1 Experiments

In this section we implement our algorithm on three domains: (1) navigation in grid world, (2) cart pole environment, (3) mountain car environment. Cart pole environment and mountain car environment are imported from OpenAI gym `https://github.com/openai/gym/wiki/CartPole-v0` and `https://github.com/openai/gym/wiki/MountainCar-v0`. We want to use the three environment settings to show that by using our safety-aware apprenticeship learning algorithm, the learnt policy is guaranteed to be safe in respect to the given safety specification; the learnt policy has comparable and even better performance in comparison with the one that directly learnt vi apprenticeship learning. We also show that by using model checking tools, PRISM and COMICS, the runtime is acceptable in comparison with using Monte Carlo method.

1.1 Navigation in Grid World

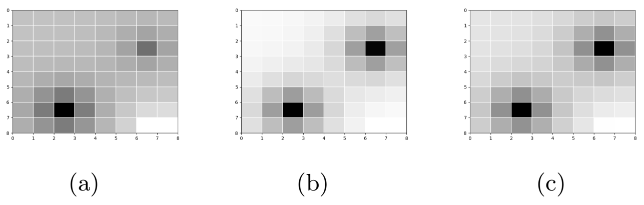
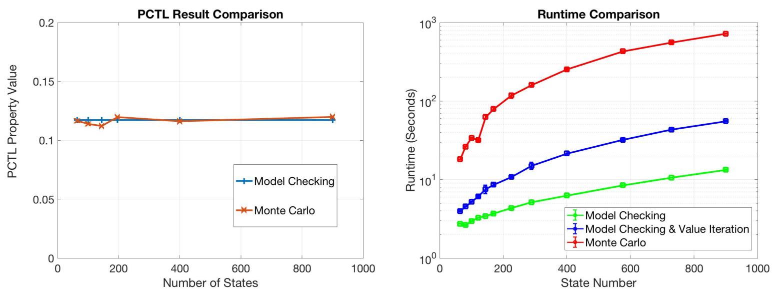


Fig. 1. (a)$p^* = 0.9$. The learnt policy has PCTL property of 0.12050272329 according to model checking result. It is optimal to a reward function which assign lower rewards to the unsafe states than states nearby. (c) $p^* = 0.4$. The learnt policy has PCTL property of 0.109697761255 according to model checking result. The reward function assigns even lower rewards to the unsafe states, thus the contrast between unsafe states and other states is increased. (d)$p^* = 0.1$. The learnt policy has PCTL property of 0.0420773618519 according to model checking result. The reward function assigns such low rewards on the unsafe states that the states nearby are also assigned with low rewards due to the radial features.

Our first experiment is an extension on the 2D navigation task in section 3. Now different safety specifications with different p^* is given, the agent is able to refine its policies according to the result of model checking. Fig. 1 shows the different reward mappings which the policies learnt by safety-aware apprenticeship learning algorithm when p^* ranges from 1.0 to 0.1. Changes in reward mapping implicitly reflects how learnt weight vectors change with p^*. The algorithm is trying to make a trade-off between keeping safety and finishing the task. It can find a weight vector which can induce high positive rewards on goal states so as to reduce the probability of stepping into other areas including the unsafe states. But when safety specification is too critical, the algorithm would have to find a weight vector which assign low rewards in unsafe states and states around, so that the agent will have a even lower probability of moving into the unsafe areas. As a result, agent has to focus more on avoiding moving into unsafe areas than reaching the goal area.



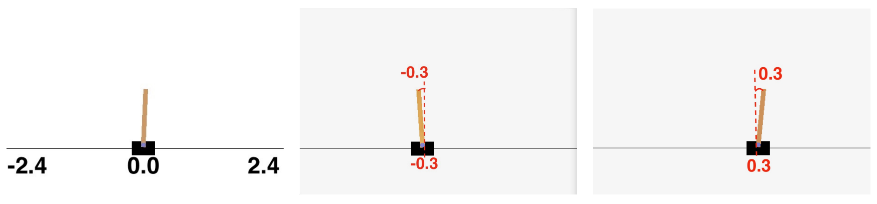
(a) (b)

Fig. 2. The comparison between model checking and Monte Carlo method. (a) PCTL property result comparison in spanning state space. The checked policy is the policy learnt via Apprenticeship Learning. Both results are close to each other. (c) Runtime comparison in spanning state space. The y axis is on log scale. Red line is the runtime of using Monte Carlo method. Green line is the runtime of model checking PCTL property and counterexample generation by using model checking tool. Blue line is the total runtime of using model checking tool and value iteration.

To illustrate the advantage of using model checking tool, we compare its runtime with Monte Carlo method. The test environment is grid worlds with varying state spaces from 64 to 900 while the locations of unsafe area and goal area stay the same as the grid world in Fig. **??**. Given same policies, we evaluate the runtime of safety verification and counterexample generation by using Monte Carlo method and model checking respectively. As Monte Carlo method can generate expected feature counts at the same time, we also measured the total runtime of model checking and calculating expected features by value iteration. As shown in Fig. 2, although the PCTL property verified by Monte Carlo method are close to model checking results, the runtime of Monte Carlo method is times longer than that of model checking and it increases with a higher rate as the state space is augmented.

1.2 Cart Pole Environment

In grid world it's hard to evaluate the performance that the agent may sacrifice for safety. In our second experiment, we implemented the Safety-Aware Apprenticeship Learning algorithm in a cart pole environment, where the goal is to keep



(a) (b) (c)

Fig. 3. The cartpole environment.(a) The original cartpole environment without safety concern. (b) The cart is at -0.3 and its pole angle is -0.3. (c) The cart is at 0.3 and its pole angle is 0.3

the pendulum on a cart from falling over as long as possible. Starting from a position at or close to the original point, in each time step, the agent have 2 actions to choose from to push the cart to the left or right in order to keep the balance of the pendulum. The position, velocity and angle of the cart and the pole are continuous and observable in each time step. The dynamics of the system are unknown. The agent fails as long as the pole angle is more than 20.9 or the cart's horizontal position is more than 2.4. The maximum step length is $t = 200$. Now it's considered 'unsafe' when the cart's position is in range [-2.4, -0.3] while pole's angle is in (-∞, 0.3], or the cart's position is in range [0.3, 2.4] while pole's angle is in [0.3, ∞). An MDP is abstracted from observed data after the expert explores the environment. The states mapped from the unsafe conditions are labelled as 'unsafe'. The feature vector in each state contains 4 feature functions which are sigmoid functions of the abstract values of the position, angle and velocities of the cart and pole, as well as other 22 feature functions which are all radial basis functions which depend on the squared Euclidean distances between current state s and other 20 states which are uniformly distributed in the state space. The expert features provided to the agent are analytically generated by a policy. The PCTL safety specification formula is still in the form of (**??**). In Table. **??**, given different safety constraints p^*, the learnt policies are compared in terms of the model checking results on the PCTL property, average steps the agent can hold in 5000 rounds, and average rates that the agent violates the safety constraints in 5000 rounds.

Safety constraint 1.0 in the first row is equivalent to no safety constraint. The policy tested in this row is learnt via Apprenticeship Learning. The probability of being in unsafe condition is close with the PCTL property of the policy in the MDP. We add a safety constraint and ramp down the safety specification. When the safety constraint ranges from 0.44 to 0.15, there is no obvious degradation in performance. The safety constraint can even refine the policy search in that keeping the cart position and pole angle in safe condition can help maintain the stability of whole system. As a result, some learnt policies can have even higher

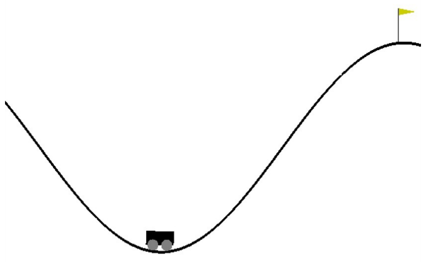
Table 1. In cart pole environment, given the same μ_E from successful and safe expert demonstrations along with different safety specifications, Safety-Aware Apprenticeship Learning algorithm can output different policies with varied bias on safety and performance. But all learnt policies are safe with respect to their safety specifications.

Safety Constraints(p^*)	Model Checking Result	Average Steps	Unsafe Rate
1.0	0.441388844015	159.145	0.44
0.44	0.0876107516626	148.695	0.0
0.4	0.0876107516626	149.541	0.0
0.35	0.0876107516626	145.281	0.00
0.3	0.0900973243006	146.954	0.0
0.25	0.0547655091631	151.852	0.0
0.2	0.110688074988	166.459	0.008
0.15	0.0703377027096	145.399,	0.0
0.1	0.0982627483725	135.493	0.0
0.05	4.57289219928e-7	24.486	0.00
0.01	0.00473854543445	31.734	0.0

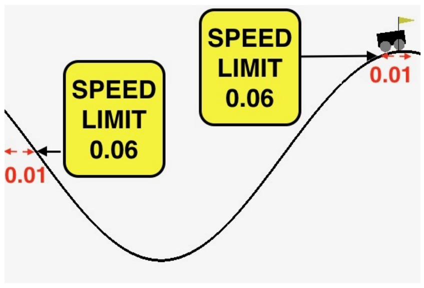
performance than the original policy learnt via Apprenticeship Learning. But when the constraint hits 0.05, the performance is extremely low. In this case, it's possible that the agent becomes so conservative that it would rather let the pendulum fall than risk pushing the cart into unsafe conditions.

1.3 Mountain Car Environment

In our third experiment, we implemented our algorithm on another dynamics system, the mountain car task. In this environment, a car starts from the bottom area of the valley between two mountains and tries to reach the mountaintop on the right by taking as less time steps as possible. In each time step the car shall perform one of the three actions, going left, turning off the engine, and going right. The agent fails when the step length reaches the maximum $t = 200$. The velocity and position of the car are continuous and observable while the dynamics are unknown. The car cannot get enough momentum to reach the top by just accelerating from start. It has to move back and forth in the valley to gain enough speed. There is an inelastic wall at the left end on the left mountain. When car reaches the end, the speed goes to 0 immediately. But now for the sake of safety, additional rule is added that the car shouldn't pass some speed limit when it's close to the left mountaintop or right mountaintop in case of driving off a cliff in realistic world. By sampling from observations, an MDP is set up, in which each state abstractly indicates car's position and velocity. The feature vector for each state contains 2 feature functions which are sigmoid functions of the abstract values of the position and velocity of the car, as well as other 20 feature functions which are all radial basis functions which respectively depend on the squared Euclidean distances between s and other 20 states which



(a)



(b)

Fig. 4. The mountain car environment. (a) The original mountaincar environment without any safety concern. (b) The mountaincar environment with traffic rules. When the distance from the car to the left edge or the right edge is shorter than 0.01, the speed of the car should be lower than 0.06.

are uniformly distributed in the state space. Certain states indicating speeding are labelled as 'unsafe'. This time the expert feature is generated by trajectories that reach the mountaintop without any violation on the speed rule. The PCTL safety specification formula is still in the form of (**??**). In Table. 2, given safety specifications with different constraints, the learnt policies are compared in terms of the model checking results on the PCTL property, average steps the agent takes to finish the goal in 5000 rounds, average rates that the agent violates the safety constraints in 5000 rounds. The average steps that exert takes during demonstration is 162.

Table 2. In mountain car environment, given the same μ_E from successful and safe expert demonstrations along with different safety specifications, Safety-Aware Apprenticeship Learning algorithm can output different policies with varied bias on safety and performance. But all learnt policies are safe with respect to their safety specifications.

Safety Constraints(p^*)	Model Checking Result	Average steps	Unsafe Rate
1.0	0.104	177.1266	0.227
0.08	0.016566	174.054	0.0
0.07	1.79e-5	165.626	0.0
0.06	1.41e-5	164.703	0.0
0.05	1.79e-5	165.336	0.0
0.04	1.0001e-5	157.43	0.0
0.01	1.0001e-5	157.858	0.0
1.0e-5	9.379e-7	fail	0.0

As can be seen from the table, still the data in first row is from policy learnt without safety specification. Not only is its probability of speeding the highest, but also its performance the worst. Compared with other policies in the table, this policy makes the car tend to speed up towards the left mountaintop in order to maximize its potential energy, which is actually a waste of steps. Thus when safety constraint is 0.08, the agent learns to slow down earlier on the left hill not only to keep safe but also save time. But if the safety constraint keeps on getting stricter, the agent has to sacrifice its performance in exchange for safety. When the safety constraint reaches 1.0e-5, the agent is no longer able to reach the mountaintop within 200 steps. In this case, the agent may be too conservative to even accelerate.