CAP6671 Intelligent Systems

Lecture 14:

Transfer for Reinforcement Learning

Instructor: Dr. Gita Sukthankar
Email: gitars@eecs.ucf.edu

Schedule: T & Th 9:00-10:15am

Location: HEC 302

Office Hours (in HEC 232):
T & Th 10:30am-12

Strengths/Problems of Paper?

Strengths/Problems of Paper?

Strengths

- Very interesting problem and approach
- Transfer learning has typically been applied to problems with same sensors/actions

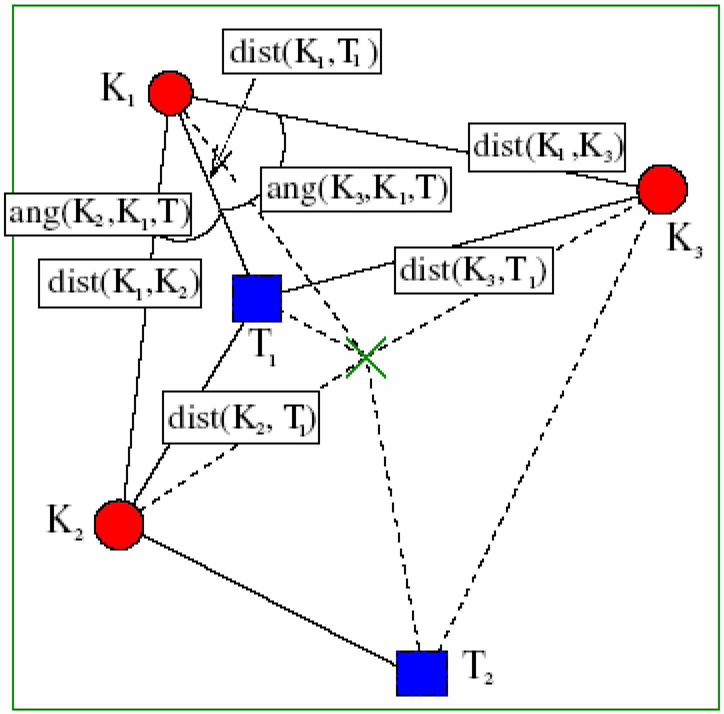
Weaknesses

- Creating cross-domain mappings seems difficult
- Devising these alternate domains also seems non-intuitive

Rule Transfer

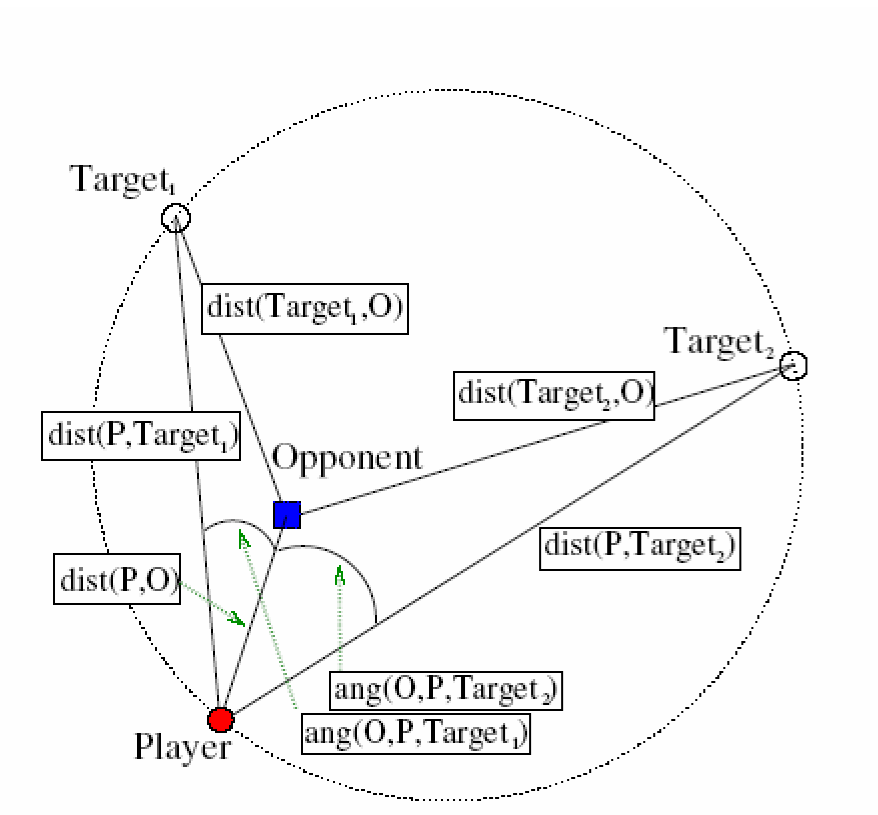
1. Learn a policy for the source task
2. Learn a decision list for the source policy
3. Modify the decision list for use in the target task
4. Use decision list to learn a policy in the target domain

Keepaway



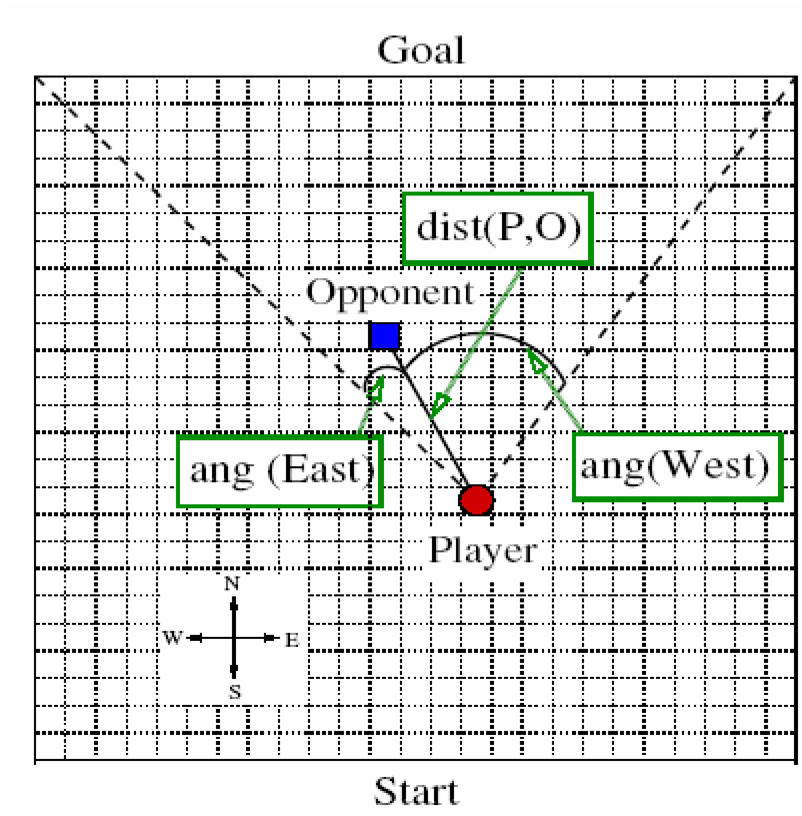
- 3 Keepers prevent 2 Takers from intercepting the ball
- Learn policy for Keeper holding the ball
- A={hold, Pass1, Pass2}
- Takers follow fixed strategy
- Keepers without ball either 1) attempt to capture an open ball or 2) get free for a pass
- Reward per timestep ball is in play
- Simulated noise in perception

Ringworld



- Opponent moves towards player on every timestep
- Player can either stay in current location or run towards a target
- As opponent approaches player the probability of the player being tagged increases
- A={Stay, RunNear, RunFar}
- Size of ring/prob of tagging chosen to be similar to Keepaway
- Reward per timestep

Knight Joust



- Players alternate moves on a grid board
- Player can either move directly north or knight's move east or west
- Players' moves are deterministic and opponent has a fixed stochastic policy
- Similarity: favor distance between player and opponent
- Reward for advancing distance

Translation between Tasks

- Define translation functions between state variables and actions

Cross-Domain Mappings for Ringworld to Keepaway

Ringworld	**Keepaway**
	δ_A
Stay	Hold Ball
Run_{Near}	$Pass_1$: Pass to K_2
Run_{Far}	$Pass_2$: Pass to K_3
	δ_X
$dist(P, O)$	$dist(K_1, T_1)$
$dist(P, Target_1)$	$dist(K_1, K_2)$
$dist(Target_1, O)$	Min$(dist(K_2, T_1), dist(K_2, T_2))$
$ang(O, P, Target_1)$	Min$(ang(K_2, K_1, T_1)$ $ang(K_2, K_1, T_2))$
$dist(P, Target_2)$	$dist(K_1, K_3)$
$dist(Target_2, O)$	Min$(dist(K_3, T_1), dist(K_3, T_2))$
$ang(O, P, Target_2)$	Min$(ang(K_3, K_1, T_1),$ $ang(K_3, K_1, T_2))$

Rule Utilization

- **Value Bonus**: give constant bonus to Q-value as recommended by the translated decision list
- **Extra Action**: add action to target task such that when the agent selects this pseudo-action it follows the action recommended by D (have exploration policy favor this action)
- **Extra Variable**: add extra state variable to target state description that takes on the value of the index for the action recommended by D (have exploration policy favor this action)

RL Method

- SARSA: "State-Action State-Reward-State Action"
- Learning rule uses 2-step lookahead instead of expected value

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \phi Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)]$$

- Remember: standard Q-learning rule

$$Q(s,a) := Q(s,a) + \alpha(r + \gamma \max_{a'} Q(s', a') - Q(s,a))$$

- Radial basis function approximation to handle continuous state space

Procedure

- Use SARSA to learn Q-funciton for source domain
- Learn decision list summarizing source task policy (RIPPER, rule induction algorithm)
- Use decision list to train an agent in the target domain
- Measure
  - Initial performance
  - Asymptotic performance after learning plateaus (40 simulator hours)
  - Accumulated reward (sum of average reward per hour)

RIPPER

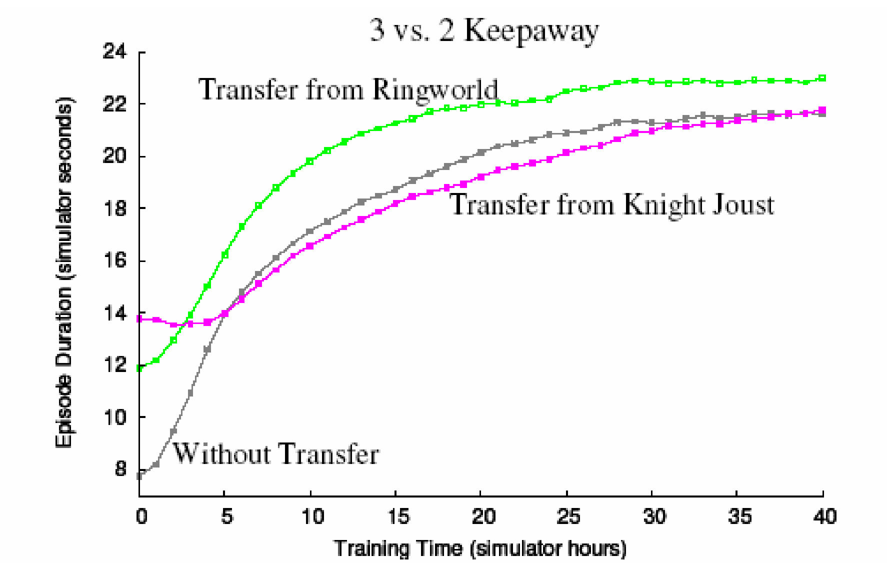
- Rule induction algorithm that improves on the efficiency of IREP
- Split training data into growing set and pruning set
- GrowRule: add conditions to an empty conjunction
- PruneRule: deleting conditions from the rule to make it more general
- Example rule produced by algorithm

IF ((dist(K_1, T_1) <= 4) AND
(Min($dist(K_3, T_1), dist(K_3, T_2)$) >= 12.8) AND
($ang(K_3, K_1, T)$ >= 36)) THEN Pass to K_3

Evaluations

- Evaluate transfer from Keepaway to Keepaway to determine reasonable parameters
- Transfer of Ringworld to Keepaway produced benefits in all 3 metrics
- Transfer of KnightJoust to Keepaway only improves initial performance
- All 3 rule utilization schemes were effective with ExtraAction being slightly superior
- Also did a sensitivity analysis to show that the learning is not that dependent on parameters of RIPPER

Transfer Results



Ringworld to Keepaway

	Initial Performance	Asymptotic Performance	Accumulated Reward
	Without Transfer		
	7.8 ± 0.1	21.6 ± 0.8	756.7 ± 21.8
Added Constant	**Value Bonus**		
5	**11.1 ± 1.4**	19.8 ± 0.6	722.3 ± 24.3
10	**11.5 ± 1.7**	**22.2 ± 0.8**	**813.7 ± 23.6**
Initial Episodes	**Extra Action**		
100	**11.9 ± 1.8**	**23.0 ± 0.5**	**842.0 ± 26.9**
250	**11.8 ± 1.9**	**23.0 ± 0.8**	**827.4 ± 33.0**
Initial Episodes	**Extra Variable**		
100	**11.8 ± 1.9**	21.9 ± 0.9	**784.8 ± 27.0**
250	**11.7 ± 1.8**	**22.4 ± 0.8**	**793.5 ± 22.2**

Knight Joust into Keepaway

Param	Initial Performance	Asymptotic Performance	Accumulated Reward
	Without Transfer		
	7.8 ± 0.1	21.6 ± 0.8	756.7 ± 21.8
	Extra Action		
100	**13.8 ± 1.1**	21.8 ± 1.2	758.5 ± 29.3
250	**13.5 ± 0.9**	21.6 ± 0.9	747.9 ± 25.3

Future Work

- Want to be able to automatically derive the rule translation function
- General approach for deriving translation:
  - Identify state variables that are near 0 when episode ends
  - Identify variable that causes those variables to decrease
  - Construct mapping between other distances and angles
- Drawback: still seems fairly awkward and not possible to fully automate it

Other Ideas?