Integrating Background Knowledge and Reinforcement Learning for Action Selection

John E. Laird

Nate Derbinsky

Miller Tinkerhess

UNIVERSITY OF MICHIGAN

Goal

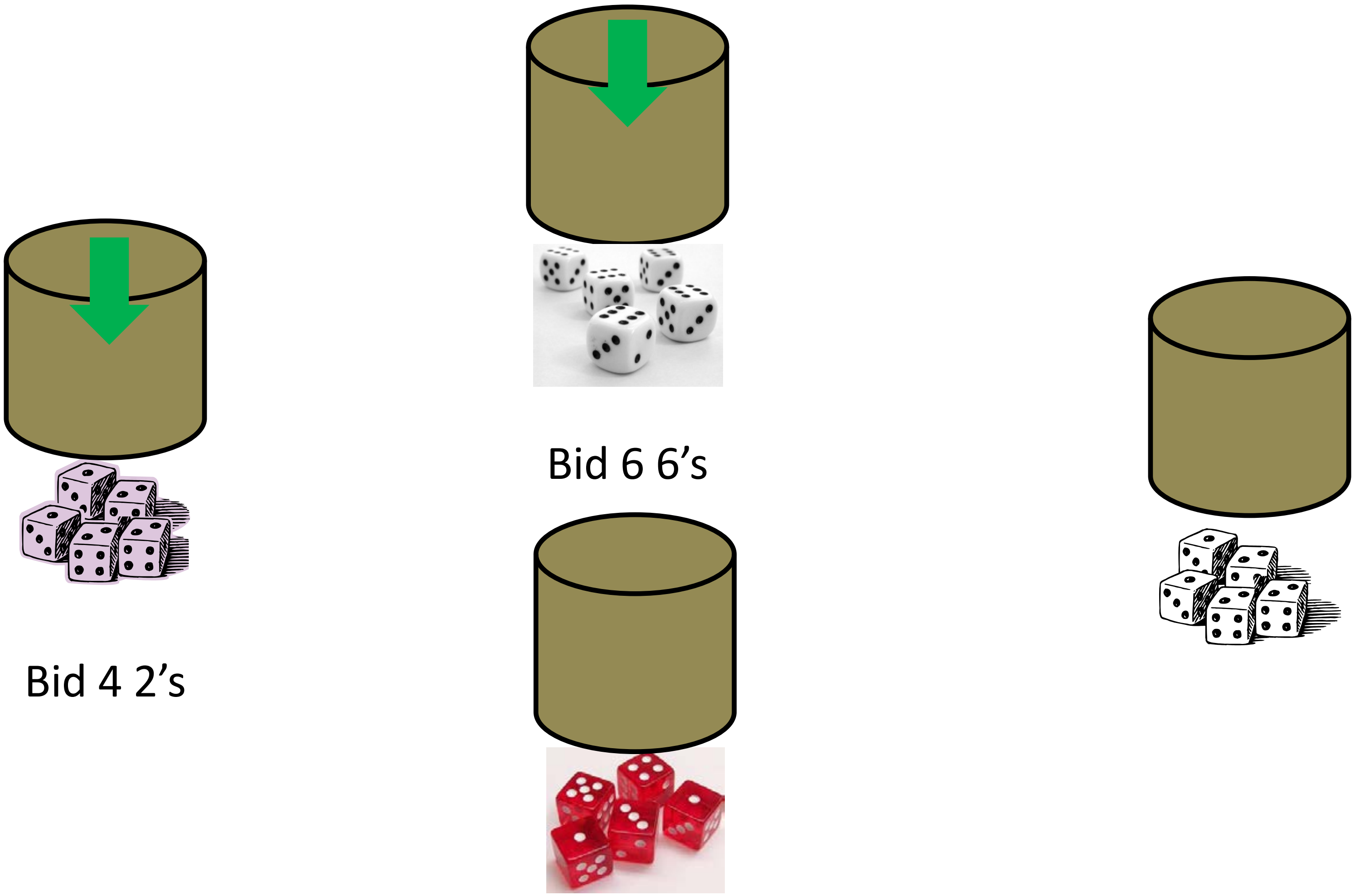
- Provide an architectural support so that:
  - Agent uses (possibly heuristic) background knowledge to initially make action selections
    - Might be non determinism in the environment that is hard to build a good theory of.
  - Learning improves action selections based on experienced-based reward
    - Captures regularities that are hard to encode by hand.

- Approach: Use chunking to learn RL rules and then use reinforcement learning to tune behavior.

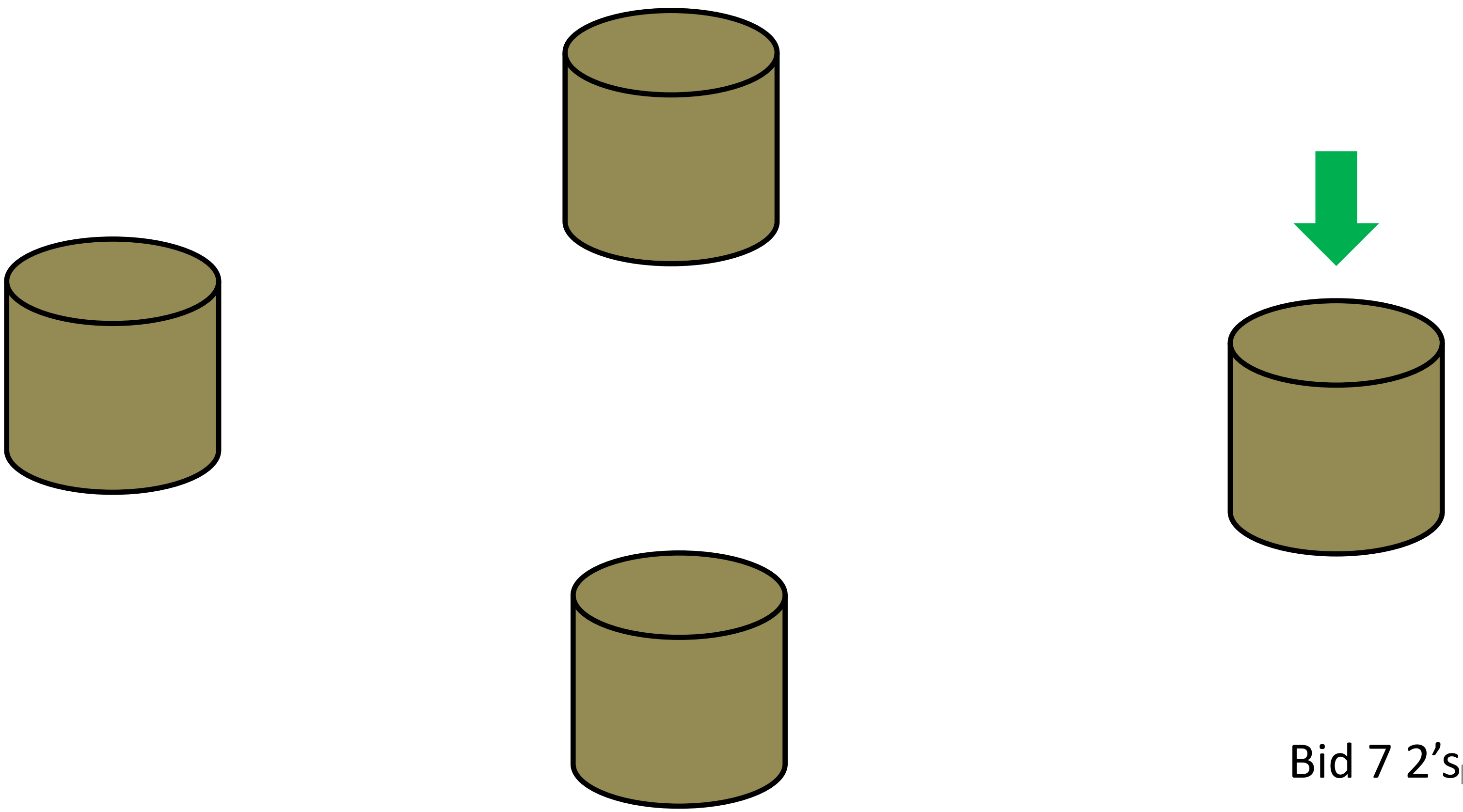
Deliberate Background Knowledge for Action Selection

- Examples:
  - Compute the likelihood of success of different actions using explicit probabilities.
  - Look-ahead internal search using an action model to predict future result from an action
  - Retrieve from episodic memory of similar situation
  - Request from instructor
- Characteristics
  - Multiple steps of internal actions
  - Not easy to incorporate accumulated experienced-based reward
- Soar Approach
  - Calculations in a substate of a tie impasse

[illegible] number of dice under cups.



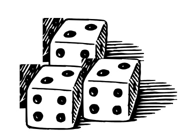
Players can "push" out a subset of their dice and reroll when bidding.



Player can Challenge previous bid.
All dice are revealed

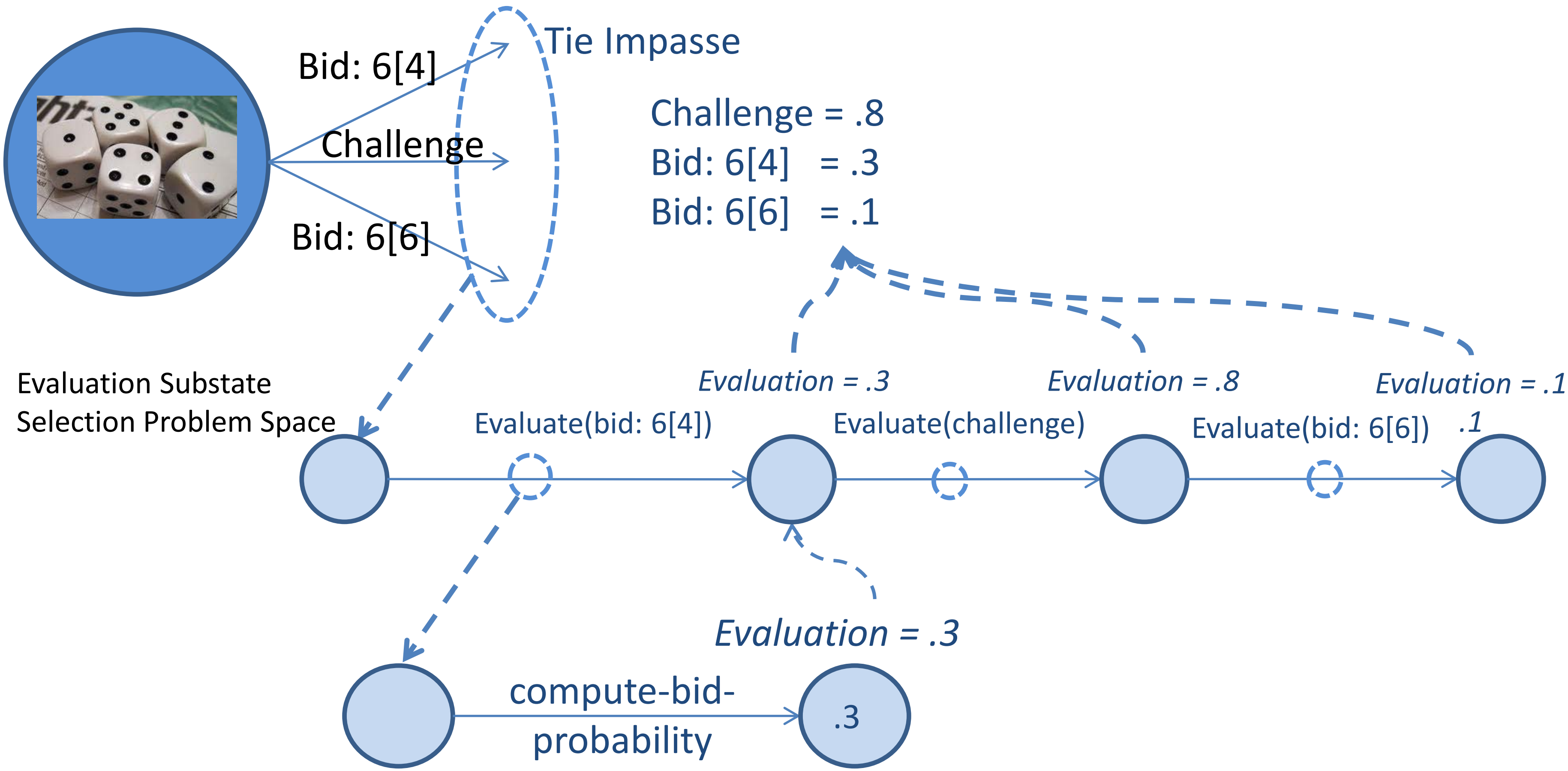
Challenge!

Challenge fails
player loses die



I bid 7 2’s

Evaluation with Probability Calculation

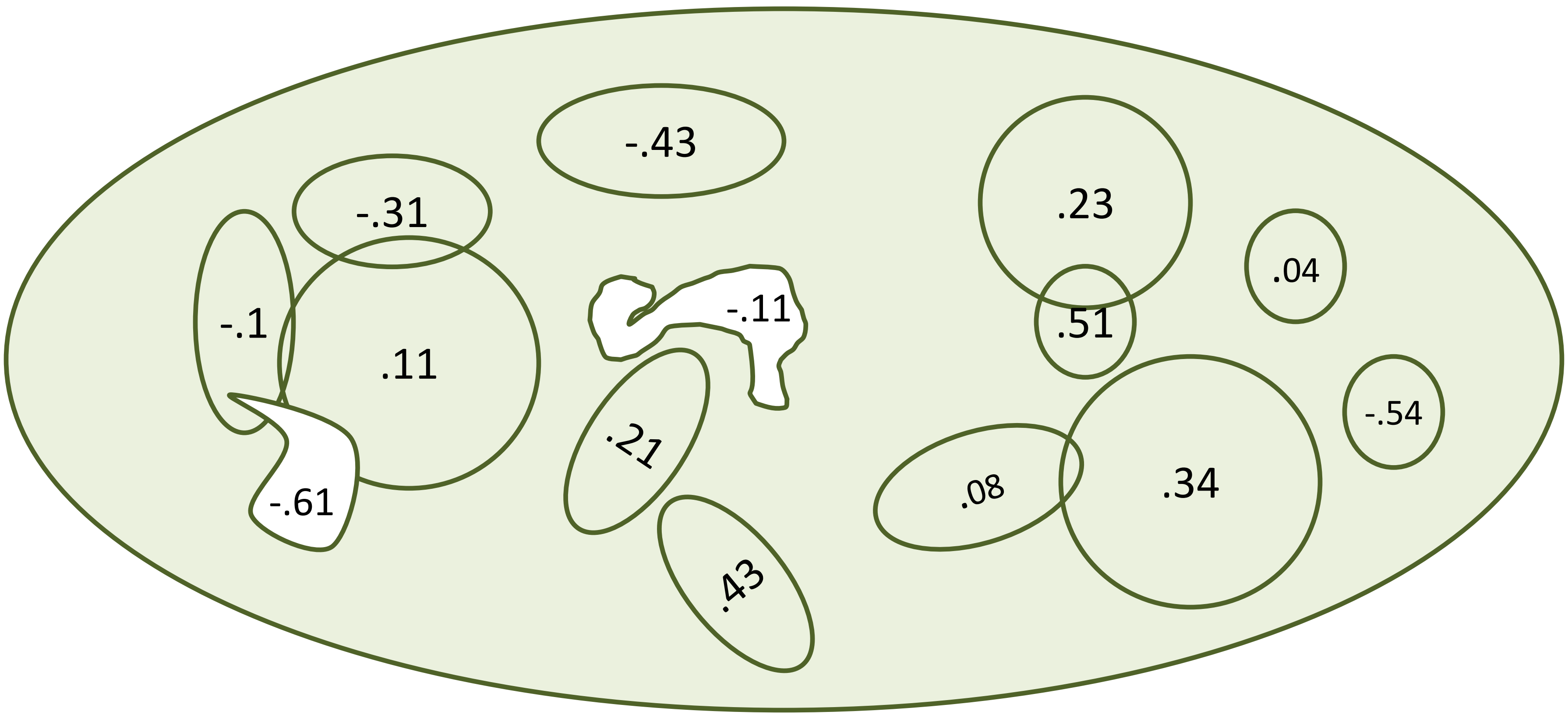


Using Reinforcement Learning for Operator Selection

- Reinforcement Learning
  - Choosing best action based on expected value
  - Expected value updated based on received reward and expected future reward
- Characteristics
  - Direct mapping between situation-action and expected value (value function)
  - Does not use any background knowledge
  - No theory of original of initial values (usually 0) or value-function
- Soar Approach
  - Operator selection rules with numeric preferences
  - Reward received base on task performance
  - Update numeric preferences based on experience
- Issues:
  - Where do RL-rules come from?
    - Conditions that determine the structure of the value function
    - Actions that initialize the value function

Value Function in Soar

- Rules map from situation-action to expected value.
  - Conditions determine generality/specificity of mapping



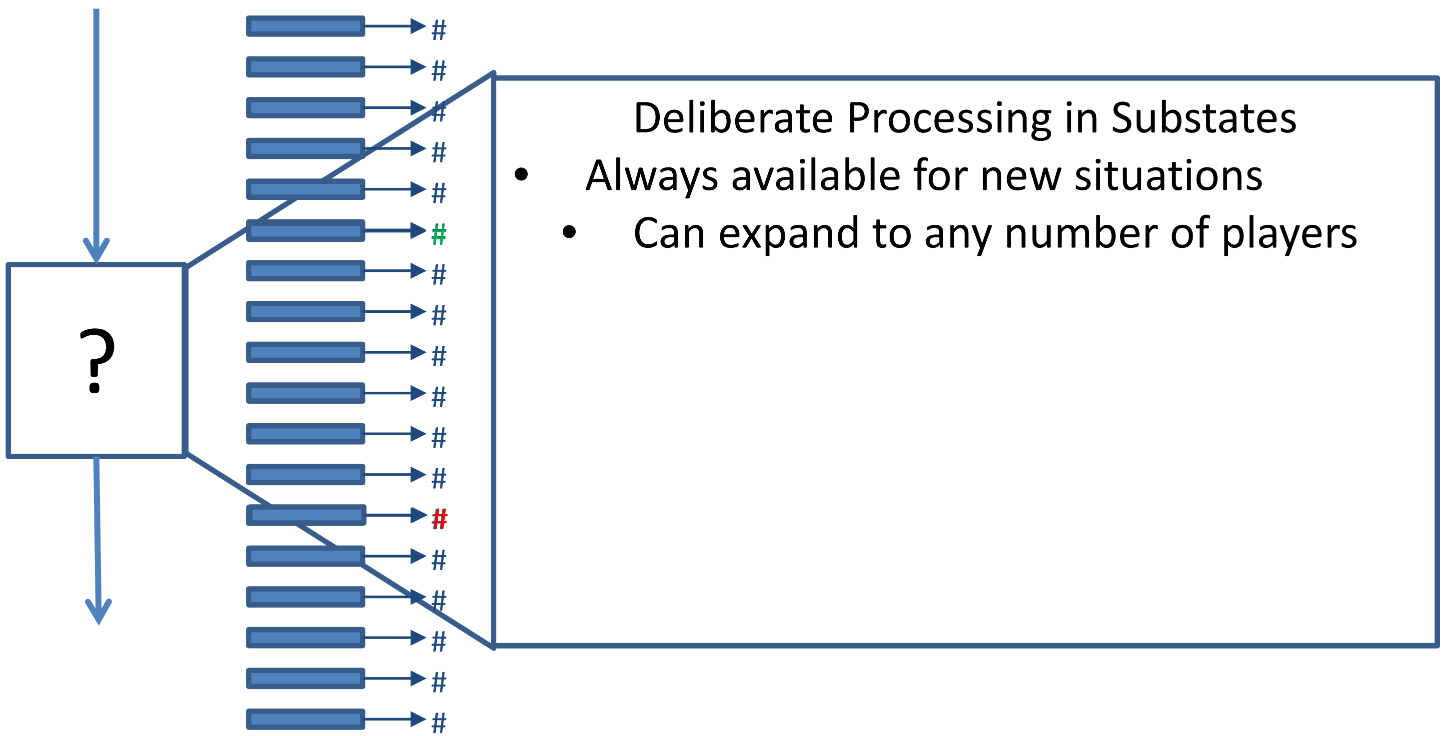
Approach:
Using Chunking over Substate

- For each preference created in the substate, chunking (EBL) creates a new RL rule
  - Actions are numeric preference
  - Conditions based on working memory elements tested in substate

- Reinforcement learning tunes rules based on experience

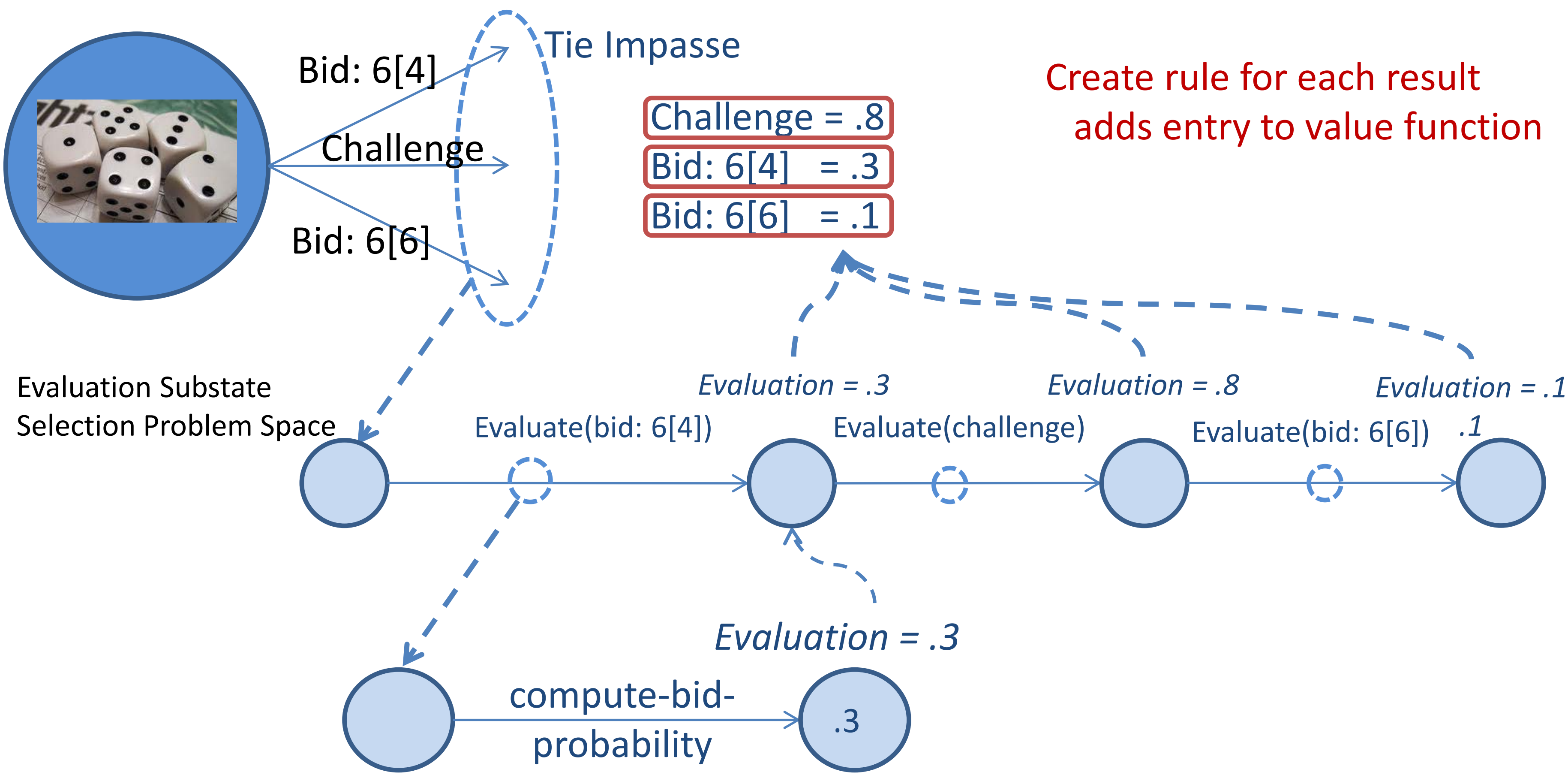
Two-Stage Learning

Select action



RL rules updated based
on agent's experience

Evaluation with Probability Calculation



Learning RL-rules

Using only probability calculation

Bid: 6 4's?

Last bid: 5 4's

Last player:
4 unknown

Shared info
9 unknown
Pushed dice
3 3 4 4

Dice in my cup
3 4 4

4 4's

Need 2 4's

2 4's out of
9 unknown? → .3

If 9 unknown dice, 2 4's pushed and 2 4's under my cup, and considering biding 6 4's, expected value is .3

Learning RL-rules

Using probability and model

Bid: 6 4’s?

Last bid: 5 4’s

Last player:
4 unknown

Shared info
9 unknown

Pushed dice
3 3 4 4

Dice in my cup
3 4 4

7 unknown
2 4’s shared
2 4’s under his cup

Need 1 4 out of
5 unknown

.60

If 9 unknown dice, 2 4’s pushed and 2 4’s under my cup, and previous player had 4 unknown dice and bid 5 4’s, and I’m considering biding 6 4’s, the expected value is .60

Research Questions

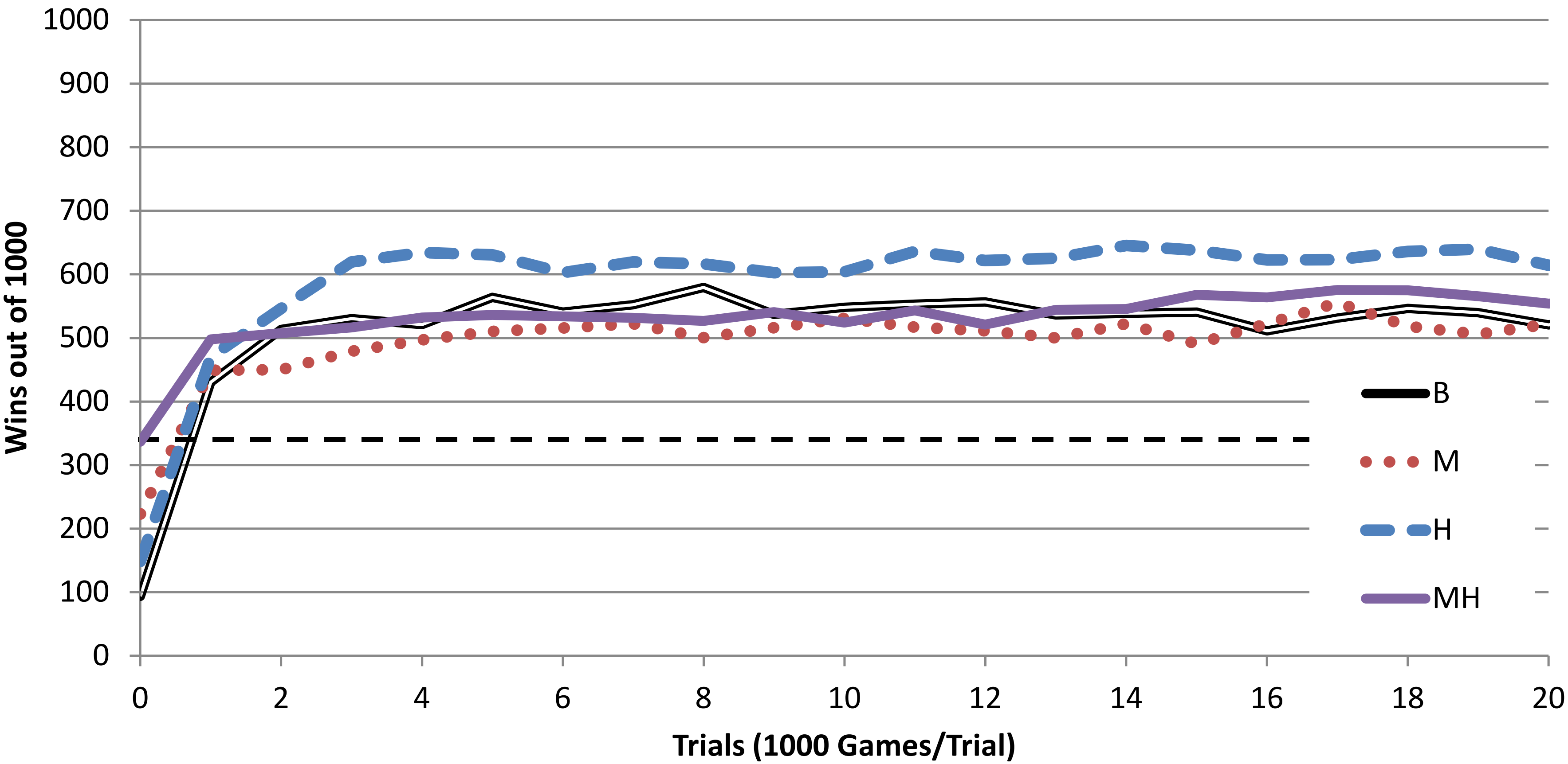
- Does RL help?
  - Do agents improve with experience?
  - Can learning lead to better performance than the best hand-coded agent?

- Does initialization of RL rules improve performance?
- How does background knowledge affect rules learned by chunking and how do they affect learning?

Evaluation of Learning

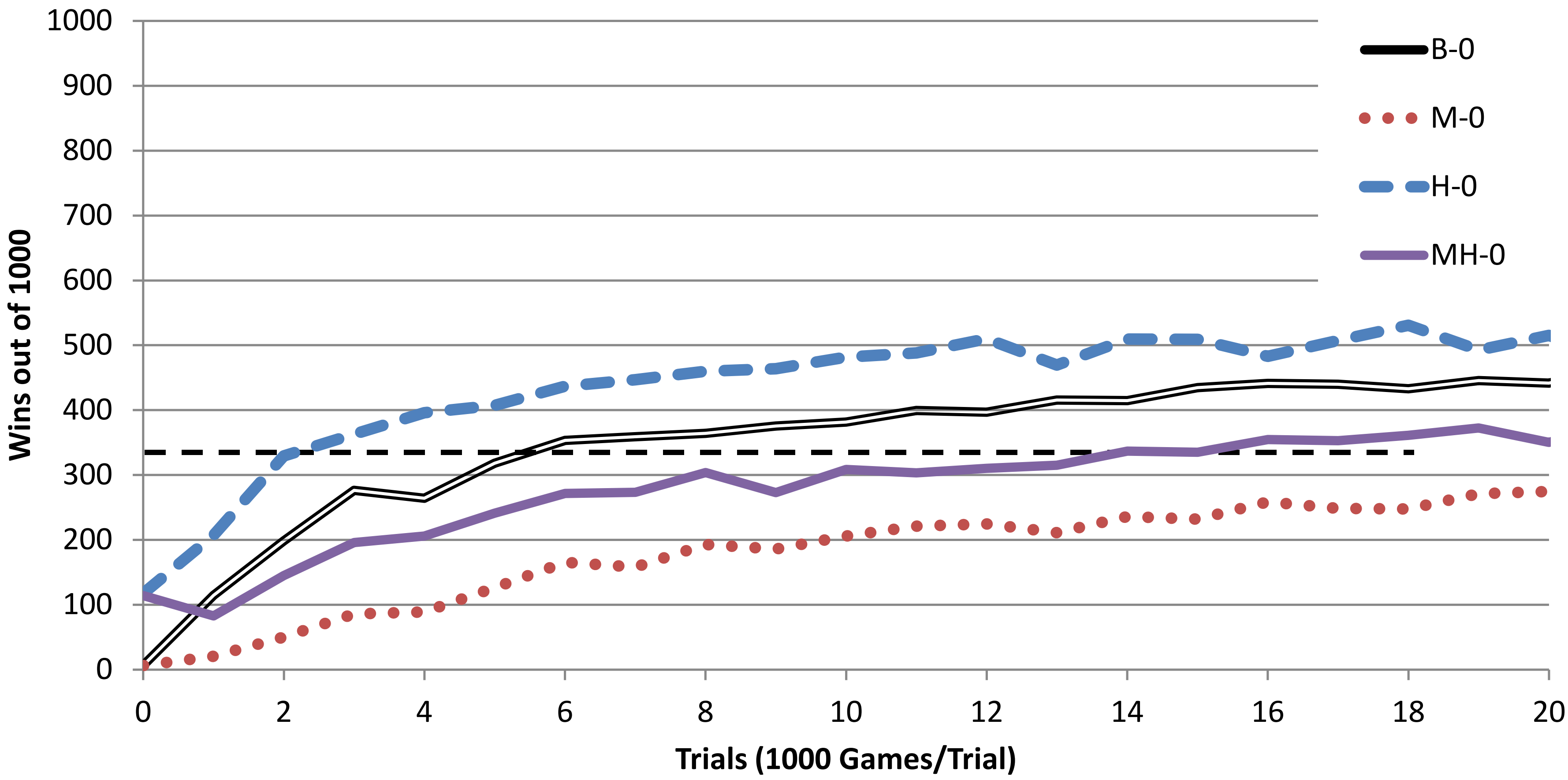
- 3-player games
  - Against best non learning player agent
    - Heuristics and opponent model
  - Alternate 1000 game blocks of testing and training
- Metrics
  - Speed of learning & asymptotic performance
- Agent variants:
  - B: baseline
  - H: with heuristics
  - M: with opponent model
  - MH: with opponent model and heuristics

Learning Agent Comparison

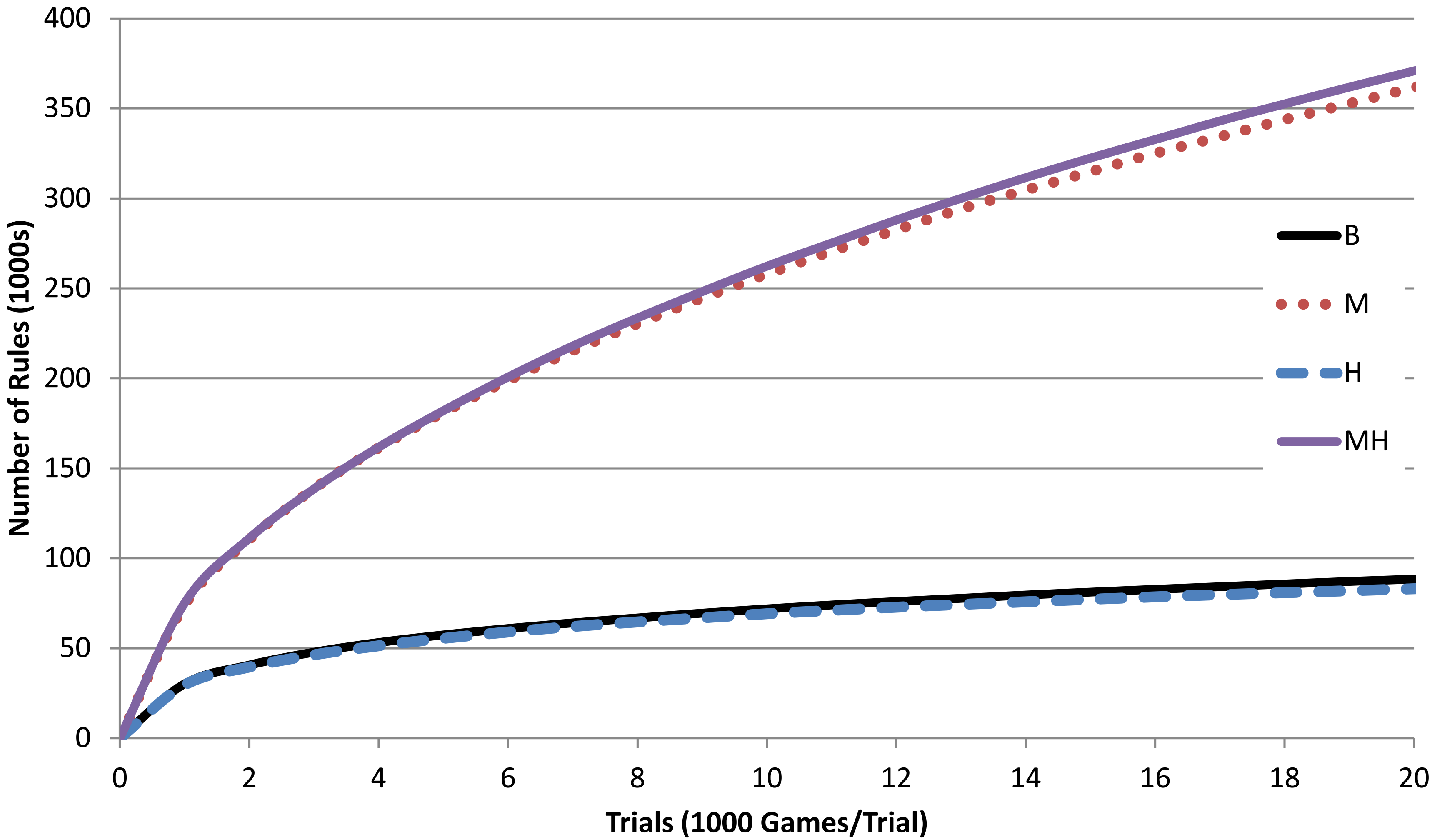


- Best agents do significantly better than hand coded.
- H and M give better initial performance than B.
- P alone speed learning (smaller state space).
- M slows learning (much larger state space).

Learning Agents with Initial Values = 0



Number of Rules Learned



Nuggets and Coal

- Nuggets:
  - First combination of chunking/EBL with RL
    - Transition from deliberate to reactive to learning
    - Potential story for origin of value functions for RL
  - Intriguing idea for creating evaluation functions for game-playing agents
    - Complex deliberation for novel and rare situations
    - Reactive RL learning for common situations
- Coal
  - Sometimes background knowledge slows learning…
  - Appears we need more than RL!
  - How recover if learn very general RL rules?