1 Safety-aware apprenticeship learning algorithm

In this section, we propose a safety-aware apprenticeship learning for the realization of the counterexample-guided inductive learning framework. Combining policy verification, counterexample generation and learning algorithm, this algorithm searches for policy that satisfies the specification iteration by iteration and keeps gathering candidate policies. Given a specification Φ, this algorithm guarantees that, if a example policy that satisfies Φ can be provided initially, the finally output policy is safe, and its margin to the expert feature μ_E is no larger than the initially provided policy.

Firstly, for the multi-objective optimization problem (**??**), we use a weighted sum approach to find a weight vector ω that fulfill both the objectives of (**??**) and (**??**), which is to keep safe by separating candidate policies from counterexamples. Assuming that $\Pi = \{\mu^1, \mu^2, \mu^3, ...\}$ is a set of expected features of policies no matter its satisfaction of the specification, $\Pi_S = \{\mu_S^1, \mu_S^2, \mu_S^3, ...\}$ is a set of expected features of policies that all satisfy the Φ, $CEX = \{\mu_{CEX}^1, \mu_{CEX}^2, \mu_{CEX}^3, ...\}$ is a set of counterexample expected features. We turn (**??**) into a weighted sum optimization function (1).

$$\max_{\omega} \quad \min_{\mu \in \Pi, \mu_S \in \Pi_S, \mu_{CEX} \in CEX} \omega^T(k(\mu_E - \mu) + (1-k)(\mu_S - \mu_{CEX})) \quad (1)$$

$$s.t. ||\omega||_2 \leq 1 \quad (2)$$

$$k \in \quad [0.0, 1.0] \quad (3)$$

$$\forall \mu' \in \Pi, \omega^T(\mu_E - \mu') \leq \omega^T(\mu_E - \mu) \quad (4)$$

$$\forall \mu'_S \in \Pi_S, \mu'_{CEX} \in CEX, \omega^T(\mu'_S - \mu'_{CEX}) \leq \omega^T(\mu_S - \mu_{CEX}) \quad (5)$$

This optimization problem contains two components respectively induced from (**??**) and (**??**). As both (**??**) and (**??**) are convex optimization problems, (1) is also a convex optimization problem. When k is close to 1.0, solving (1) has the same effect of solving (**??**) and when k is close to 1.0, it has the same effect of solving (**??**). Π is used in the first component as the counterexample of getting high performance since that the true objective of (**??**) is to find a high performance policy by regarding all the found ones as having low performance. Π_S is used in the second component rather than Π so that only the safe property of Π_S is maximized rather than the unknown property of Π. Also, considering the independence of (**??**) and (**??**), additional constraints (4)(5) are added in order to ensure the dominance of the optimal solution at least with respect to μ, μ_S, μ_{CEX}. The ω in the optimal solution shall be used to generate a policy π that is optimal to the reward function determined by ω. This can be done with algorithm such as policy iteration. Then we use model checking tool, such as PRISM, to check if π satisfies Φ. If it satisfies, then it will be added to Π_S. Oracle then checks the expected features of π with $||\mu_E - \mu_\pi||_2$ to see if its margin is the minimal one so far. If true, oracle shall deem π the best candidate policy found and set $\pi^* = \pi$. If π doesn't satisfy Φ, oracle use model checking tool, such as COMICS, to generate a minimal counterexample μ_{CEX_π} of π and add to CEX. Then the margin of π will be compared against π^*. Only if π has a

larger margin than $\pi*$, $||\mu_E - \mu_{\pi^*}||_2 \geq ||\mu_E - \mu_\pi||_2$, shall π be added to Π. This constraint is added so that the search space for high performance safe π^* is not narrowed down by a high performance but unsafe policy. Due to the existence of safe policy set Π_S and μ_S in (1), an initial safe policy is required so that $\Pi_S \neq \emptyset$. But as there is no performance requirement on it, (**??**) can be used to generate a safe policy as shown in Algorithm 1.

Algorithm 1 Initial Safe Policy Generation

SafePolicyGen() ▷ This is only one proposal of generating safe policy
    $t = 0$
    while True:
        $\omega, \mu_{CEX}^{(j)} = argmax_{\omega:||\omega||_2 \leq 1} argmin_{\mu_{CEX}^{(j)} \in CEX}(-w^T \mu_{CEX}^{(j)})$
        $t' = -w^T \mu_{CEX}^{(j)}$
        $\pi \leftarrow$ Optimal policy of ω
        $\mu \leftarrow$ Expected feature of π
        Model check if π satisfies the safety specification
            If False
                If $|t' - t| \leq \epsilon$ ▷ Difference is too small, end loop
                    return Null
                $t = t'$
                $\mu_{CEX} \leftarrow$ expected feature of counterexample of π
                add μ_{CEX} to the counterexample set CEX
            If True
                return π

Algorithm 2 is the complete safety-aware apprenticeship learning algorithm with Algorithm 1 integrated. This algorithm learns and gathers policies in iterations. In each iteration, weight k is updated and adapted to the newly learnt policy. The notion of the algorithm is to try to learn safe policy while making k as close to 1 as possible. The strategy is that once a safe policy is learnt, the multi-objective optimization problem tends to improve performance by adding weight on the (**??**) component in (1), otherwise it tends to keep safe by adding weight on the (**??**) component. We introduce two parameters inf, sup indicating the upper and lower bound of k so that $k \in [inf, sup]$. $inf = 1.0$ is constant while sup is initially 0.0 but can be updated. Assuming that the first safe policy has been found and in iteration i a policy $\pi^{(i)}$ is found. If $\pi^{(i)}$ is verified by the model checker to be satisfying specification Φ, then sup is updated to be equal to current k and k itself is updated to be equal to inf. If $\pi^{(i)}$ doesn't satisfy Φ, k is reduced to $\alpha inf + (1 - \alpha)k$ where $\alpha \in (0, 1)$ is a step length parameter. Obviously the range of k, which is $[inf, sup]$, is narrowed down through iterations. When $|k - inf|$ is smaller than some error bound ϵ and $inf \neq sup$, the algorithm converges. When if $inf = sup = k = 1.0$, it's equivalent to apprenticeship learning and the algorithm stops once the learnt policy doesn't satisfy Φ.it's equivalent to apprenticeship learning and the algorithm stops once the

Algorithm 2 Safety-Aware Apprenticeship Learning Algorithm

Given:
  $\mu_E \leftarrow$ expert feature count
  $\Phi \leftarrow$ the safety specification
  $\epsilon \leftarrow$ error bound
  $\alpha \in (0,1) \leftarrow steplength$

Initialize:
  $\pi^{(0)} \leftarrow$ Optimal policy learnt from μ_E via original AL
  $\mu^{(0)} \approx \mu_E \leftarrow$ Expected feature of $\pi^{(0)}$
  Model check if $\pi^{(0)}$ satisfies Φ
    If True, then $\pi^* = \pi^{(0)}$, $\mu^* = \mu^{(0)}$, return π^*
      ▷ The current policy already satisfies Φ
    If False, $\mu_{CEX}^{(0)} \leftarrow$ Generate counterexample expected features from $\pi^{(0)}$
      add $\mu_{CEX}^{(0)}$ to CEX
      $\pi^* =$ **SafePolicyGen()**
      If π^*, μ^* are Null, then return π^*
        ▷ As $\pi^* = Null$, failed to find a safe policy
      $\mu^* \leftarrow$ Expected feature of π^*
      $inf = 0, sup = 1, k = \alpha(inf + sup)$
      Go to Iteration

Iteration:
  In iteration $i \geq 1$
  Solve $\omega^{(i)} = \underset{\omega}{argmax} \underset{\mu \in \Pi, \mu_S \in \Pi_S, \mu_{CEX} \in CEX}{min} \omega^T(k(\mu_E - \mu) + (1-k)(\mu_S - \mu_{CEX}))$ in (1) with constraints (2)(3)(4)(5).
  $\pi^{(i)} \leftarrow$ Optimal policy of $\omega^{(i)}$
  Model check if $\pi^{(i)}$ satisfies Φ
    If True, $\mu^{(i)} \leftarrow$ Expected feature of $\pi^{(i)}$
      Add $\mu^{(i)}$ to Π, Π_S
      if $||\mu_E - \mu^{(i)}||_2 < ||\mu_E - \mu^*||_2$, then $\mu^* = \mu^{(i)}, \pi^* = \pi^{(i)}$
      inf = k
      k = sup
    If False, $\mu_{CEX}^{(i)} \leftarrow$ Generate counterexample expected features from $\pi^{(i)}$
      If $|k - inf| \leq \epsilon$, then return π^* ▷ Converge, end iteration
      add $\mu_{CEX}^{(i)}$ to CEX
      If $||\mu_E - \mu^{(i)}||_2 \geq ||\mu_E - \mu^*||_2$, then add $\mu^{(i)}$ to Π
      k = $\alpha inf + (1-\alpha)k$
  Go to next iteration

learnt policy doesn't satisfy Φ.it's equivalent to apprenticeship learning and the algorithm stops once the learnt policy doesn't satisfy Φ. During iteration, the best learnt policy is recorded and updated in real time so that the final output policy has the smallest margin to μ_E among all learnt policy.

This algorithm is guaranteed to converge. When $inf = sup = 1.0$, the algorithm is reduced to an apprenticeship learning iteration. So we only consider the convergence when $|k - inf| \leq \epsilon$. After inf is assigned inf_t, k is updated by $k_t = \alpha inf + (1-\alpha)k_{t+1}$ in every iteration until either convergence due to $|k^* - inf_t| \leq \epsilon$ or inf_t is updated to $inf_{t+1} > inf_t$. In the former case, the update of k satisfies:

$$\frac{|k_{t+1} - k^*|}{|k_t - k^*|} \tag{6}$$

$$= \frac{\alpha inf + (1-\alpha)k_t - k^*}{k_t - k^*} \tag{7}$$

$$< \frac{\alpha(k_t - \epsilon) + (1-\alpha)k_t - k^*}{k_t - k^*} \tag{8}$$

$$< 1 - \alpha\epsilon \tag{9}$$

Actually for any ϵ, it takes less than $\log_{1-\alpha} \frac{k-inf}{\epsilon}$ iterations before the algorithm converges or find a safe policy. In the latter case where a safe policy is found and inf_t is updated to $inf_{t+1} > inf_t + \epsilon$, take each such update as a time node and assume that finally inf converges to inf^* then obviously:

$$\frac{|inf_{t+1} - inf^*|}{|inf_t - inf^*|} = \frac{inf^* - inf_{t+1}}{inf^* - inf_t} \tag{10}$$

$$< \frac{inf^* - inf_t - \epsilon}{inf^* - inf_t} \tag{11}$$

$$< 1 - \epsilon \tag{12}$$

Therefore, it takes at most $\frac{1}{\epsilon}$ time nodes for inf to be updated from 0.0 to 1.0. And between each two consecutive time nodes, it takes at most $\log_{1-\alpha} \frac{k-inf}{\epsilon}$ iterations to converge or to cross the time nodes. After $inf = 1.0$, the algorithm becomes apprenticeship learning. The convergence of apprenticeship learning has been proved in [**?**].