Reinforcement Learning and Utility-Based Decisions

Michael L. Littman

Rutgers University

Department of Computer Science

Rutgers Laboratory for Real-Life Reinforcement Learning

Outline

- One view of utility-based data mining
- Parallels with PAC reinforcement learning
- Survey of PAC RL results
- Lame attempt to relate UBDM and RL

What is Utility as it relates to Data Mining?

- In the context of data mining, utility refers to total utility derived from the entire data mining *process*
  - It factors in utilities from 3 stages of data mining:
    1. Costs of acquiring the data (U1 or C1)
    2. Costs of mining the data (U2 or C2)
    3. Benefits of using the mined knowledge (U3 or B)

$$\text{Utility}_{DM} = U1+U2+U3 \quad (U1, U2 \leq 0; U3 \geq 0)$$
$$= B - (C1 + C2)$$

Note the definition of data mining refers only to U3, "potentially useful patterns"

copied from Gary Weiss

Two Problems

- UBDM: Act so as to maximize the total benefit of using the mined knowledge minus the costs of acquiring and mining the data.

- Reinforcement learning: Act to maximize the utility of behavior, while minimizing experience and computational costs.

k-Armed Bandits

- Perhaps the simplest possible RL problem.

- *k* bandits.
- each step *t*, agent chooses an arm/action a_t
- receives payoff r_t
- expected value of r_t is $R(a_t)$
- optimal behavior is $a_t = \text{argmax}_a R(a)$
- $R(a)$ unknown; some experimentation needed

Relaxations of the Utility Problem

- <u>UBDM</u>: Act so as to maximize the total benefit of using the mined knowledge minus the costs of acquiring and mining the data.
- <u>Reinforcement learning</u>: Act to maximize the utility of behavior, while minimizing experience and computational costs.
- Joint minimality intractable. Instead, satisficing:
  - near-optimal utility
  - polynomial-bounded experience
  - polynomial-bounded computation

PAC Version* of Bandit

- Given $\epsilon>0$, $\delta>0$, k arms.
- We say a strategy makes a <u>mistake</u> each timestep t it selects an action in which $R(a_t) < \max_a R(a)-\epsilon$. utility of behavior
- Let m be a bound on the number of mistakes that holds with probability $1-\delta$. experience
- We want m to be polynomial in k, $1/\epsilon$, $1/\delta$.
- Each decision should be similarly bounded.

* There are many equivalent definitions!

computational

A PAC Algorithm

- Naïve (Round Robin!)
  - Select each arm c times.
  - Average resulting rewards to estimate R(a).
  - Choose $\max_a r(a)$ (where r(a) is the estimate).
- Analysis
  - Hoeffding bound shows how to set c so r(a)s accurate with sufficient prob. ($\approx k \ln(1/\delta)/\epsilon^2$).
- All explore, all exploit.

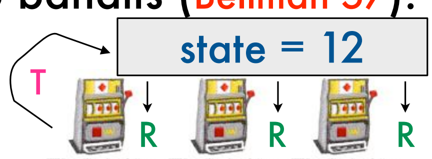
More Elegant PAC Algorithm

- Interval estimation (IE, Kaelbling 93)
  - Estimate mean and confidence interval of arms.
  - Choose max $_a$ (r(a) + interval(a))
    (where r(a) is the mean and interval(a) is the CI).
- Analysis (Fong 95)
  - Chooses an arm if known good or unknown.
  - No worse than Naïve .
- Blends explore/exploit.

- Strategy: "Best of all possible worlds"

Markov Decision Processes

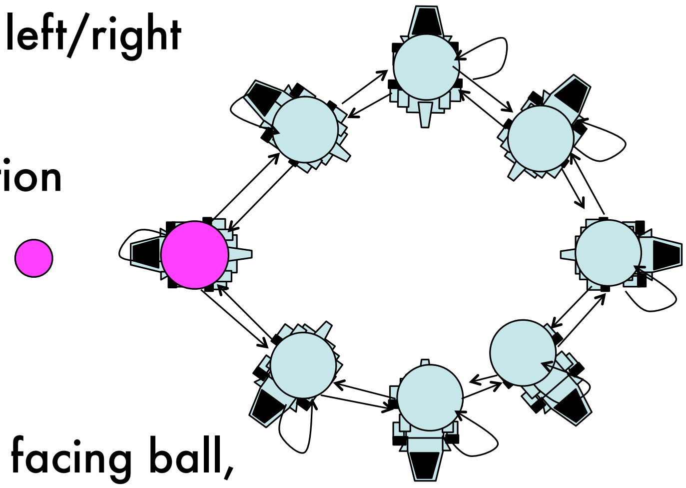
- Brings sequentiality to bandits (Bellman 57).



- n states, k actions
- step t, agent informed state is s_t, chooses a_t
- receives payoff r_t; expected value is $R(s_t, a_t)$
- probability that next state is s' is $T(s_t, a_t, s')$
  $Q(s,a) = R(s,a) + \gamma \sum_{s'} T(s,a,s') \max_{a'} Q(s',a')$
- Optimal behavior is $a_t = \operatorname{argmax}_a Q(s_t,a)$
- R, T unknown; some experimentation needed

Find the Ball: MDP Example

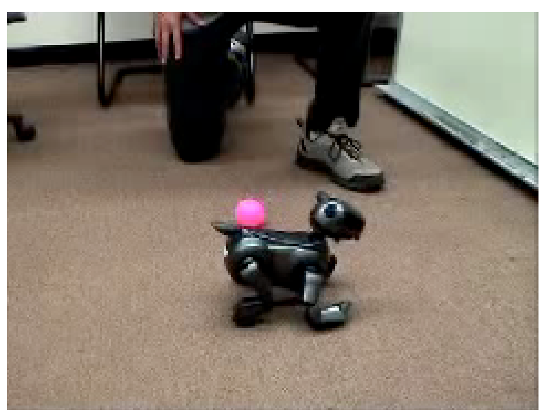
- Actions: rotate left/right
- States: orientation
- Reward: $+1$ for facing ball,
  0 otherwise



Find The Ball

Learn:

- which way to turn
- to minimize time
- to see goal (ball)
- from camera input
- given experience.



Flavors of RL Algorithms

Model-based

- Estimate T, R; solve approximate MDP.
- Prioritized sweeping, Dyna

Value-function-based

- Use observed transitions to modify Q itself.
- Q-learning, SARSA

Policy search

- Try out different policies to find the best.
- policy gradient, genetic approaches

Achieving PAC Bounds

E^3: explicit explore exploit (Kearns & Singh 02)

- Model-based, distinguishes "known/ unknown" transitions/rewards (seen c times)
- Plans in approximate model: value of staying in known states, time to "escape"

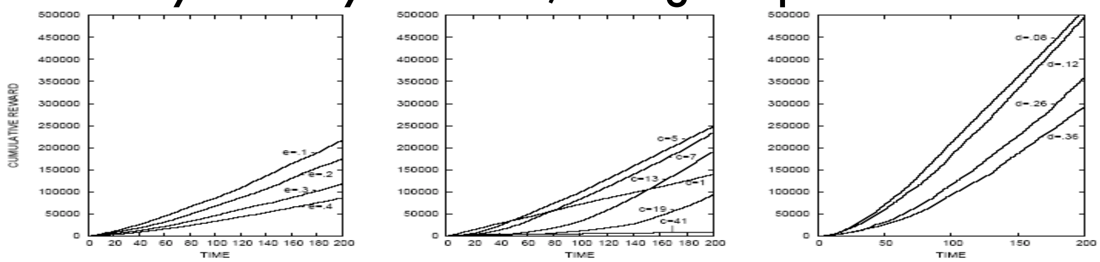
R_{MAX} (Brafman & Tennenholtz 02)

- Same idea, only simpler
- Unknown transitions assumed to yield maximum reward (R_{MAX})

Model-based Interval Estimation

MBIE (Strehl & Littman, 05, 06)

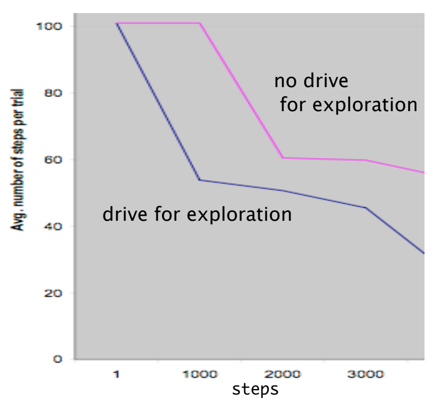
- R_{MAX} like Naïve algorithm for bandits: must try action c times (in each state) to estimate.
- MBIE like IE: transition has confidence interval; assume best of all possible worlds.
- Polynomially solvable, though expensive.



ϵ-greedy R-Max MBIE

Exploration Speeds Learning

Task: Exit room using bird's-eye state representation.



Details: Discretized 15x15 grid x 18 orientation (4050 states);
6 actions. Rewards via R_{MAX} (Brafman & Tennenholtz 02).

Model-Free PAC?

- E^3, R_{MAX}, MBIE all PAC, all model based
- States/actions, sample complexity: $O(n^2\, k)$.
- Seems necessary: $T(s,a,s')$ size $n^2\, k$.

- Can a model-free approach be PAC?
- Is $O(n\, k)$ possible?
- Is Q-learning PAC?

- Set out to prove no...

Delayed Q-learning

Sketch:

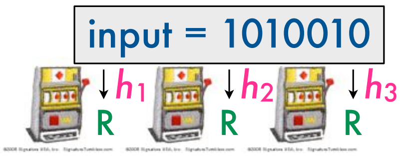
- Q values initialized high.
- Q-learning updates in batches of c.
- Only if update significantly decreases value.
- Greedy action selection.
- Details to make the proof go through.

$O(nk)$ sample, space, $O(\lg k)$ computation (Strehl, Li, Wiewiora, Langford, Littman 06).

Appears impractical...

Associative Bandits

- Brings generalization to bandits (Kaelbling 93).



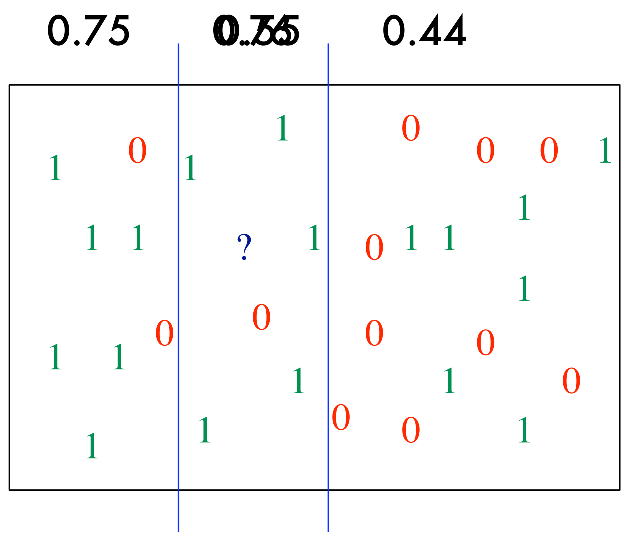
- inputs X, k actions; hypothesis class H
- step t, agent informed input is x_t, chooses a_t
- payoff r_t; expected value is $R_i(a_t)$; $i = h_{at}(x_t)$
- x_t selected iid from a fixed distribution
- Best choice is $a_t = \text{argmax}_a\ R_i(a)$; $i = h_a(x_t)$
- h_{at}, R unknown; some experimentation needed

Main Idea: Reductions

- Associative Bandit
  - which arm to pull?
- Associative Prediction
  - estimate each arm, take best (Naïve)
- Cost-sensitive Classification
  - treat prediction as classification with mistake cost
  - right cost gets right classifier, then R is easy
- Classification
  - many classification algs; modifiable for costs
  - few provably PAC, though

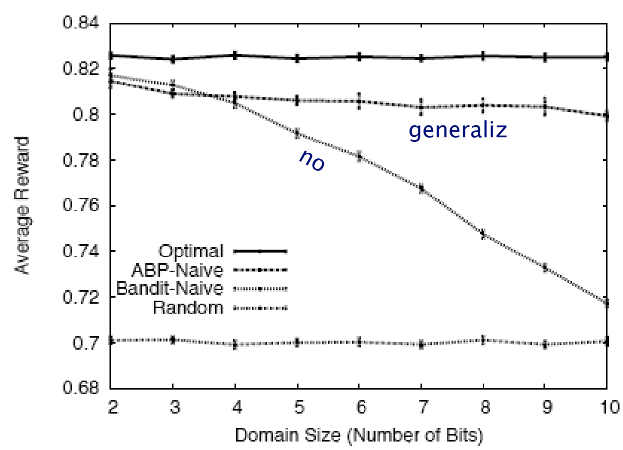
Visualization

- Single arm, what's the payoff at "?" ?
- *X*: rectangle, *H*: vertical dividers
- Each hypothesis leads to estimated payoffs.
- Right one is that with minimum cost (maximum contrast).
- So, ? = 0.76.



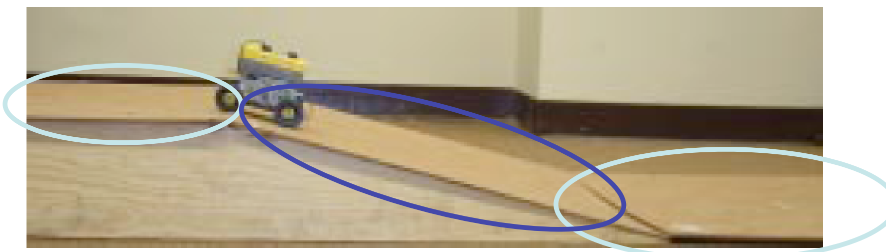
Implemented Example

- Inputs: *n*-bit patterns (n = 2 to 10).
- Hypothesis class: conjunctions of literal pairs
- $k = 2$ arms; $h_1 = h_2 = x_1$ and not(x_n).
- $R_1(1) = .5$,
  $R_0(1) = .8$,
  $R_1(2) = .9$,
  $R_0(2) = .6$
- m=3000 trials



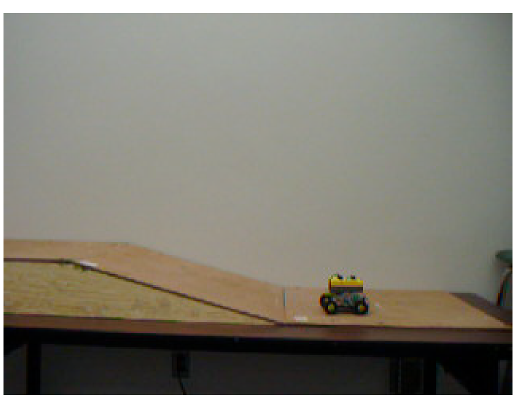
Robotic Example (Leffler, Littmam, Strehl, Walsh 05)

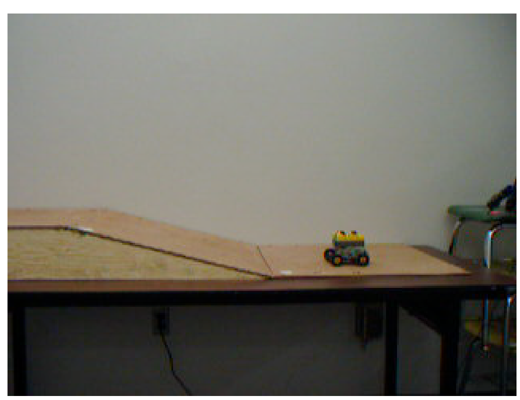
- Input: 18 different locations along a track
- Two underlying classes (flat, up)
- Hypothesis class: all subsets
- Clusters locations based on action outcomes
- Theoretical/experimental advantage over non-generalizing approach



Movie

- Learns to hold consistent speed.





Aside: Closing The Loop

Cost-sensitive classification

- Query an attribute: Cost to learn its value.
- Choose class: Cost for wrong choice.
  - Ends game.

Cost-sensitive fault remediation

- Query an attribute: Cost to learn its value.
- Choose class: Cost to learn its outcome.
  - Ends game if correct, otherwise games continues!

Subtle distinction; opens door for autonomous learning.

CSFR Example

Network repair example (Littman, Ravi, Fenson, Howard 04).

- Recover from corrupted network interface config.
- Minimize time to repair.
- Info. gathering actions: PluggedIn, PingIP, PingLhost, PingGateway, DNSLookup, …
- Repair actions: *RenewLease*, *UseCachedIP*, *FixIP*.

Additional information helps to make the right choice.

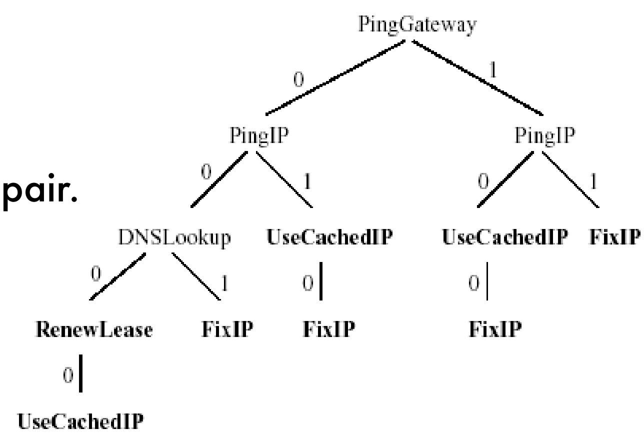
Never know why things failed, just that it's working.

Learning Network Troubleshooting

Recovery from corrupted network interface configuration.

Java/Windows XP: Minimize time to repair.

After 95 failure episodes



Conclusion

- Including data collection and computation with the utility of the outcome of learning is admirable.
- Likely to be intractable without relaxing.
- Idea: Instead of jointly minimizing, keep quantities within bounds.
- Practical algorithms, apply idea to UBDM?