r> are the horizon and the window radius and are only considered if mode 2 or 3 is selected. -p ApproximateQAgent is the implementation of our RL policy that is used in all our tests. <extractor> defines which feature extractor should be used, and can only be HungryExtractor or SimpleExtractor. -x 0 -n 1 tells the program to not train (0) but test on exactly 1 run. Finally, <path-to-layout> is the path to the stored instance. The program expects that there is a pre-trained .pkl file in the directory <path-to-layout>/learning/<extractor> named equally to the instance.

There is an optional argument -q, which tells the program to suppress the visualization of the agent playing the game. Finally, the optional argument -f <seed> fixes the random seed to a specific value. In our testing, we used the starting random seed 42 to generate new random seeds for every test.

3.3 Logic Program

The following code corresponds to the logic program Code/pacman/program.lp, which is the logic program used in our framework for all instances of the Pacman game. The logic program is grounded only once per instance. To reflect the state of the window around the agent, the external atoms are then changed dynamically in Code/pacman/clingohelper.py in each step. Hence, Pacman includes all optimizations laid out in Section "Program Optimization" of the main document.

To account for the different norms (vegan and vegetarian), Code/pacman/clingohelper.py excludes setting the external atoms containing information about the blue ghost in the vegetarian case. Therefore, the logic program is "blind" regarding this ghost and will not hinder Pacman from eating it.

The rewards from following policy preferences and penalties from violating a norm are encoded in the logic program. In our implementation, norm violations are accounted at a higher priority level (2), than rewards (1). The reward of an action is based on policy preference $(0-3)$, where 0 is the preferred action by the policy and a higher number indicates a less preferred action. Earlier violations are weighted higher to discourage immediate violations.

```
%%% Windowing %%%
% This section implements MSS and the windowing technique described in section
% "Program Optimization" in the main document

#external gcol(G,C,0) : G=0..ghosts-1, C=-radius..radius.
#external grow(G,R,0) : G=0..ghosts-1, R=-radius..radius.
#external goutside(G) : G=0..ghosts-1.
#external wall(C,R) : C=-radius..radius, R=-radius..radius.

% rewards for the actions of the agent
#external action(A,R) : A=0..4, R=0..4.

% player always starts at the 0 position
pcol(0,0).
prow(0,0).


%%% P_gen %%%
% This section corresponds to the subprogram P_gen described in section
% "ASP Framework" in the main document

% movement of the agent
pmove(0, T) | pmove(1, T) | pmove(2,T) :- T=1..horizon - 1.
prow(R + 1,T) | prow(R - 1,T) :- prow(R,T-1), pmove(0, T).
pcol(C,T) :- pcol(C,T-1), pmove(0, T).
pcol(C + 1,T) | pcol(C - 1,T) :- pcol(C,T-1), pmove(1, T).
prow(R,T) :- prow(R,T-1), pmove(1, T).

% hard restrictions (agent cannot leave window or move into a wall)
:- prow(R,T), pcol(C,T), wall(C,R).
sat :- grow(G,R,T), gcol(G,C,T), wall(C,R).

%%% P_check %%%
% This section corresponds to the subprogram P_check described in section
% "ASP Framework" in the main document

% movement of the ghosts
gmove(G,0, T) | gmove(G,1, T) :- G=0..ghosts-1, T=1..horizon - 1.
```

```
grow(G,R + 1,T) | grow(G,R - 1,T) :- grow(G,R,T-1), gmove(G,0, T).
gcol(G,C,T) :- gcol(G,C,T-1), gmove(G,0, T).
gcol(G,C + 1,T) | gcol(G,C - 1,T) :- gcol(G,C,T-1), gmove(G,1, T).
grow(G,R,T) :- grow(G,R,T-1), gmove(G,1, T).

% check saturation criteria
ok(0).
ok(0,G) :- G=0..ghosts-1.
sat :- ok(horizon-1).
:- not sat.

% saturate to M_sat as described in section "ASP Framework -> Saturation
% technique" in the main document
gcol(G,C,T) :- C = -radius..radius, sat, T=1..horizon-1, G=0..ghosts-1.
grow(G,R,T) :- R = -radius..radius, sat, T=1..horizon-1, G=0..ghosts-1.
gmove(G,0,T) :- G=0..ghosts-1, T=1..horizon-1, sat.
gmove(G,1,T) :- G=0..ghosts-1, T=1..horizon-1, sat.

% each ghost either needs to be safely away from player or give penalty
% also consider ghosts previous position here, as pacman cannot "swap places"
% with ghosts like the agent can in gardener
ok(T,G,0) :- pcol(C,T), gcol(G,C',T-1), C != C', ok(T-1).
ok(T,G,1) :- pcol(C,T), gcol(G,C',T), C != C', ok(T-1).
ok(T,G,0) :- prow(R,T), grow(G,R',T-1), R != R', ok(T-1).
ok(T,G,1) :- prow(R,T), grow(G,R',T), R != R', ok(T-1).
ok(T,G) :- ok(T,G,0), ok(T,G,1).
ok(T,G) :- goutside(G), T=1..horizon-1.
good(T,G) | bad(T,G) :- G=0..ghosts-1, T=1..horizon-1.
ok(T,G) :- bad(T,G).
ok(T) :- {ok(T,G) : G=0..ghosts-1} = ghosts, ok(T-1).

% penalty for violating the norm of eating a ghost
penalty(P, T) :- P = horizon - T, bad(T,F).

% encode the reward of the taken action
reward(X) :- action(0,X), pcol(0,0), pcol(1,1).
reward(X) :- action(1,X), pcol(0,0), pcol(-1,1).
reward(X) :- action(2,X), prow(0,0), prow(1,1).
reward(X) :- action(3,X), prow(0,0), prow(-1,1).
reward(X) :- action(4,X), pcol(0,0), pcol(0,1), prow(0,0), prow(0,1).


%%% Optimization %%%
% Here (in addition with weak constraints above) we implement the utilitarian
% policy fix described in section "Policy Fixing" in the main document.

% optimize for minimum reward (best action has a reward of 0, higher is worse)
% and minimal penalty
#minimize {S@1 : reward(S)}.
#minimize {S@2,T : penalty(S, T)}.

#show pcol/2.
#show prow/2.
```

3.4 Results

The results achieved are divided in subsections depending on the tested instance. Each of the tables in their respective subsections lists the results achieved computing a k-policy fix, with various parameters r and k, in comparison with a RL baseline. The value labelled "% Int." hereby describes the percentage of steps in which the framework deviates from the most preferred action by the RL policy.

Each row shows the average values achieved in the instance with the corresponding parameter configuration over 1000 runs. In addition to these results, each table displays the variance of killed blue and other ghosts for the RL baseline, as well as the combined average over all applications of the framework.

Over all our tests in Pacman we saw an increase in norm compliance when using our framework compared to the RL baseline at the cost of a reduction in average score. Win percentage remained steady or increased with k. The average computation time was $54ms$ and $60ms$ at worst when using the `HungryExtractor` and `SimpleExtractor` respectively.

3.4.1 Small

Config	Ghosts eaten Blue/Other	% Won [Avg. Score]	Avg./ Startup Time (ms)	% Int.
RL	0.03 / 0.04	83 [779]	0 / 0	0.00
r=3, k=1	0.01 / 0.01	85 [798]	3 / 6	0.09
r=3, k=2	0.01 / 0.01	87 [828]	4 / 9	0.57
r=3, k=4	0.01 / 0.01	91 [870]	9 / 18	1.47
r=3, k=6	0.00 / 0.00	90 [849]	31 / 34	1.74
r=5, k=1	0.01 / 0.01	84 [788]	6 / 9	0.11
r=5, k=2	0.01 / 0.01	89 [852]	6 / 14	0.50
r=5, k=4	0.00 / 0.00	93 [889]	14 / 28	1.79
r=5, k=6	0.00 / 0.00	92 [878]	60 / 54	2.81

Table 33: Instance *originalClassic.lay* with *HungryExtractor* and *Vegan* norm applied in ASP
Variance RL (blue / other): 0.03484 / 0.03578
Variance FR (blue / other): 0.00670 / 0.00584

Config	Ghosts eaten Blue/Other	% Won [Avg. Score]	Avg./ Startup Time (ms)	% Int.
RL	0.03 / 0.04	83 [779]	0 / 0	0.00
r=3, k=1	0.01 / 0.01	87 [832]	3 / 6	0.18
r=3, k=2	0.01 / 0.00	83 [782]	4 / 9	0.25
r=3, k=4	0.00 / 0.00	82 [769]	7 / 18	0.40
r=3, k=6	0.00 / 0.00	85 [796]	20 / 34	0.45
r=5, k=1	0.02 / 0.01	86 [818]	6 / 10	0.23
r=5, k=2	0.01 / 0.00	87 [822]	6 / 14	0.21
r=5, k=4	0.00 / 0.00	82 [766]	10 / 28	0.43
r=5, k=6	0.00 / 0.00	85 [797]	30 / 53	0.44

Table 34: Instance *originalClassic.lay* with *HungryExtractor* and *Vegetarian* norm applied in ASP
Variance RL (blue / other): 0.03484 / 0.03578
Variance FR (blue / other): 0.00596 / 0.00299

Config	Ghosts eaten Blue/Other	% Won [Avg. Score]	Avg./ Startup Time (ms)	% Int.
RL	0.02 / 0.02	83 [779]	0 / 0	0.00
r=3, k=1	0.01 / 0.01	85 [800]	3 / 6	0.03
r=3, k=2	0.01 / 0.02	88 [842]	4 / 9	0.45
r=3, k=4	0.01 / 0.01	92 [875]	9 / 18	1.36
r=3, k=6	0.01 / 0.01	91 [864]	31 / 34	1.77
r=5, k=1	0.01 / 0.01	85 [806]	6 / 9	0.03
r=5, k=2	0.01 / 0.02	89 [845]	6 / 14	0.46
r=5, k=4	0.00 / 0.00	93 [884]	14 / 28	1.70
r=5, k=6	0.00 / 0.00	93 [888]	60 / 54	2.96

Table 35: Instance *originalClassic.lay* with *SimpleExtractor* and *Vegan* norm applied in ASP
Variance RL (blue / other): 0.01482 / 0.02254
Variance FR (blue / other): 0.00781 / 0.00880

Config	Ghosts eaten Blue/Other	% Won [Avg. Score]	Avg./ Startup Time (ms)	% Int.
RL	0.02 / 0.02	83 [779]	0 / 0	0.00
r=3, k=1	0.01 / 0.01	87 [832]	3 / 6	0.10
r=3, k=2	0.00 / 0.00	84 [787]	4 / 9	0.23
r=3, k=4	0.00 / 0.00	82 [765]	7 / 18	0.46
r=3, k=6	0.00 / 0.00	84 [790]	19 / 34	0.41
r=5, k=1	0.01 / 0.01	85 [810]	6 / 9	0.13
r=5, k=2	0.01 / 0.00	86 [806]	6 / 14	0.19
r=5, k=4	0.00 / 0.00	84 [786]	10 / 28	0.41
r=5, k=6	0.00 / 0.00	84 [785]	30 / 53	0.43

Table 36: Instance *originalClassic.lay* with *SimpleExtractor* and *Vegetarian* norm applied in ASP
Variance RL (blue / other): 0.01482 / 0.02254
Variance FR (blue / other): 0.00423 / 0.00361

3.4.2 Medium

Config	Ghosts eaten Blue/Other	% Won [Avg. Score]	Avg./ Startup Time (ms)	% Int.
RL	0.83 / 0.82	92 [1559]	0 / 1	0.00
r=3, k=1	0.02 / 0.03	91 [1204]	3 / 6	5.98
r=3, k=2	0.01 / 0.01	92 [1224]	4 / 9	6.17
r=3, k=4	0.02 / 0.01	94 [1244]	8 / 18	6.56
r=3, k=6	0.01 / 0.01	94 [1242]	23 / 34	6.39
r=5, k=1	0.01 / 0.01	91 [1209]	5 / 10	6.27
r=5, k=2	0.02 / 0.02	93 [1236]	6 / 14	6.23
r=5, k=4	0.01 / 0.01	95 [1262]	11 / 28	6.56
r=5, k=6	0.01 / 0.01	95 [1257]	41 / 59	6.74

Table 37: Instance *originalClassic.lay* with *HungryExtractor* and *Vegan* norm applied in ASP
Variance RL (blue / other): 0.45307 / 0.49224
Variance FR (blue / other): 0.01295 / 0.01223

Config	Ghosts eaten Blue/Other	% Won [Avg. Score]	Avg./ Startup Time (ms)	% Int.
RL	0.83 / 0.82	92 [1559]	0 / 1	0.00
r=3, k=1	0.78 / 0.00	90 [1343]	3 / 6	3.34
r=3, k=2	0.82 / 0.00	90 [1345]	4 / 9	3.22
r=3, k=4	0.80 / 0.00	89 [1344]	7 / 18	3.16
r=3, k=6	0.78 / 0.00	89 [1328]	16 / 34	3.39
r=5, k=1	0.79 / 0.00	91 [1362]	5 / 10	3.19
r=5, k=2	0.84 / 0.00	90 [1364]	6 / 14	3.43
r=5, k=4	0.82 / 0.00	90 [1352]	9 / 28	3.38
r=5, k=6	0.77 / 0.00	90 [1340]	22 / 57	3.25

Table 38: Instance *originalClassic.lay* with *HungryExtractor* and *Vegetarian* norm applied in ASP
Variance RL (blue / other): 0.45307 / 0.49224
Variance FR (blue / other): 0.46705 / 0.00000

Config	Ghosts eaten Blue/Other	% Won [Avg. Score]	Avg./ Startup Time (ms)	% Int.
RL	0.02 / 0.02	90 [1208]	0 / 1	0.00
r=3, k=1	0.01 / 0.01	88 [1168]	3 / 6	1.37
r=3, k=2	0.01 / 0.01	93 [1226]	4 / 9	1.49
r=3, k=4	0.00 / 0.01	94 [1239]	8 / 18	1.61
r=3, k=6	0.01 / 0.01	94 [1251]	23 / 34	1.76
r=5, k=1	0.02 / 0.02	89 [1190]	5 / 10	1.37
r=5, k=2	0.01 / 0.01	92 [1206]	6 / 14	1.38
r=5, k=4	0.01 / 0.00	96 [1281]	11 / 28	1.74
r=5, k=6	0.01 / 0.01	95 [1261]	40 / 59	2.14

Table 39: Instance *originalClassic.lay* with *SimpleExtractor* and *Vegan* norm applied in ASP
Variance RL (blue / other): 0.01671 / 0.02056
Variance FR (blue / other): 0.00953 / 0.01014

Config	Ghosts eaten Blue/Other	% Won [Avg. Score]	Avg./ Startup Time (ms)	% Int.
RL	0.02 / 0.02	90 [1208]	0 / 1	0.00
r=3, k=1	0.04 / 0.00	89 [1183]	3 / 6	0.74
r=3, k=2	0.02 / 0.00	89 [1187]	4 / 9	0.78
r=3, k=4	0.03 / 0.00	90 [1191]	7 / 18	0.84
r=3, k=6	0.03 / 0.00	90 [1187]	16 / 34	0.79
r=5, k=1	0.03 / 0.00	90 [1201]	5 / 9	0.74
r=5, k=2	0.02 / 0.00	89 [1180]	6 / 14	0.77
r=5, k=4	0.04 / 0.00	88 [1172]	9 / 28	0.77
r=5, k=6	0.03 / 0.00	89 [1178]	22 / 57	0.82

Table 40: Instance *originalClassic.lay* with *SimpleExtractor* and *Vegetarian* norm applied in ASP
Variance RL (blue / other): 0.01671 / 0.02056
Variance FR (blue / other): 0.02983 / 0.00075

3.4.3 Original

Config	Ghosts eaten Blue/Other	% Won [Avg. Score]	Avg./ Startup Time (ms)	% Int.
RL	0.60 / 1.68	81 [2581]	0 / 0	0.00
r=3, k=1	0.02 / 0.05	76 [2014]	4 / 7	4.11
r=3, k=2	0.01 / 0.04	79 [2056]	5 / 13	4.21
r=3, k=4	0.01 / 0.04	83 [2124]	11 / 32	4.31
r=3, k=6	0.01 / 0.04	83 [2120]	32 / 73	4.29
r=5, k=1	0.02 / 0.07	78 [2023]	7 / 13	4.10
r=5, k=2	0.02 / 0.06	83 [2126]	8 / 22	4.23
r=5, k=4	0.01 / 0.04	85 [2156]	16 / 54	4.22
r=5, k=6	0.01 / 0.02	88 [2212]	54 / 114	4.57

Table 41: Instance *originalClassic.lay* with *HungryExtractor* and *Vegan* norm applied in ASP
Variance RL (blue / other): 0.49840 / 1.49302
Variance FR (blue / other): 0.01465 / 0.04559

Config	Ghosts eaten Blue/Other	% Won [Avg. Score]	Avg./ Startup Time (ms)	% Int.
RL	0.60 / 1.68	81 [2581]	0 / 0	0.00
r=3, k=1	0.56 / 0.03	78 [2139]	4 / 7	3.28
r=3, k=2	0.53 / 0.03	80 [2181]	5 / 13	3.25
r=3, k=4	0.53 / 0.03	81 [2180]	11 / 32	3.51
r=3, k=6	0.52 / 0.04	81 [2187]	27 / 73	3.31
r=5, k=1	0.55 / 0.02	78 [2145]	7 / 13	3.18
r=5, k=2	0.52 / 0.04	82 [2202]	8 / 22	3.34
r=5, k=4	0.52 / 0.01	82 [2210]	15 / 54	3.21
r=5, k=6	0.54 / 0.02	83 [2243]	43 / 114	3.48

Table 42: Instance *originalClassic.lay* with *HungryExtractor* and *Vegetarian* norm applied in ASP
Variance RL (blue / other): 0.49840 / 1.49302
Variance FR (blue / other): 0.45433 / 0.02972

Config	Ghosts eaten Blue/Other	% Won [Avg. Score]	Avg./ Startup Time (ms)	% Int.
RL	0.01 / 0.05	79 [2090]	0 / 0	0.00
r=3, k=1	0.03 / 0.05	77 [2019]	4 / 7	0.87
r=3, k=2	0.01 / 0.04	78 [2032]	5 / 13	0.93
r=3, k=4	0.02 / 0.04	82 [2102]	11 / 32	1.13
r=3, k=6	0.01 / 0.03	83 [2127]	32 / 73	1.16
r=5, k=1	0.01 / 0.05	77 [2018]	7 / 13	0.92
r=5, k=2	0.01 / 0.05	80 [2072]	8 / 22	1.02
r=5, k=4	0.01 / 0.03	86 [2190]	16 / 54	1.20
r=5, k=6	0.00 / 0.03	90 [2245]	53 / 114	1.41

Table 43: Instance *originalClassic.lay* with *SimpleExtractor* and *Vegan* norm applied in ASP
Variance RL (blue / other): 0.01478 / 0.04840
Variance FR (blue / other): 0.01344 / 0.03944

Config	Ghosts eaten Blue/Other	% Won [Avg. Score]	Avg./ Startup Time (ms)	% Int.
RL	0.01 / 0.05	79 [2090]	0 / 0	0.00
r=3, k=1	0.03 / 0.03	78 [2046]	4 / 7	0.74
r=3, k=2	0.02 / 0.03	77 [1996]	5 / 13	0.77
r=3, k=4	0.03 / 0.03	79 [2052]	10 / 32	0.80
r=3, k=6	0.02 / 0.03	81 [2091]	27 / 73	0.88
r=5, k=1	0.03 / 0.04	77 [2031]	7 / 13	0.74
r=5, k=2	0.03 / 0.02	79 [2061]	8 / 22	0.75
r=5, k=4	0.02 / 0.02	82 [2096]	15 / 54	0.87
r=5, k=6	0.03 / 0.03	82 [2109]	42 / 114	1.01

Table 44: Instance *originalClassic.lay* with *SimpleExtractor* and *Vegetarian* norm applied in ASP
Variance RL (blue / other): 0.01478 / 0.04840
Variance FR (blue / other): 0.02605 / 0.02831

4 CCN

4.1 Data

All CCN experiments use the same network topology that is visualized in Figure 1. The five different scenarios found in `Data/CCN` instead differ in the amount and frequency of the requested packages by the five customers $C1$ - $C5$. Producer $P1$ fulfills packages with the prefix `/movie`, producer $P2$ fulfills packages with the prefix `/news` and producer $P3$ fulfills packages with the prefix `/gov`. All routers $R1$ - $R5$ run the *LRU* policy by default, with only router $R3$ switching between *LRU* and our framework in combination with *LRU* depending on the setting. A detailed description of the packages requested by the consumers in each scenario can be found in Table 45.

The investigated router ($R3$) has a cache size of 18. All other nodes have a cache size of 10. In all scenarios, the packages with the prefix `/news` are marked as low priority, while packages with the prefix `/gov` are marked as high priority.

Each scenario has a runtime of 30 seconds, with consumers requesting packages only within the first 20 seconds.

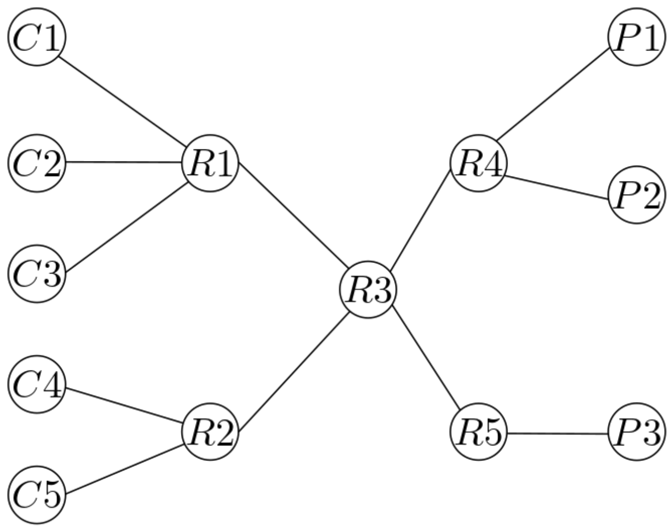

Figure 1: Network topology used for experiments with CCN

Instance	C1		C2		C3		C4		C5	
	C	F	C	F	C	F	C	F	C	F
scenario 1	/movie/lotr	5	/movie/lotr	2	/movie/hp	3	/news/14-08	2	/gov/emergency	1
scenario 2	/movie/lotr	5	/news/15-08	2	/news/14-08	3	/news/14-08	1	/gov/emergency	1
scenario 3	/gov/emergency	3	/movie/hp	2	/movie/hp	3	/news/14-08	1	/gov/emergency	1
scenario 4	/movie/hp	5	/movie/hp	2	/news/14-08	3	/gov/emergency	2	/movie/hp	1
scenario 5	/gov/emergency	3	/news/15-08	2	/news/14-08	3	/news/14-08	5	/gov/law	1

Table 45: Content (C) and frequency (F), measured in requests per second, of packages requested by the consumers in different scenarios

4.2 Setup

Running the instances of this experiment requires the software *ndnSim 2.9*, which is available at `https://ndnsim.net`. After ndnSim is built, there are a few files that need to be changed, in order for ndnSim to communicate with our framework. All files that need to be replaced can be found in the ndnSim build folder at `ndnSIM/ns-3/src/ndnSIM/NFD/daemon/table`. The three updated files that need to be changed can be found in our appendix at `Code/ccn`. When successfully changed, we can use the new policy labeled `nfd::cs::rl`.

Upon use of this policy, ndnSim expects to find the relevant Python code in its build folder at the path `ndnSIM/ns-3/src/ndnSIM/NFD/daemon/table/framework`. All scripts required to be in this folder can be found in our appendix at `Code/ccn/framework`.

To run one of the scenarios included in `Data/ccn` copy the file and place it in the ndnSim build folder at `ndnSIM/ns-3/src/ndnSIM/examples`. Afterward, the examples can be run from the command line inside the ndnSim build folder using the following command:

```
sudo ./waf --run=ndn-custom-example-<n>
```

where `<n>` is a number between 1 and 5 defining the experiment that should be run. After the experiment is finished, the amount of cache hits is output to the console. To output more details about the cache performance during the experiment, call the following command after the experiment is finished and before another experiment is started:

```
python3 ndnSIM/ns-3/src/ndnSIM/NFD/daemon/table/framework/run.py
```

The horizon k, encoded in `Code/ccn/framework/run.py`, is set to 3 for all scenarios. The threshold age for low priority packages is set to 10 and the threshold age for high priority packages is set to 20 (both encoded in `Code/ccn/framework/run.py`).

To use the original LRU policy instead of our framework in combination with LRU uncomment Line 68 in `Code/ccn/framework/run.py`.

4.3 Logic Program

The following code corresponds to the logic program `Code/ccn/framework/program.lp`, which is the logic program used in our framework for all scenarios of the CCN experiment. `Code/ccn/framework/clingohelper.lp` complements this logic program in each step with information about the current state of the cache and incoming packages, as well as information about the policy preferences of LRU. Because of limitations in the connection between ndnSim and native Python code of our framework, the logic program is grounded anew in every time step.

The rewards from following policy preferences and penalties from violating a norm are encoded in the logic program. In our implementation, all rewards and penalties are accounted on the same priority level. The penalty received for violating a norm is equal to the age of the violating content. The reward of an action is based on policy preference $(0 - 18)$, where 0 is the preferred action by the policy and a higher number indicates a less preferred action.

```
% external constants: size, prefixes, horizon, high_prio_age, low_prio_age

%%% P_gen %%%
% This section corresponds to the subprogram P_gen described in section
% "ASP Framework" in the main document

% Encode the ranking of the executed action
% The ranking is missing here and dynamically generated and inserted depending
% on the current state of the cache
action_rank(R) :- action(0,A), ranking(A,R).

% inertia and state transition rules for content in cells
cell_content(T+1,C,P) :- action(T,C), C < size, inc_cell(T,P), T=0..horizon-1.
cell_content(T+1,C,P) :- cell_content(T,C,P), not action(T,C), T=0..horizon-1.

% inertia and state transition rules for the age of cells
cell_age(T+1,C,0) :- action(T,C), C < size, cell_age(T,C,A), T=0..horizon-1.
cell_age(T+1,C,A+1) :- not action(T,C), cell_age(T,C,A), T=0..horizon-1.

%%% P_check %%%
% This section corresponds to the subprogram P_check described in section
% "ASP Framework" in the main document
% The guessing rules are missing here and are dynamically generated
% and inserted depending on the instance

% Low Priority Norm is not violated if content is below threshold or not
% low priority
ok_low(T,C) :- cell_age(T,C,A), cell_content(T,C,P), low_prio(P), A <= low_prio_age.
ok_low(T,C) :- cell_content(T,C,P), not low_prio(P).

% High Priority Norm is not violated if content is above threshold or not
% high priority
ok_high(T,C) :- cell_age(T,C,A), cell_content(T,C,P), high_prio(P), A >= high_prio_age.
ok_high(T,C) :- cell_age(T,C,A), cell_content(T,C,P), high_prio(P), A < high_prio_age, action(
    T, C'), C != C'.
```

```
ok_high(T,C) :- cell_content(T,C,P), not high_prio(P).
% Count norm violations in any given time step
ok(T) :- {ok_low(T,C) : C=0..size-1} = size, {ok_high(T,C) : C=0..size-1} = size, ok(T-1).
good_low(T,C) | bad_low(T,C) :- C=0..size-1, T=1..horizon-1.
good_high(T,C) | bad_high(T,C) :- C=0..size-1, T=1..horizon-1.
ok_low(T,C) :- bad_low(T,C).
ok_high(T,C) :- bad_high(T,C).
% saturate to M_sat as described in section "ASP Framework -> Saturation
% technique" in the main document
inc_cell(T,P) :- sat, P=0..prefixes-1, T=1..horizon.
cell_content(T,C,P) :- sat, C=0..size, P=0..prefixes-1, T=1..horizon.
% check saturation criteria
ok(0).
ok_low(0,C) :- C=0..size-1.
ok_high(0,C) :- C=0..size-1.
sat :- ok(horizon-1).
:- not sat.
% Compute penalties for violating norms dependent on age as described in
% section "Experiments->CCN Caching" of the main document
penalty(A, T, C) :- bad_low(T,C), cell_age(T,C,A).
penalty(A, T, C) :- bad_high(T,C), cell_age(T,C,A).
%%% Optimization %%%
% Here (in addition with weak constraints above) we implement the utilitarian
% policy fix described in section "Policy Fixing" in the main document.
% optimize for minimum reward (best action has a reward of 0, higher is worse)
% and minimal penalty
#minimize {S@1,T,C : penalty(S, T, C)}.
#minimize {R@1 : action_rank(R)}.
#show action/2.
#show ranking/2.
#show action_rank/1.
```

4.4 Results

The results achieved are shown in Table 46, where each row corresponds to one of the above defined scenarios and is tested with our framework in combination with an LRU policy and only using the LRU policy. The results show a big increase in norm compliance using our framework compared to the LRU baseline. In scenario 3, the 88 norm violations in case high are due to the many priority package requests sent in this scenario (see Table 45), that fill up the cache, leading necessarily to norm violations.

instance [packages]	LRU norm violations low / high	LRU cache hits	ASP + LRU norm violations low / high	ASP + LRU cache hits
scenario 1 [385]	504 / 20	2	52 / 0	2
scenario 2 [358]	1419 / 20	2	76 / 0	2
scenario 3 [289]	271 / 95	7	52 / 88	8
scenario 4 [389]	744 / 53	4	62 / 0	5
scenario 5 [419]	2352 / 99	10	128 / 0	7

Table 46: Results of CCN Caching