Co-training for Policy Learning

Jialin Song[†] **Ravi Lanka**[‡] **Yisong Yue**[†] **Masahiro Ono**[‡]
[†] California Institute of Technology
[‡]Jet Propulsion Laboratory, California Institute of Technology

Abstract

We study the problem of learning sequential decision-making policies in settings with multiple state-action representations. Such settings naturally arise in many domains, such as planning (e.g., multiple integer programming formulations) and various combinatorial optimization problems (e.g., those with both integer programming and graph-based formulations). Inspired by the classical co-training framework for classification, we study the problem of co-training for policy learning. We present sufficient conditions under which learning from two views can improve upon learning from a single view alone. Motivated by these theoretical insights, we present a meta-algorithm for co-training for sequential decision making. Our framework is compatible with both reinforcement learning and imitation learning. We validate the effectiveness of our approach across a wide range of tasks, including discrete/continuous control and combinatorial optimization.

1 INTRODUCTION

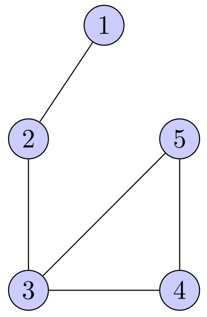
Conventional wisdom in problem-solving suggests that there is more than one way to look at a problem. For sequential decision making problems, such as those in reinforcement learning and imitation learning, one can often utilize multiple different state-action representations to characterize the same problem. A canonical application example is learning solvers for hard optimization problems such as combinatorial optimization [23, 41, 15, 56, 4]. It is well-known in the operations research community that many combinatorial optimization problems have multiple formulations. For example, the maximum cut problem admits a quadratic integer program as well as a linear integer program formulation [7, 18]. Another example is the traveling salesman problem, which admits multiple integer programming formulations [47, 45]. One can also formulate many problems using a graph-based representation (see Figure 1). Beyond learning combinatorial optimization solvers, other examples with multiple state-action representations include robotic applications with multiple sensing modalities such as third-person view demonstrations [57] and multilingual machine translation [27].

In the context of policy learning, one natural question is how different state-action formulations impact learning and, more importantly, how learning can make use of multiple formulations. This is related to the co-training problem [6], where different feature representations of the same problem enable more effective learning than using only a single representation [64, 33]. While co-training has received much attention in classification tasks, little effort has been made to apply it to sequential decision making problems. One issue that arises in the sequential case is that some settings have completely separate state-action representations while others can share the action space.

In this paper, we propose CoPiEr (co-training for policy learning), a meta-framework for policy co-training that can incorporate both reinforcement learning and imitation learning as subroutines. Our approach is based on a novel theoretical result that integrates and extends results from PAC analysis for co-training [16] and general policy learning with demonstrations [29]. To the best of our knowledge, we are the first to formally extend the co-training framework to policy learning.

Our contributions can be summarized as:

- We present a formal theoretical framework for policy co-training. Our results include: 1) a general theoretical characterization of policy improvement, and 2) a specialized analysis in the shared-action setting to explicitly quantify the performance gap



$$\max \; -\sum_{i=1}^{5} x_i,$$
subject to:
$$x_1 + x_2 \geq 1,$$
$$x_2 + x_3 \geq 1,$$
$$x_3 + x_4 \geq 1,$$
$$x_3 + x_5 \geq 1,$$
$$x_4 + x_5 \geq 1,$$
$$x_i \in \{0, 1\}, \forall i \in \{1, \cdots, 5\}$$

Figure 1: Two ways to encode minimum vertex cover (MVC) problems. Left: policies learn to operate directly on the graph view to find the minimal cover set [30]. Right: we express MVC as an integer linear program, then polices learn to traverse the resulting combinatorial search space, i.e., learn to branch-and-bound [23, 56].

(i.e., regret) versus the optimal policy. These theoretical characterizations shed light on rigorous algorithm design for policy learning that can appropriately exploit multiple state-action representations.

- We present CoPiEr (co-training for policy learning), a meta-framework for policy co-training. We specialize CoPiEr in two ways: 1) a general mechanism for policies operating on different representations to provide demonstrations to each other, and 2) a more granular approach to sharing demonstrations in the shared-action setting.

- We empirically evaluate on problems in combinatorial optimization and discrete/continuous control. We validate our theoretical characterizations to identify when co-training can improve on single-view policy learning. We further showcase the practicality of our approach for the combinatorial optimization setting, by demonstrating superior performance compared to a wide range of strong learning-based benchmarks as well as commercial solvers such as Gurobi.

2 RELATED WORK

Co-training Our work is inspired by the classical co-training framework for classification [6], which utilizes two different feature representations, or views, to effectively use unlabeled data to improve the classification accuracy. Subsequent extensions of co-training includes co-EM [44] and co-regularization [55]. Co-training has been widely used in natural language processing [64, 31], clustering [33, 40], domain adaptation [11] and game playing [34]. For policy learning, some related ideas have been explored where multiple estimators of the value or critic function are trained together [67, 63].

In addition to the empirical successes, several previous works also establish theoretical properties of co-training [6, 3, 16, 65]. Common assumptions in these analyses include: 1) each view is sufficient for learning a good classifier on its own, and 2) conditional independence of the features given the labels. Recently, there are works considering weakened assumptions, such as allowing for weak dependencies between the two views [5], or relaxing the sufficiency condition [66].

Policy Learning for Sequential Decision Making Sequential decision making pertains to tasks where the policy performs a series of actions in a stateful environment. A popular framework to characterize the interaction between the agent and the environment is a Markov Decision Process (MDP). There are two main approaches for policy learning in MDPs: reinforcement learning and imitation learning. For both reinforcement learning and imitation learning, we show that co-training on two views can provide improved exploration in the former and surrogate demonstrations in the latter, in both cases leading to superior performance.

Reinforcement learning uses the observed environmental rewards to perform policy optimization. Recent works include Q-Learning approaches such as deep Q-networks [42], as well as policy gradient approaches such as DDPG [38], TRPO [52] and PPO [53]. Despite its successful applications to a wide variety of tasks including playing games [42, 54], robotics [37, 32] and combinatorial optimization [15, 41], high sample complexity and unstable learning pose significant challenges in practice [24], often causing learning to be unreliable.

Imitation learning uses demonstrations (from an expert) as the primary learning signal. One popular class of algorithms is reduction-based [17, 49, 51, 50, 10], which generates cost-sensitive supervised examples from demonstrations. Other approaches include estimating the expert's cost-to go [58], inverse reinforcement learning [1, 26, 68], and behavioral cloning [61]. One major limitation of imitation learning is the reliance on demonstrations. One solution is to combine imitation and reinforcement learning [36, 29, 12, 43] to learn from fewer or coarser demonstrations.

3 BACKGROUND & PRELIMINARIES

Markov Decision Process with Two State Representations. A Markov decision process (MDP) is defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma, \mathcal{S}_T)$. Let $\mathcal{S}$ denote the state space,

$\mathcal{A}$ the action space, $\mathcal{P}(s'|s,a)$ the (probabilistic) state dynamics, $r(s,a)$ the reward function, γ the discount factor and (optinal) $\mathcal{S}_T$ a set of terminal states where the decision process ends. We consider both stochastic and deterministic MDPs. An MDP with two views can be written as $\mathcal{M}^A = (\mathcal{S}^A, \mathcal{A}^A, \mathcal{P}^A, r^A, \gamma^A, \mathcal{S}_T^A)$ and $\mathcal{M}^B = (\mathcal{S}^B, \mathcal{A}^B, \mathcal{P}^B, r^B, \gamma^B, \mathcal{S}_T^B)$. To connect the two views, we make the following assumption about the ability to translate trajectories between the two views.

Assumption 1. *For a (complete) trajectory in $\mathcal{M}^A$, $\tau^A = (s_0^A, a_0^A, s_1^A, a_1^A, \cdots, s_n^A)$, there is a function $f_{A\to B}$ that maps τ^A to its corresponding (complete) trajectory τ^B in the other view $\mathcal{M}^B$: $f_{A\to B}(\tau^A) = \tau^B = (s_0^B, a_0^B, s_1^B, a_1^B, \cdots, s_m^B)$. The rewards for τ^A and τ^B are the same under their respective reward functions, i.e., $\sum_{i=0}^{n-1} r^A(s_i^A, a_i^A) = \sum_{j=0}^{m-1} r^B(s_j^B, a_j^B)$. Similarly, there is a function $f_{B\to A}$ that maps trajectories in $\mathcal{M}^B$ to $\mathcal{M}^A$ which preserves the total rewards.*

Note that in Assumption 1, the length of τ^A and τ^B can be different because of different state and action spaces.

Combinatorial Optimization Example. Minimum vertex cover (MVC) is a combinatorial optimization problem defined over a graph $G = (V, E)$. A cover set is a subset $U \subset V$ such that every edge $e \in E$ is incident to at least one $v \in U$. The objective is to find a U with the minimal cardinality. For the graph in Figure 1, a minimal cover set is $\{2, 3, 4\}$.

There are two natural ways to represent an MVC problem as an MDP. The first is graph-based [15] with the action space as V and the state space as sequences of vertices in V representing partial solutions. The deterministic transition function is the obvious choice of adding a vertex to the current partial solution. The rewards are -1 for each selected vertex. A terminal state is reached if the selected vertices form a cover.

The second way is to formulate an integer linear program (ILP) that encodes an MVC problem:

$$\begin{aligned}
&\max \; -\sum\nolimits_{v\in V} x_v, \\
&\text{subject to :} \\
&x_u + x_v \geq 1, \forall e = (u,v) \in E, \\
&x_v \in \{0,1\}, \forall v \in V.
\end{aligned}$$

We can then use branch-and-bound [35] to solve this ILP, which represents the optimization problem as a search tree, and explores different areas of a search tree through a sequence of branching operations. The MDP states then represent current search tree, and the actions correspond to which node to explore next. The deterministic transition function is the obvious choice of adding a new node into the search tree. The reward is 0 if an action does not lead to a feasible solution and is the objective value of the feasible solution minus the best incumbent objective if an action leads to a node with a better feasible solution. A terminal state is a search tree which contains an optimal solution or reaches a limit on the number of nodes to explore.

The relationship between solutions in the two formulations are clear. For a graph $G = (V, E)$, a feasible solution to the ILP corresponds to a vertex cover by selecting all the vertices $v \in V$ with $x_v = 1$ in the solution. This correspondence ensures the existence of mappings between two representations that satisfy Assumption 1.

Note that, despite the deterministic dynamics, solving MVC and other combinatorial optimization problems can be extremely challenging due to the very large state space. Indeed, policy learning for combinatorial optimization is a topic of active research [30, 23, 56, 41, 4].

Policy Learning. We consider policy learning over a distribution of MDPs. For instance, there can be a distribution of MVC problems. Formally, we have a distribution $\mathcal{D}$ of MDPs that we can sample from (i.e., $\mathcal{M} \sim \mathcal{D}$). For a policy π, we define the following terms:

$$\begin{aligned}
\eta(\pi, \mathcal{M}) &= \mathbb{E}_{\tau\sim\pi}\Big[\sum\nolimits_{i=0}^{n-1} \gamma^i r(s_i, a_i)\Big], \\
J(\pi) &= \mathbb{E}_{\mathcal{M}\sim\mathcal{D}}[\eta(\pi, \mathcal{M})], \\
Q_\pi(s,a) &= \mathbb{E}_{\tau\sim\pi}\Big[\sum\nolimits_{i=0}^{n-1} \gamma^i r(s_i, a_i) | s_0 = s, a_0 = a\Big], \\
V_\pi(s) &= \mathbb{E}_{\tau\sim\pi}\Big[\sum\nolimits_{i=0}^{n-1} \gamma^i r(s_i, a_i) | s_0 = s\Big], \\
A_\pi(s,a) &= Q_\pi(s,a) - V_\pi(s),
\end{aligned}$$

with η being the expected cumulative reward of an individual MDP $\mathcal{M}$, J the overall objective, Q the Q function, V the value function and A the advantage function. The performance of two policies can be related via the advantage function [52, 28]: $\eta(\pi', \mathcal{M}) = \eta(\pi, \mathcal{M}) + \mathbb{E}_{\tau\sim\pi'}[\sum_{i=0}^{n-1} \gamma^i A_\pi(s_i, a_i)]$. Based on Theorem 1 below, we can rewrite the final term with the occupancy measure, $\rho_\pi(s,a) = \mathbb{P}(\pi(s) = a) \sum_{i=0}^{\infty} \gamma^i \mathbb{P}(s_i = s|\pi)$.

Theorem 1. *(Theorem 2 of [60]). For any policy π, it is the only policy that has its corresponding occupancy measure ρ_π, i.e., there is a one-to-one mapping between policies and occupancy measures. Specifically, $\mathbb{P}(\pi(s) = a) = \frac{\rho_\pi(s,a)}{\sum_{a'} \rho_\pi(s,a')}$.*

With slight notation abuse, define $\rho_\pi(s) = \sum_{i=0}^{\infty} \gamma^i \mathbb{P}(s_i = s|\pi)$ to be the state visitation distribution. In policy iteration, we aim to maximize:

$$
\begin{aligned}
&\mathbb{E}_{\tau\sim\pi'}[\sum_{i=0}^{n-1}\gamma^i A_\pi(s_i,a_i)],\\
&=\sum_{i=0}^{n-1}\mathbb{E}_{s_i\sim\rho_{\pi'}(s)}[\mathbb{E}_{a_i\sim\pi'(s_i)}[\gamma^i A_\pi(s_i,a_i)]],\\
&\approx\sum_{i=0}^{n-1}\mathbb{E}_{s_i\sim\rho_{\pi}(s)}[\mathbb{E}_{a_i\sim\pi'(s_i)}[\gamma^i A_\pi(s_i,a_i)]].
\end{aligned}
$$

This is done instead of taking an expectation over $\rho_{\pi'}(s)$ which has a complicated dependency on a yet unknown policy π'. Policy gradient methods tend to use the approximation by using ρ_π which depends on the current policy. We define the approximate objective as:

$$
\begin{aligned}
&\eta_\pi(\pi',\mathcal{M})\\
&=\eta(\pi,\mathcal{M})+\sum_{i=0}^{n-1}\mathbb{E}_{s_i\sim\rho_{\pi}(s)}[\mathbb{E}_{a_i\sim\pi'(s_i)}[\gamma^i A_\pi(s_i,a_i)]],
\end{aligned}
$$

and its associated expectation over $\mathcal{D}$ as $J_\pi(\pi')=\mathbb{E}_{\mathcal{M}\sim\mathcal{D}}[\eta_\pi(\pi',\mathcal{M})]$.

4 A THEORY OF POLICY CO-TRAINING

In this section, we provide two theoretical characterizations of policy co-training. These characterizations highlight a trade-off in sharing information between different views, and motivates the design of our CoPiEr algorithm presented in Section 5.

We restrict our analysis to infinite horizon MDPs, and thus require a strict discount factor $\gamma<1$. We show in our experiments that our CoPiEr algorithm performs well even in finite horizon MDPs with $\gamma=1$. Due to space constraints, we defer all proofs to the appendix.

We present two theoretical analyses with different types of guarantees:

- Section 4.1 quantifies the **policy improvement** in terms of policy advantages and differences, and caters to policy gradient approaches.
- Section 4.2 quantifies the **performance gap** with respect to an optimal policy in terms of policy disagreements, which is a stronger guarantee than policy improvement. This analysis is restricted to the shared action space setting, and caters to learning reduction approaches.

4.1 GENERAL CASE: POLICY IMPROVEMENT WITH DEMONSTRATIONS

For an MDP $\mathcal{M}\sim\mathcal{D}$, consider the rewards of two policies with different views $\eta^A(\pi^A,\mathcal{M}^A)$ and $\eta^B(\pi^B,\mathcal{M}^B)$. If $\eta^A(\pi^A,\mathcal{M}^A)>\eta^B(\pi^B,\mathcal{M}^B)$, π^A performs better than π^B on this instance , and we could use the translated trajectory of π^A as a demonstration for π^B. Even when $J(\pi^A)>J(\pi^B)$, because J is computed in expectation over $\mathcal{D}$, π^B can still outperform π^A on some MDPs. Thus it is possible for the exchange of demonstrations to go in both directions.

Formally, we can partition the distribution $\mathcal{D}$ into two (unnormalized) parts $\mathcal{D}_1$ and $\mathcal{D}_2$ such that the support of $\mathcal{D}$, supp$(\mathcal{D})$ = supp$(\mathcal{D}_1)\cup$ supp$(\mathcal{D}_2)$ and supp$(\mathcal{D}_1)\cap$ supp$(\mathcal{D}_2)=\emptyset$, where for an MDP $\mathcal{M}\in$ supp$(\mathcal{D}_1),\eta(\pi^A,\mathcal{M}^A)\geq\eta(\pi^B,\mathcal{M}^B)$ and for an MDP $\mathcal{M}\in$ supp$(\mathcal{D}_2),\eta(\pi^B,\mathcal{M}^B)>\eta(\pi^A,\mathcal{M}^A)$. By construction, we can quantify the performance gap as:

Definition 1.

$$
\begin{aligned}
\delta_1&=\mathbb{E}_{\mathcal{M}\sim\mathcal{D}_1}[\eta(\pi^A,\mathcal{M}^A)-\eta(\pi^B,\mathcal{M}^B)]\geq 0,\\
\delta_2&=\mathbb{E}_{\mathcal{M}\sim\mathcal{D}_2}[\eta(\pi^B,\mathcal{M}^B)-\eta(\pi^A,\mathcal{M}^A)]>0.
\end{aligned}
$$

We can now state our first result on policy improvement.

Theorem 2. *(Extension of Theorem 1 in [29]) Define:*

$$
\begin{aligned}
\alpha^A_{\mathcal{D}}&=\mathbb{E}_{\mathcal{M}\sim\mathcal{D}}[\max_s D_{KL}(\pi^A(s)\|\pi'^A(s))],\\
\beta^B_{\mathcal{D}_2}&=\mathbb{E}_{\mathcal{M}\sim\mathcal{D}_2}[\max_s D_{JS}(\pi^B(s)\|\pi^A(s))],\\
\alpha^B_{\mathcal{D}}&=\mathbb{E}_{\mathcal{M}\sim\mathcal{D}}[\max_s D_{KL}(\pi^B(s)\|\pi'^B(s))],\\
\beta^A_{\mathcal{D}_1}&=\mathbb{E}_{\mathcal{M}\sim\mathcal{D}_1}[\max_s D_{JS}(\pi^A(s)\|\pi^B(s))],\\
\epsilon^B_{\mathcal{D}_2}&=\max_{\mathcal{M}\in supp(\mathcal{D}_2)}\max_{s,a}|A_{\pi^B}(s,a)|,\\
\epsilon^A_{\mathcal{D}}&=\max_{\mathcal{M}\in supp(\mathcal{D})}\max_{s,a}|A_{\pi^A}(s,a)|,\\
\epsilon^A_{\mathcal{D}_1}&=\max_{\mathcal{M}\in supp(\mathcal{D}_1)}\max_{s,a}|A_{\pi^A}(s,a)|,\\
\epsilon^B_{\mathcal{D}}&=\max_{\mathcal{M}\in supp(\mathcal{D})}\max_{s,a}|A_{\pi^B}(s,a)|.
\end{aligned}
$$

Here D_{KL} & D_{JS} denote the Kullback-Leibler and Jensen-Shannon divergence respectively. Then we have:

$$
\begin{aligned}
J(\pi'^A)&\geq J_{\pi^A}(\pi'^A)-\frac{2\gamma^A(4\beta^B_{\mathcal{D}_2}\epsilon^B_{\mathcal{D}_2}+\alpha^A_{\mathcal{D}}\epsilon^A_{\mathcal{D}})}{(1-\gamma^A)^2}+\delta_2,\\
J(\pi'^B)&\geq J_{\pi^B}(\pi'^B)-\frac{2\gamma^B(4\beta^A_{\mathcal{D}_1}\epsilon^A_{\mathcal{D}_1}+\alpha^B_{\mathcal{D}}\epsilon^B_{\mathcal{D}})}{(1-\gamma^B)^2}+\delta_1.
\end{aligned}
$$

Compared to conventional analyses on policy improvement, the new key terms that determine how much the policy improves are the β's and δ's. The β's, which quantify the maximal divergence between π^A and π^B, hinders improvement, while the δ's contribute positively. If the net contribution is positive, then the policy improvement bound is larger than that of conventional single view policy gradient. This insight motivates co-training algorithms that explicitly aim to minimize the β's.

One technicality is how to compute $D_{JS}(\pi^A(s)\|\pi^B(s))$ given that the state and action spaces for the two representations might be different. Proposition 1 ensures that we can measure the Jensen-Shannon divergence between two policies with different MDP representations.

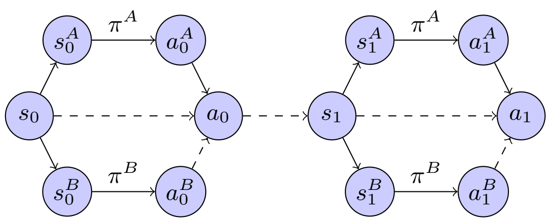

$$\begin{tikzcd} & s_0^A \arrow[r, "\pi^A"] & a_0^A \arrow[dr] & & & s_1^A \arrow[r, "\pi^A"] & a_1^A \arrow[dr] & \\ s_0 \arrow[ur] \arrow[dr] \arrow[rrr, dashed] & & & a_0 \arrow[r, dashed] & s_1 \arrow[ur] \arrow[dr] \arrow[rrr, dashed] & & & a_1 \\ & s_0^B \arrow[r, "\pi^B"] & a_0^B \arrow[ur, dashed] & & & s_1^B \arrow[r, "\pi^B"] & a_1^B \arrow[ur, dashed] & \end{tikzcd}$$

Figure 2: Co-training with shared action space.

Proposition 1. *For representations $\mathcal{M}^A$ and $\mathcal{M}^B$ of an MDP satisfying Assumption 1, the quantities $\max_s D_{JS}(\pi^B(s)\|\pi^A(s))$ and $\max_s D_{JS}(\pi^A(s)\|\pi^B(s))$ are well-defined.*

Minimizing $\beta^B_{\mathcal{D}_2}$ and $\beta^A_{\mathcal{D}_1}$ is not straightforward since the trajectory mappings between the views can be very complicated. We present practical algorithms in Section 5.

4.2 SPECIAL CASE: PERFORMANCE GAP FROM OPTIMAL POLICY IN SHARED ACTION SETTING

We now analyze the special case where the action spaces of the two views are the same, i.e., $\mathcal{A}^A = \mathcal{A}^B$. Figure 2 depicts the learning interaction between π^A and π^B. For each state s, we can directly compare actions chosen by the two policies since the action space is the same. This insight leads to a stronger analysis result where we can bound the gap between a co-trained policy with an optimal policy. The approach we take resembles learning reduction analyses for interactive imitation learning.

For this analysis we focus on discrete action spaces with k actions, deterministic learned policies, and a deterministic optimal policy (which is guaranteed to exist [48]). We reduce policy learning to classification: for a given state s, the task of identifying the optimal action $\pi^*(s)$ is a classification problem. We build upon the PAC generalization bound results in [16] and show that under Assumption 2, optimizing a measure of disagreements between the two policies leads to effective learning of π^*.

Assumption 2. *For a state s, its two representations s^A and s^B are conditionally independent given the optimal action $\pi^*(s)$.*

This assumption is common in analyses of co-training for classification [6, 16]. Although this assumption is typically violated in practice [44], our empirical evaluation still demonstrates strong performance.

Assumption 2 corresponds to a graphical model describing the relationship between optimal actions and the state representations (Figure 3). The intuition is that, when we do not know $a^* = \pi^*(s)$, we should maximize the agreement between $a^A = \pi^A(s^A)$ and $a^B = \pi^B(s^B)$. By the data-processing inequality in information theory [13], we know that $\mathbb{I}(a^A; a^*) \geq \mathbb{I}(a^A; a^B)$. In practice, this means that if a^A and a^B agree a lot, they must reveal substantial information about what a^* is. We formalize this intuition and obtain an upper bound on the classification error rate, which enables quantifying the performance gap. Notice that if we do not have any information from π^*, the best we can hope for is to learn a mapping that matches π^* up to some permutation of the action labels [16]. Thus we assume we have enough state-action pairs from π^* so that we can recover the permutation. In practice this is satisfied as we demonstrate in Section 6.1.

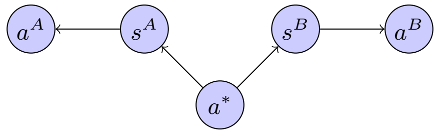

$$\begin{tikzcd} a^A & s^A \arrow[l] & & s^B \arrow[r] & a^B \\ & & a^* \arrow[ul] \arrow[ur] & & \end{tikzcd}$$

Figure 3: Graphical model encodes the conditional independence model.

Formally, we connect the performance gap between a learned policy and an optimal policy with an empirical estimation on the disagreement in action choices among two co-trained policies. Let $\{\tau_i^A\}_{i=1}^m$ be sampled trajectories from π^A and $\{f_{A\to B}(\tau_i^A)\}_{i=1}^m$ be the mapped trajectories in $\mathcal{M}^B$. In $\{f_{A\to B}(\tau_i^A)\}_{i=1}^m$, let $N(a^A = i)$ be the number of times action i is chosen by π^A and $N = \sum_{i=1}^k N(a^A = i)$ be the total number of actions in one trajectory set. Let $N(a^B = i)$ be the number of times action i is chosen by π^B when going through the states in $\{f_{A\to B}(\tau_i^A)\}_{i=1}^m$ and $N(a^A = i, a^B = i)$ record when both actions agree on i.

We also require a measure of model complexity, as is common in PAC style analysis. We use $|\pi|$ to denote the number of bits needed to represent π. We can now state our second main result quantifying the performance gap with respect to an optimal policy:

Theorem 3. *If Assumption 2 holds for $\mathcal{M} \sim \mathcal{D}$ and a deterministic optimal policy π^*. Let π^A and π^B be two deterministic policies for the two representations. Define:*

$$\hat{\mathbb{P}}(a^A = i \mid a^B = i) = \frac{N(a^A = i, a^B = i)}{N(a^B = i)},$$

$$\hat{\mathbb{P}}(a^A \neq i \mid a^B = i) = \frac{N(a^A \neq i, a^B = i)}{N(a^B = i)},$$

$$\epsilon_i(\pi^A, \pi^B, \sigma) = \sqrt{\frac{\ln 2(|\pi^A| + |\pi^B|) + \ln(2k/\sigma)}{2N(a^B = i)}},$$

$$\zeta_i(\pi^A, \pi^B, \sigma) = \hat{\mathbb{P}}(a^A = i \mid a^B = i) - \hat{\mathbb{P}}(a^A \neq i \mid a^B = i) - 2\epsilon_i(\pi^A, \pi^B, \sigma),$$

$$\begin{aligned} b_i(\pi^A, \pi^B, \sigma) &= \frac{1}{\zeta_i(\pi^A, \pi^B, \delta)}(\hat{\mathbb{P}}(a^A \neq i \mid a^B = i) \\ &\quad + \epsilon_i(\pi^A, \pi^B, \sigma)), \\ \ell(s, \pi) &= \mathbf{1}(\pi(s) \neq \pi^*(s)), \\ \epsilon_A &= \mathbb{E}_{s \sim \rho_{\pi^A}}[\ell(s, \pi^A)]. \end{aligned}$$

Then with probability $1-\sigma$:

$$\epsilon_A \leq \max_{j \in \{1,\cdots,k\}} b_j(\pi^A, \pi^B, \sigma),$$
$$\eta(\pi^A, \mathcal{M}^A) \geq \eta(\pi^*, \mathcal{M}) - uT\epsilon_A,$$

where T is the time horizon and u is the largest one-step deviation loss compared with π^*.

To obtain a small performance gap compared to π^*, one must minimize ϵ_A, which measures the disagreement between π^A and π^*. However, we cannot directly estimate this quantity since we only have limited sample trajectories from π^*. Alternatively, we can minimize an upper bound, $\max_{j\in\{1,\cdots,k\}} b_j(\pi^A, \pi^B, \delta)$, which measures the maximum disagreement on actions between π^A and π^B and, importantly, can be estimated via samples. In Section 5.2, we design an algorithm that approximately minimizes this bound. The advantage of two views over a single view enables us to establish an upper bound on ϵ_A, which is otherwise unmeasureable.

5 THE CoPiEr ALGORITHM

We now present practical algorithms motivated by the theoretical insights from Section 4. We start with a meta-algorithm named CoPiEr (Algorithm 1), whose important subroutines are EXCHANGE and UPDATE. We provide two concrete instantiations for the general case and the special case with a shared action space.

Algorithm 1 CoPiEr (Co-training for Policy Learning)

```
1: Input: A distribution $\mathcal{D}$ of MDPs, two policies
   $\pi^A, \pi^B$, mapping functions $f_{A\to B}, f_{B\to A}$
2: repeat
3:     Sample $\mathcal{M} \sim \mathcal{D}$, form $\mathcal{M}^A, \mathcal{M}^B$
4:     Run $\pi^A$ on $\mathcal{M}^A$ to generate trajectories $\{\tau_i^A\}_{i=1}^m$
5:     Run $\pi^B$ on $\mathcal{M}^B$ to generate trajectories $\{\tau_j^B\}_{j=1}^n$
6:     $\{\tau_i'^A\}, \{\tau_j'^B\} \leftarrow$ EXCHANGE$(\{\tau_i^A\}, \{\tau_j^B\})$
7:     $\pi^A \leftarrow$ UPDATE$(\pi^A, \{\tau_i^A\}, \{\tau_j'^A\})$
8:     $\pi^B \leftarrow$ UPDATE$(\pi^B, \{\tau_i^B\}, \{\tau_j'^B\})$
9: until Convergence
```

5.1 GENERAL CASE

Algorithm 2 covers the general case for exchanging trajectories generated by the two policies. First we estimate

Algorithm 2 EXCHANGE: General Case

```
1: Input: Trajectories $\{\tau_i^A\}_{i=1}^m$ and $\{\tau_j^B\}_{j=1}^n$
2: Compute estimate $\hat{\eta}(\pi^A, \mathcal{M}^A) = \frac{1}{m}\sum_{i=1}^m r(\tau_i^A)$
3: Compute estimate $\hat{\eta}(\pi^B, \mathcal{M}^B) = \frac{1}{n}\sum_{j=1}^n r(\tau_j^B)$
4: if $\hat{\eta}(\pi^A, \mathcal{M}^A) > \hat{\eta}(\pi^B, \mathcal{M}^B)$ then
5:     $\{\tau_i^{A\to B}\} \leftarrow \{f_{A\to B}(\tau_i^A)\}_{i=1}^m$
6:     $\{\tau_j^{B\to A}\} \leftarrow \emptyset$
7: else
8:     $\{\tau_i^{A\to B}\} \leftarrow \emptyset$
9:     $\{\tau_j^{B\to A}\} \leftarrow \{f_{B\to A}(\tau_j^B)\}_{j=1}^n$
10: end if
11: return $\{\tau_i^{A\to B}\}, \{\tau_j^{B\to A}\}$
```

Algorithm 3 UPDATE

```
1: Input: Current policy $\pi$, sampled trajectories from
   $\pi$, $\{\tau_i\}_{i=1}^m$ and demonstrations $\{\tau_j'\}_{j=1}^n$
2: Form a loss function $\mathcal{L}(\pi) =$
   $\begin{cases} -\sum_{i=1}^m r(\tau_i) + \lambda C(\pi, \{\tau_j'\}_{j=1}^n), & \text{RL with IL loss} \\ \lambda C(\pi, \{\tau_j'\}_{j=1}^n), & \text{IL loss only} \end{cases}$
3: Update $\pi \leftarrow \pi - \alpha \nabla \mathcal{L}(\pi)$
```

the relative quality of the two policies from sampled trajectories (Lines 2-4 in Algorithm 2). Then we use the trajectories from the better policy as demonstrations for the worse policy on this MDP. This mirrors the theoretical insight presented in Section 4, where based on which sub-distribution an MDP is sampled from, the relative quality of the two policies is different.

For UPDATE, we can form a loss function that is derived from either imitation learning or reinforcement learning. Recall that we aim to optimize the β terms in Theorem 2, however it is infeasible to directly optimize them. So we consider a surrogate loss C (Line 2 of Algorithm 3) that measures the policy difference. In practice, we typically use behavior cloning loss as the surrogate.

5.2 SPECIAL CASE: SHARED ACTION SPACE

For the special case with a shared action space, we can collect more informative feedback beyond the trajectory level. Instead, we collect interactive state-level feedback, as is popular in imitation learning algorithms such as DAgger [51] and related approaches [58, 17, 49, 56, 23]. Specifically, we can use Algorithms 4 & 5 to exchange actions in a state-coupled manner. This process is depicted in Figure 2, where π^A's visited states, s_0^A and s_1^A, are mapped to s_0^B and s_1^B, resulting in receiving π^B's actions, a_0^B and a_1^B, in the exchange.

Unlike the general case where information exchange is asymmetric, as Theorem 3 indicates, we aim to minimize

Algorithm 4 EXCHANGE: Special Case

```
1: Input: Trajectories $\{\tau_i^A\}_{i=1}^m$ and $\{\tau_j^B\}_{j=1}^n$
2: $D^{A\to B}$ = INTERACTIVE($\{f_{B\to A}(\tau_j^B)\}_{j=1}^n, \pi^A$)
3: $D^{B\to A}$ = INTERACTIVE($\{f_{A\to B}(\tau_i^A)\}_{i=1}^m, \pi^B$)
4: return $D^{A\to B}, D^{B\to A}$
```

Algorithm 5 INTERACTIVE

```
1: Input: Trajectories $\{\tau_i\}_{i=1}^m$, query policy $\pi$
2: $D = \emptyset$
3: for $i \leftarrow 1$ to $m$ do
4:     for each state $s \in \tau_i$ do
5:         $D \leftarrow D \cup \{(s, \pi(s))\}$
6:     end for
7: end for
8: return $D$
```

policy disagreement. Both policies are simultaneously optimizing this objective, which requires both directions of information exchange (Lines 2-3 in Algorithm 4). The update step (Algorithm 3) is the same as the general case.

6 EXPERIMENTS

We now present empirical results on both the special and general cases of CoPiEr. We demonstrate the generality of our approach by applying three distinct combinations of policy co-training: reinforcement learning on both views (Section 6.1), reinforcement learning on one view and imitation learning on the other (Section 6.2), and imitation learning on both views (Section 6.3). Furthermore, our experiments on combinatorial optimization (Sections 6.2 & 6.3) demonstrate significant improvements over strong learning-based baselines as well as commercial solvers, and thus showcase the practicality of our approach. More details about the experiment setup can be found in the appendix.

6.1 DISCRETE & CONTINUOUS CONTROL: SPECIAL CASE WITH RL+RL

Setup. We conduct experiments on discrete and continuous control tasks with OpenAI Gym [8] and Mujoco physical engine [62]. We use the garage repository [19] to run reinforcement learning for both views.

Two Views and Features. For each environment, states are represented by feature vectors, typically capturing location, velocity and acceleration. We create two views by removing different subsets of features from the complete feature set. Note that both views have the same underlying action space as the original MDP, so it is the special case covered in Section 5.2. We use interactive feedback for policy optimization.

Policy Class. We use a feed-forward neural network with two hidden layers (64 & 32 units) and tanh activations as the policy class. For discrete actions, $\pi(s)$ outputs a softmax distribution. For continuous actions, $\pi(s)$ outputs a (multivariate) Gaussian. For policy update, we use Policy Gradient [59] with a linear baseline function [21] and define the loss function C in Algorithm 3 to be the KL-divergence between output action distributions.

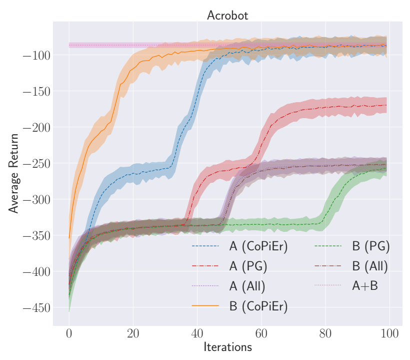
Methods Compared. We compare with single view policy gradient, labelled as "A (PG)" and "B (PG)", and with a policy trained on the union of the two views but test on two views separately, labelled as "A (All)" and "B (All)". We also establish an upper bound on performance by training a model without view splitting ("A+B"). Each method uses the same total number of samples (i.e., CoPiEr uses half per view).

Results. Figure 4 shows the results. CoPiEr is able to converge to better or comparable solutions in almost all cases except for view A in Hopper. The poor performance in Hopper could be due to the disagreement between the two policies not shrinking enough to make Theorem 3 meaningful. As a comparison, at end of the training, the average KL-divergence for the two policies is about 2 for Hopper, compared with 0.23 for Swimmer and 0.008 for Acrobot. One possible cause for such large disagreement is that the two views have significance differences in difficulty for learning, which is the case for Hopper by noticing A (PG) and B (PG) have a difference in returns of about 190.

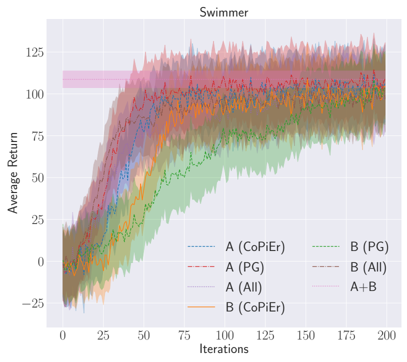
6.2 MINIMUM VERTEX COVER: GENERAL CASE WITH RL+IL

Setup. We now consider the challenging combinatorial optimization problem of minimum vertex cover (MVC). We use 150 randomly generated Erdős-Rényi [20] graph instances for each scale, with scales ranging {100-200, 200-300, 300-400, 400-500} vertices. For training, we use 75 instances, which we partition into 15 labeled and 60 unlabeled instances. We use the best solution found by Gurobi within 1 hour as the expert solution for the labeled set to bootstrap imitation learning. For each scale, we use 30 held-out graph instances for validation, and we report the performance on 45 test graph instances.

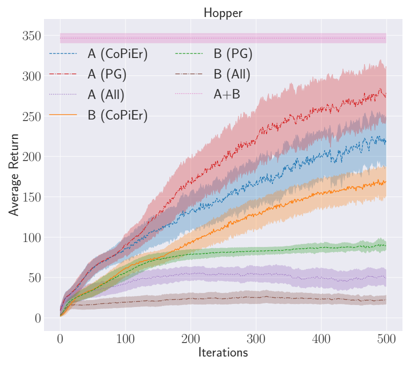
Views and Features. The two views are the graphs themselves and integer linear programs constructed from the graphs. For the graph view, we use DQN-based reinforcement learning [15] to learn a sequential vertex selection policy. We use `structure2vec` [14] to compute graph embeddings to use as state representations. For the ILP, we use imitation learning [23] to learn node



(a) Acrobot Swing-up. A denotes removing the first coordinate in the state vector and B removing the second coordinate.



(b) Swimmer. A denotes removing all even index coordinates in the state vector and B removing all odd index ones.



(c) Hopper. A denotes removing all even index coordinates in the state vector and B removing all odd index ones.

Figure 4: Discrete & continuous control tasks. Experiment results are across 5 random seeded runs. Shaded area indicates ± 1 standard deviation.

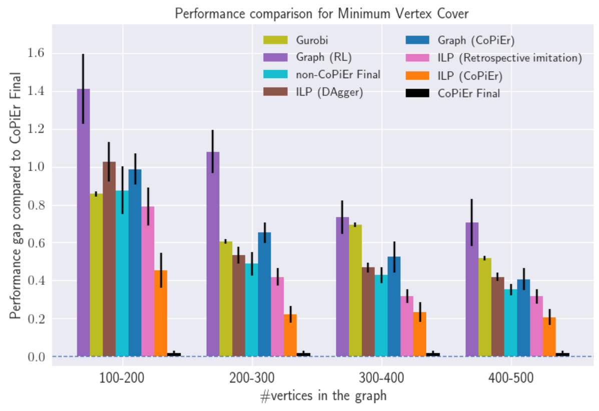


Figure 5: Comparison of CoPiEr with other learning-based baselines and a commercial solver, Gurobi. The y-axis measure relative gaps of various methods compared with CoPiEr Final. CoPiEr Final outperforms all the baselines. Notably, the gaps are significant because getting optimizing over large graphs is very challenging.

selection policy for branch-and-bound search. A node selection policy determines which node to explore next in the current branch-and-bound search tree. We use node-specific features (e.g., LP relaxation lower bound and objective value) and tree-specific features (e.g., integrality gap, and global lower and upper bounds) as our state representations. Vertex selection in graphs and node selection in branch-and-bound are different. So we use the general case algorithm in Section 5.1.

Policy Class. For the graph view, our policy class is similar to [15]. In order to perform end-to-end learning of the parameters with labeled data exchanged between the two views, we use DQN [42] with supervised losses [25] to learn to imitate better demonstrations from the ILP view. For all our experiments, we determined the regularizer for the supervised losses and other parameters through cross-validation on the smallest scale (100-200 vertices). The graph view models are pre-trained with the labeled set using behavior cloning. We use the same number of training iterations for all the methods.

For the ILP view, our policy class consists of a node ranking model that prioritizes which node to visit next. We use RankNet [9] as the ranking model, instantiated using a 2-layer neural network with ReLU as activation functions. We implement our approach for the ILP view within the SCIP [2] integer programming framework.

Methods Compared. At test time, when a new graph is given, we run both policies and return the better solution. We term this practical version "CoPiEr Final" and measure other policies' performance against it. We compare with single view learning baselines. For the graph view, we compare with RL-based policy learning over graphs [15], labelled as "Graph (RL)". And for the ILP view, we compare with imitation learning [23] "ILP (DAgger)", retrospective imitation [56] "ILP (Retrospective Imitation)" and a commercial solver Gurobi [22]. We combine "Graph (RL)" and "ILP (DAgger)" as non-CoPiEr (Final) by returning the better solution of the two. We also show the performance of the two policies in CoPiEr as standalone policies instead of combining them, labelled "Graph (CoPiEr)" and "ILP (CoPiEr)". ILP methods are limited by the same node budget in branch-and-bound trees.

Results. Figure 5 shows the results. We see that CoPiEr Final outperforms all baselines as well as

Gurobi. Interestingly, it also performs much better than either standalone CoPiEr policies, which suggests that Graph (CoPiEr) is better for some instances while ILP (CoPiEr) is better on others. This finding validates combining the two views to maximize the benefits from both. For the exact numbers on the final performance, please refer to Appendix 8.4.

6.3 RISK-AWARE PATH PLANNING: GENERAL CASE WITH IL+IL

Setup. We finally consider a practical application of risk-aware path planning [46]. Given a start point, a goal point, a set of polygonal obstacles, and an upper bound of the probability of failure (risk bound), we must find a path, represented by a sequence of way points, that minimizes cost while limiting the probability of collision to within the risk bound. Details on the data generation can be found in the Appendix 8.3. We report the performance evaluations on 50 test instances.

Views and Features. This problem can be formulated into a mixed integer linear program (MILP) as well as a quadratically constrained quadratic program (QCQP), both of which can be solved using branch-and-bound [35, 39]. For each view, we learn a node selection policy for branch-and-bound via imitation learning. Feature representations are similar to ILP view in MVC experiment (Section 6.2). For the QCQP view, we use the state variables bounds along the trace for each node from the root in the branch and bound tree as an additional feature. Although the search framework is the same, because of the different nature of the optimization problem formulations, the state and action space are incompatible, and so we use the general case of CoPiEr. A pictorial representation of the two views is presented in Appendix 8.2.

Policy Class. The policy class for both MILP and QCQP views is similar to that of ILP view in MVC (Section 6.2), and we learn node ranking models.

Methods Compared. Similar to MVC experiment, we compare other methods with "CoPiEr Final" which returns the better solution of the two. We use single view learning baselines, specifically those based on imitation learning [23], "QCQP (DAgger)" and "MILP(DAgger)", and on retrospective imitation [56], "QCQP (Retrospective Imitation)" and "MILP (Retrospective Imitation)". Two versions of non-CoPiEr Final are presented, based on DAgger and Retrospective Imitation, respectively. Gurobi is also used to solve MILPs but it is not able to solve the QCQPs because they are non-convex.

Results. Figure 6 shows the results. Like in MVC, we again see that CoPiEr Final outperforms baselines as well as Gurobi. We also observe a similar benefit of aggregating both policies. The effectiveness of CoPiEr enables solving much larger problems than considered in previous work [56] (560 vs 1512 binary variables).

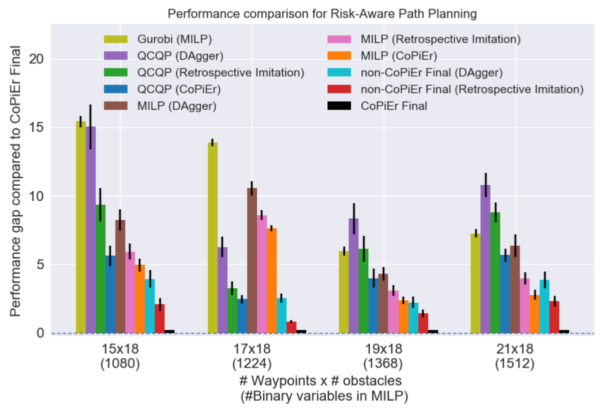


Figure 6: Comparison of CoPiEr with other learning-based baselines and a commercial solver, Gurobi. The y-axis measure relative gaps of various methods compared with CoPiEr Final. CoPiEr Final outperforms all the baselines. Notably, the scale of problems as measured by the number of integer variables far exceed previous state-of-the-art method [56].

7 CONCLUSION & FUTURE WORK

We have presented CoPiEr (Co-training for Policy Learning), a general framework for policy learning for sequential decision making tasks with two representations. Our theoretical analyses and algorithm design cover both the general case as well as a special case with shared action spaces. Our approach is compatible with both reinforcement learning and imitation learning as subroutines. We evaluated on a variety of settings, including control and combinatorial optimization. Our results on showcase the generality of our framework and significant improvements over numerous baselines.

There are many interesting directions for future work. On the theory front, directions include weakening assumptions such as conditional independence, or extending to more than two views. On the application front, algorithms such as CoPiEr can potentially improve performance in a wide range of robotic and other autonomous systems that utilize different sensors and image data.

Acknowledgments. The work was funded in part by NSF awards #1637598 & #1645832, and support from Raytheon and Northrop Grumman. This research was also conducted in part at the Jet Propulsion Lab, California Insitute of Technology under a contract with the National Aeronautics and Space Administration.

References

[1] Pieter Abbeel and Andrew Y Ng. Apprenticeship learning via inverse reinforcement learning. In *International Conference on Machine Learning*, 2004.

[2] Tobias Achterberg. SCIP : solving constraint integer programs. *Mathematical Programming Computation*, 2009.

[3] Maria-Florina Balcan, Avrim Blum, and Ke Yang. Co-training and expansion: Towards bridging theory and practice. In *Neural information processing systems*, 2005.

[4] Mislav Balunovic, Pavol Bielik, and Martin Vechev. Learning to solve smt formulas. In *Neural Information Processing Systems*, 2018.

[5] Avrim Blum and Yishay Mansour. Efficient co-training of linear separators under weak dependence. In *Conference on Learning Theory*, 2017.

[6] Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *Conference on Learning Theory*, 1998.

[7] Endre Boros and Peter L Hammer. The max-cut problem and quadratic 0–1 optimization; polyhedral aspects, relaxations and bounds. *Annals of Operations Research*, 1991.

[8] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv*, 2016.

[9] Chris Burges, Erin Renshaw, and Matt Deeds. Learning to rank using gradient descent. In *International conference on Machine learning*, 1998.

[10] Kai-Wei Chang, Akshay Krishnamurthy, Alekh Agarwal, Hal Daume, and John Langford. Learning to search better than your teacher. In *International Conference on Machine Learning*, 2015.

[11] Minmin Chen, Kilian Q Weinberger, and John Blitzer. Co-training for domain adaptation. In *Neural information processing systems*, 2011.

[12] Ching-An Cheng, Xinyan Yan, Nolan Wagener, and Byron Boots. Fast policy learning through imitation and reinforcement. In *Conference on Uncertainty in Artificial Intelligence*, 2018.

[13] Thomas M Cover and Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[14] Hanjun Dai, Bo Dai, and Le Song. Discriminative Embeddings of Latent Variable Models for Structured Data. In *International Conference on Machine Learning*, pages 1–23, 2016.

[15] Hanjun Dai, Elias B Khalil, Yuyu Zhang, Bistra Dilkina, and Le Song. Learning combinatorial optimization algorithms over graphs. In *Neural Information Processing Systems*, 2017.

[16] Sanjoy Dasgupta, Michael L Littman, and David A McAllester. Pac generalization bounds for co-training. In *Neural information processing systems*, 2002.

[17] Hal Daumé, John Langford, and Daniel Marcu. Search-based structured prediction. *Machine learning*, 2009.

[18] Wenceslas Fernandez de la Vega and Claire Kenyon-Mathieu. Linear programming relaxations of maxcut. In *ACM-SIAM symposium on Discrete algorithms*, 2007.

[19] Yan Duan, Xi Chen, Rein Houthooft, John Schulman, and Pieter Abbeel. Benchmarking deep reinforcement learning for continuous control. In *International Conference on Machine Learning*, 2016.

[20] Paul Erdős and Alfréd Rényi. On the evolution of random graphs. *Publ. Math. Inst. Hung. Acad. Sci*, 1960.

[21] Evan Greensmith, Peter L Bartlett, and Jonathan Baxter. Variance reduction techniques for gradient estimates in reinforcement learning. *Journal of Machine Learning Research*, 2004.

[22] LLC Gurobi Optimization. Gurobi optimizer reference manual, 2018.

[23] He He, Hal Daume III, and Jason M Eisner. Learning to search in branch and bound algorithms. In *Neural information processing systems*, 2014.

[24] Peter Henderson, Riashat Islam, Philip Bachman, Joelle Pineau, Doina Precup, and David Meger. Deep reinforcement learning that matters. In *AAAI Conference on Artificial Intelligence*, 2018.

[25] Todd Hester, Olivier Pietquin, Marc Lanctot, Tom Schaul, Dan Horgan, John Quan, Andrew Sendonaris, Ian Osband, Gabriel Dulac-arnold, John Agapiou, and Joel Z Leibo. Deep Q-Learning from Demonstrations. In *AAAI Conference on Artificial Intelligence*, 2018.

[26] Jonathan Ho and Stefano Ermon. Generative adversarial imitation learning. In *Neural Information Processing Systems*, 2016.

[27] Melvin Johnson, Mike Schuster, Quoc V Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, et al. Googles multilingual neural machine translation system: Enabling zero-shot translation. *Transactions of the Association for Computational Linguistics*, 2017.

[28] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *International Conference on Machine Learning*, 2002.

[29] Bingyi Kang, Zequn Jie, and Jiashi Feng. Policy optimization with demonstrations. In *International Conference on Machine Learning*, 2018.

[30] Elias Boutros Khalil, Pierre Le Bodic, Le Song, George L Nemhauser, and Bistra N Dilkina. Learning to branch in mixed integer programming. In *AAAI Conference on Artificial Intelligence*, 2016.

[31] Svetlana Kiritchenko and Stan Matwin. Email classification with co-training. In *Conference of the Center for Advanced Studies on Collaborative Research*, 2011.

[32] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 2013.

[33] Abhishek Kumar and Hal Daumé. A co-training approach for multi-view spectral clustering. In *International Conference on Machine Learning*, 2011.

[34] Guillaume Lample and Devendra Singh Chaplot. Playing fps games with deep reinforcement learning. In *AAAI Conference on Artificial Intelligence*, 2017.

[35] Ailsa H Land and Alison G Doig. An automatic method for solving discrete programming problems. In *50 Years of Integer Programming 1958-2008*, pages 105–132. 2010.

[36] Hoang Le, Nan Jiang, Alekh Agarwal, Miroslav Dudik, Yisong Yue, and Hal Daumé. Hierarchical imitation and reinforcement learning. In *International Conference on Machine Learning*, 2018.

[37] Sergey Levine, Chelsea Finn, Trevor Darrell, and Pieter Abbeel. End-to-end training of deep visuomotor policies. *The Journal of Machine Learning Research*, 2016.

[38] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations*, 2016.

[39] Jeff Linderoth. A simplicial branch-and-bound algorithm for solving quadratically constrained quadratic programs. *Mathematical programming*, 2005.

[40] Jialu Liu, Chi Wang, Jing Gao, and Jiawei Han. Multi-view clustering via joint nonnegative matrix factorization. In *SIAM International Conference on Data Mining*, 2013.

[41] Azalia Mirhoseini, Hieu Pham, Quoc V Le, Benoit Steiner, Rasmus Larsen, Yuefeng Zhou, Naveen Kumar, Mohammad Norouzi, Samy Bengio, and Jeff Dean. Device placement optimization with reinforcement learning. In *International Conference on Machine Learning*, 2017.

[42] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv*, 2013.

[43] Ashvin Nair, Bob McGrew, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Overcoming exploration in reinforcement learning with demonstrations. In *International Conference on Robotics and Automation*, 2018.

[44] Kamal Nigam and Rayid Ghani. Analyzing the effectiveness and applicability of co-training. In *ACM Conference on Information and knowledge Management*, 2000.

[45] Temel Öncan, İ Kuban Altınel, and Gilbert Laporte. A comparative analysis of several asymmetric traveling salesman problem formulations. *Computers & Operations Research*, 2009.

[46] Masahiro Ono and Brian C Williams. An efficient motion planning algorithm for stochastic dynamic systems with constraints on probability of failure. In *AAAI Conference on Artificial Intelligence*, 2008.

[47] AJ Orman and HP Williams. A survey of different integer programming formulations of the travelling salesman problem. In *Optimisation, econometric and financial analysis*. Springer, 2007.

[48] Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

[49] Stéphane Ross and Drew Bagnell. Efficient reductions for imitation learning. In *International Conference on Artificial Intelligence and Statistics*, 2010.

[50] Stephane Ross and J Andrew Bagnell. Reinforcement and imitation learning via interactive no-regret learning. *arXiv*, 2014.

[51] Stéphane Ross, Geoffrey Gordon, and Drew Bagnell. A reduction of imitation learning and structured prediction to no-regret online learning. In *International Conference on Artificial Intelligence and Statistics*, 2011.

[52] John Schulman, Sergey Levine, Philipp Moritz, Michael Jordan, and Pieter Abbeel. Trust region policy optimization. In *International Conference on Machine Learning*, 2015.

[53] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv*, 2017.

[54] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 2016.

[55] Vikas Sindhwani, Partha Niyogi, and Mikhail Belkin. A co-regularization approach to semi-supervised learning with multiple views. In *ICML workshop on learning with multiple views*, 2005.

[56] Jialin Song, Ravi Lanka, Albert Zhao, Aadyot Bhatnagar, Yisong Yue, and Masahiro Ono. Learning to search via retrospective imitation. *arXiv*, 2018.

[57] Bradly C Stadie, Pieter Abbeel, and Ilya Sutskever. Third-person imitation learning. *arXiv*, 2017.

[58] Wen Sun, Arun Venkatraman, Geoffrey J Gordon, Byron Boots, and J Andrew Bagnell. Deeply aggrevated: Differentiable imitation learning for sequential prediction. In *International Conference on Machine Learning*, 2017.

[59] Richard S Sutton, David A McAllester, Satinder P Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In *Neural information processing systems*, 2000.

[60] Umar Syed, Michael Bowling, and Robert E Schapire. Apprenticeship learning using linear programming. In *International Conference on Machine Learning*, 2008.

[61] Umar Syed and Robert E Schapire. A game-theoretic approach to apprenticeship learning. In *Neural information processing systems*, 2008.

[62] Emanuel Todorov, Tom Erez, and Yuval Tassa. Mujoco: A physics engine for model-based control. In *International Conference on Intelligent Robots and Systems*, 2012.

[63] Hado Van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *AAAI Conference on Artificial Intelligence*, 2016.

[64] Xiaojun Wan. Co-training for cross-lingual sentiment classification. In *Joint conference of ACL and IJCNLP*. Association for Computational Linguistics, 2009.

[65] Wei Wang and Zhi-Hua Zhou. A new analysis of co-training. In *International Conference on Machine Learning*, 2010.

[66] Wei Wang and Zhi-Hua Zhou. Co-training with insufficient views. In *Asian conference on machine learning*, 2013.

[67] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas. Dueling network architectures for deep reinforcement learning. In *International Conference on Machine Learning*, 2016.

[68] Brian Ziebart, Andrew Maas, J Andrew Bagnell, and Anind Dey. Maximum entropy inverse reinforcement learning. In *AAAI Conference on Artificial Intelligence*, 2008.

8 APPENDIX

8.1 PROOFS

Proof for Proposition 1:

Proof. We show that $\max_s D_{JS}(\pi^B(s)\|\pi^A(s))$ is well-defined for an MDP $\mathcal{M}$ with two representations $\mathcal{M}^A$ and $\mathcal{M}^B$. From Theorem 1, we know the distribution $\pi(s)$ can be written with respect to its occupancy measure ρ_π. It is sufficient to show that we can map occupancy measures of π^A and π^B to a common MDP. By the definition of an occupancy measure,

$$\begin{aligned}\rho_\pi(s,a) &= \mathbb{P}(\pi(s)=a)\sum_{i=0}^{\infty}\gamma^i\mathbb{P}(s_i=s|\pi)\\ &= \mathbb{E}_{\tau=(s_0,a_0,\cdots,s_n)\sim\pi}[\sum_{i=0}^{n}\gamma^i\mathbf{1}((s_i,a_i)=(s,a))]\end{aligned}$$

that is to say, the occupancy measure is the expected discounted count of a state-action pair to appear in all possible trajectories. Since we have trajectory mappings between $\mathcal{M}^A$ and $\mathcal{M}^B$, we can convert an occupancy measure in $\mathcal{M}^A$ to one in $\mathcal{M}^B$ by mapping each trajectory and perform the count in the new MDP representation. Formally, the occupancy measure $\rho^B_{\pi^B}$ of π^B in $\mathcal{M}^B$ can be mapped to an occupancy measure in $\mathcal{M}^A$ by

$$\begin{aligned}&\rho^A_{\pi^B}(s,a)\\ &= \mathbb{E}_{\substack{\tau^B\sim\pi^B,\\ f_{B\to A}(\tau^B)=(s_0,a_0,\cdots,s_n)}}[\sum_{i=0}^{n}\gamma^i\mathbf{1}((s_i,a_i)=(s,a))]\end{aligned}$$

Following from this, we can compute $D_{JS}(\pi^B(s)\|\pi^A(s))$ using any s in $\mathcal{M}^A$. And the maximum is defined. In the definition, there is a choice whether to map π^A's occupancy measure to $\mathcal{M}^B$ or π^B's to $\mathcal{M}^A$. Though both approaches lead to a valid definition, we use the definition that for $D_{JS}(\cdot\|\cdot)$, we always map the representation in the first argument to that of the second argument. It is preferable to the other one because in Theorem 2, we want to optimize

$$J(\pi'^A) \geq J_{\pi^A}(\pi'^A) - \frac{2\gamma^A(4\beta^B_{\mathcal{D}_2}\epsilon^B_{\mathcal{D}_2}+\alpha^A_{\mathcal{D}}\epsilon^A_{\mathcal{D}})}{(1-\gamma^A)^2} + \delta_2$$

by optimizing

$$\beta^B_{\mathcal{D}_2} = \mathbb{E}_{\mathcal{M}\sim\mathcal{D}_2}[\max_s D_{JS}(\pi^B(s)\|\pi^A(s))]$$

usually via computing the gradient of $\beta^B_{\mathcal{D}_2}$ w.r.t. π^A. If we use $f_{A\to B}$ to map from $\mathcal{M}^A$ to $\mathcal{M}^B$, the gradient will involve a complex composition of $f_{A\to B}$ and π^A, which is undesirable. □

To prove Theorem 2, we need to use a policy improvement result for a single MDP (a modified version of Theorem 1 in [29]).

Theorem 4. *Assume for an MDP $\mathcal{M}$, an expert policy π_E have a higer advantage of over a policy π with a margin, i.e., $\eta(\pi_E,\mathcal{M})-\eta(\pi,\mathcal{M})\geq\delta$ Define*

$$\begin{aligned}\alpha &= \max_s D_{KL}(\pi'(s)\|\pi(s))\\ \beta &= \max_s D_{JS}(\pi'(s)\|\pi_E(s))\\ \epsilon_{\pi_E} &= \max_{s,a}|A_{\pi_E}(s,a)|\\ \epsilon_\pi &= \max_{s,a}|A_\pi(s,a)|\end{aligned}$$

then $\eta(\pi',\mathcal{M}) \geq \eta_\pi(\pi',\mathcal{M}) - \frac{2\gamma(4\beta\epsilon_{\pi_E}+\alpha\epsilon_\pi)}{(1-\gamma)^2}+\delta$

Proof. The only difference from the original theorem is that the original assumes $\mathbb{E}_{a_E\sim\pi_E(s),a\sim\pi(s)}[A_\pi(s,a_E)-A_\pi(s,a)]\geq\delta'>0$ for every state s. It is a stronger assumption which is not needed in their analysis. Notice that the advantage of a policy over itself is zero, i.e., $\mathbb{E}_{a\sim\pi(s)}[A_\pi(s,a)]=0$ for every s, so the margin assumption simplifies to $\mathbb{E}_{a_E\sim\pi_E(s)}[A_\pi(s,a_E)]\geq\delta'$.

By the policy advantage formula,

$$\begin{aligned}\eta(\pi_E,\mathcal{M})-\eta(\pi,\mathcal{M}) &= \mathbb{E}_{\tau\sim\pi_E}[\sum_{i=0}^{\infty}\gamma^iA_\pi(s_i,a_i)]\\ &= \mathbb{E}_{s_i\sim\rho_{\pi_E}}\mathbb{E}_{a_i\sim\pi_E(s_i)}[\sum_{i=0}^{\infty}\gamma^iA_\pi(s_i,a_i)]\\ &\geq \mathbb{E}_{s_i\sim\rho_{\pi_E}}[\delta'\sum_{i=0}^{\infty}\gamma^i]\\ &= \frac{\delta'}{1-\gamma}\end{aligned}$$

So an assumption on per-state advantage translates to a overall advantage. Thus we can make this weaker assumption which is also more intuitive and the original statement still holds with a different δ term. □

Proof of Theorem 2:

Proof. Theorem 2 is a distributional extension to the theorem above. For $\mathcal{M}\sim\mathcal{D}_2$, let $\delta_\mathcal{M}=\eta(\pi^B,\mathcal{M}^B)-$

$\eta(\pi^A, \mathcal{M}^A)$.

$$
\begin{aligned}
& J(\pi'^A) \\
& = \mathbb{E}_{\mathcal{M}\sim\mathcal{D}}[\eta(\pi'^A, \mathcal{M}^A)] \\
& = \mathbb{E}_{\mathcal{M}\sim\mathcal{D}_1}[\eta(\pi'^A, \mathcal{M}^A)] + \mathbb{E}_{\mathcal{M}\sim\mathcal{D}_2}[\eta(\pi'^A, \mathcal{M}^A)] \\
& \geq \mathbb{E}_{\mathcal{M}\sim\mathcal{D}_1}[\eta(\pi'^A, \mathcal{M}^A)] + \\
& \mathbb{E}_{\mathcal{M}\sim\mathcal{D}_2}[\eta_{\pi^A}(\pi'^A, \mathcal{M}^A) - \frac{2\gamma^A(4\beta\epsilon_{\pi^B} + \alpha\epsilon_{\pi^A})}{(1-\gamma^A)^2} + \delta_{\mathcal{M}}] \\
& \geq \mathbb{E}_{\mathcal{M}\sim\mathcal{D}_1}[\eta_{\pi^A(}(\pi'^A, \mathcal{M}^A) - \frac{2\gamma^A\alpha\epsilon_{\pi^A}}{(1-\gamma^A)^2}] + \\
& \mathbb{E}_{\mathcal{M}\sim\mathcal{D}_2}[\eta_{\pi^A}(\pi'^A, \mathcal{M}^A) - \frac{2\gamma^A(4\beta\epsilon_{\pi^B} + \alpha\epsilon_{\pi^A})}{(1-\gamma^A)^2} + \delta_{\mathcal{M}}] \\
& = \mathbb{E}_{\mathcal{M}\sim\mathcal{D}}[\eta_{\pi^A}(\pi'^A, \mathcal{M}^A)] - \mathbb{E}_{\mathcal{M}\sim\mathcal{D}}[\frac{2\gamma^A\alpha\epsilon_{\pi^A}}{(1-\gamma^A)^2}] - \\
& \mathbb{E}_{\mathcal{M}\sim\mathcal{D}_2}[\frac{2\gamma^A\cdot 4\beta\epsilon_{\pi^B}}{(1-\gamma^A)^2}] + \mathbb{E}_{\mathcal{M}\sim\mathcal{D}_2}[\delta_{\mathcal{M}}] \\
& \geq J_{\pi^A}(\pi'^A) - \frac{2\gamma^A(4\beta^B_{\mathcal{D}_2}\epsilon^B_{\mathcal{D}_2} + \alpha^A_{\mathcal{D}}\epsilon^A_{\mathcal{D}})}{(1-\gamma^A)^2} + \delta_2
\end{aligned}
$$

The derivation for $J(\pi'^B)$ is the same. □

Finally, we provide the proof for Theorem 3. We first quantify the performance gap between a policy π and an optimal policy π^*. For a policy that is able to achieve ϵ $0-1$ loss, $\ell(s,\pi) = \mathbf{1}(\pi(s) \neq \pi^*(s))$, measured against π^*'s action choices under its own state distributions, then we can bound the performance gap. Let $Q_t^{\pi'}(s,\pi)$ denote the t-step cost of executing π in initial state s and then following policy π'

Theorem 5. *(Theorem 2.2 from [51], adpated to our notations) Let π be such taht $\mathbb{E}_{s\sim\rho_\pi}[\ell(s,\pi)] = \epsilon$, and $Q^{\pi^*}_{T-t+1}(s,\pi^*) - Q^{\pi^*}_{T-t+1}(s,a) \leq u$ for all action $a, t \in \{1, 2, \cdots, T\}$, then $\eta(\pi, \mathcal{M}) \geq \eta(\pi^*, \mathcal{M}) - uT\epsilon$.*

Thus the important quantity to measure is ϵ, and by measuring the disagreements between two policies in two views, we can upper bound ϵ. The result is originally stated in the context of classification, and the above theorem justifies the learning reduction approach of reducing policy learning to classification.

Theorem 6. *(Corollary 5 in [16] applied to full classifiers) Using the definitions in Theorem 3, with probability $1-\sigma$ over the choice of a sample set N, for all pairs of classifiers h_1, h_2 such that for all i we have $\zeta_i(h_1, h_2, \sigma) > 0$ and $b_i(h_1, h_2, \sigma) \leq 1$.*

$$\epsilon \leq \max_{j\in\{1,\cdots,k\}} b_j(h_1, h_2, \sigma)$$

Proof. The only change from the original proof is that instead of a partial classifier which can output $\perp$, we consider a full classifier. Then we could eliminate the estimates for $\mathbb{P}(h_1 \neq \perp)$ and the error introduced by converting a partial classifier to a full classifier via random labelling when the output is $\perp$. □

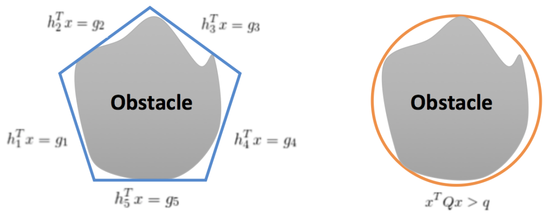


Figure 7: Two views for Risk-Aware Path Planning. On the left, the obstacle is enclosed by a polytope (MILP view) and on the right the obstacle is enclosed by an ellipse (QCQP view).

Proof of Theorem 3:

Proof. For the bound for π^A, we are measuring ϵ_A on its sampled paths. Then directly apply Theorem 6 gives an upper bound on ϵ_A. Apply Theorem 5 gives the result of Theorem 3. □

8.2 PICTORIAL REPRESENTATION OF THE TWO-VIEWS IN RISK-AWARE PATH PLANNING:

We present a pictorial representation of the two different views used in the experiments in Fig 7. In the MILP view, the constraint space is represented using additional auxiliary binary variables to choose the active side of the polytope, whereas in the QCQP view, the constraint space can be encoded in a quadratic constraint.

8.3 RISK-AWARE PLANNING DATASET GENERATION:

We generate 150 obstacle maps. Each map contains 10 rectangle obstacles, with the center of each obstacle chosen from a uniform random distribution over the space $0 \leq y \leq 1, 0 \leq x \leq 1$. The side length of each obstacle was chosen from a uniform distribution in range [0.01, 0.02] and the orientation was chosen from a uniform distribution between 0° and 360°. In order to avoid trivial infeasible maps, any obstacles centered close to the destination are removed. For MILP view, we directly use the randomly generated rectangles for defining the constraint space. However, for the QCQP view, we enclose the rectangle obstacles with a circle for defining the quadratic constraint.

8.4 DISCRETE/CONTINUOUS CONTROL RESULTS IN TABULAR FORM

	Acrobot	Swimmer	Hopper
A (CoPiEr)	-86.44 ± 10.80	106.35 ± 23.11	217.83 ± 30.03
A (PG)	-169.57 ± 10.48	109.09 ± 21.58	278.66 ± 32.87
A (All)	-252.42 ± 8.73	100.36 ± 22.37	49.39 ± 10.35
B (CoPiEr)	-88.48 ± 15.13	104.16 ± 19.32	168.88 ± 18.21
B (PG)	-257.16 ± 10.93	103.48 ± 21.89	89.34 ± 4.89
B (All)	-251.74 ± 9.65	96.74 ± 19.57	22.59 ± 5.55
A + B	-86.42 ± 3.48	108.71 ± 5.03	346.53 ± 5.91