Iterative Learning of Weighted Rule Sets for Greedy Search

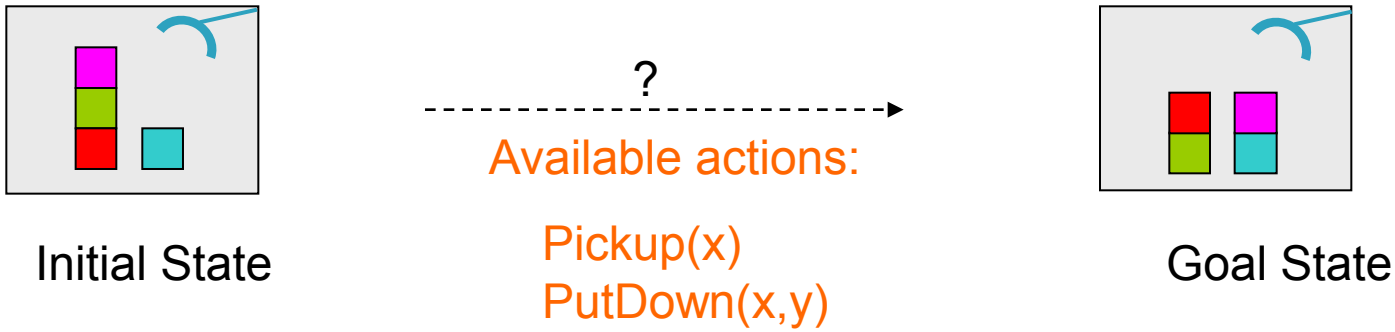
Yuehua Xu, Alan Fern, and Sungwook Yoon

05/15/2010

Deterministic STRIPS Planning

A planning problem gives:

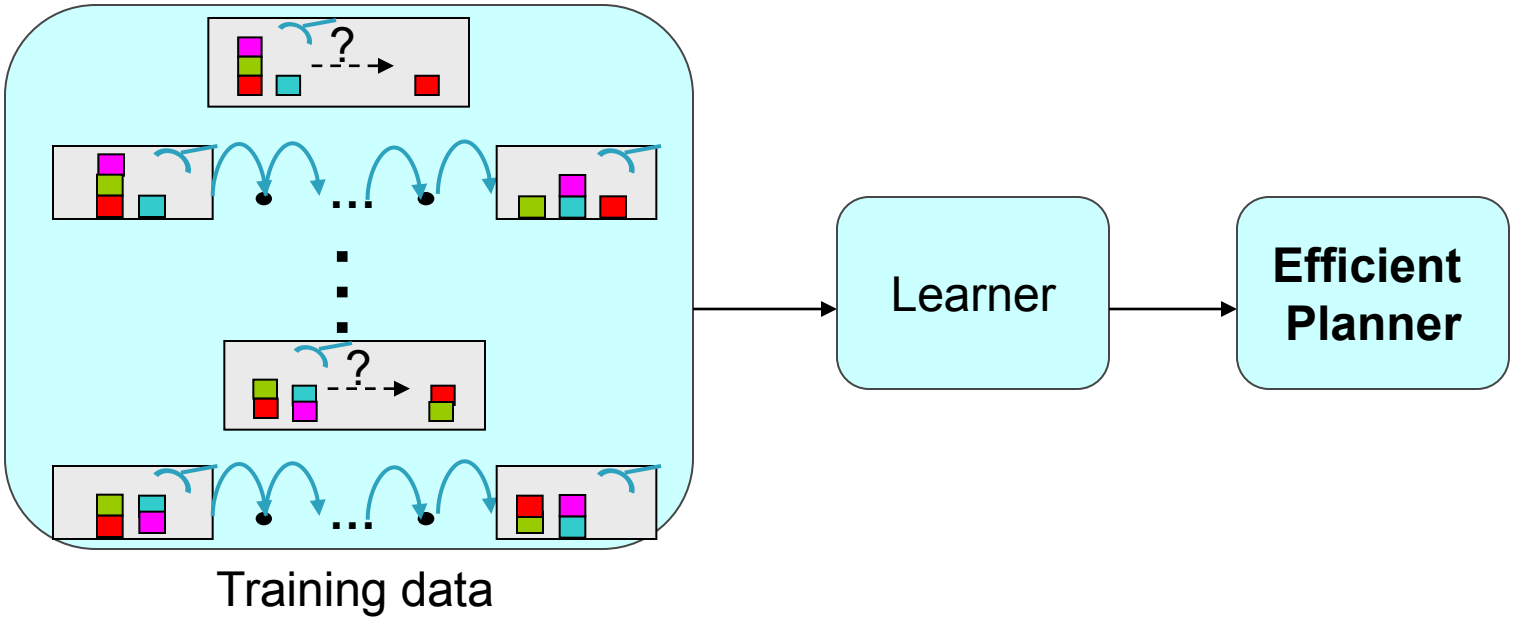
- an initial state
- a goal condition
- a list of actions and their semantics (e.g. STRIPS)



Objective: find action sequence from initial state to goal

Learning Efficient Planners

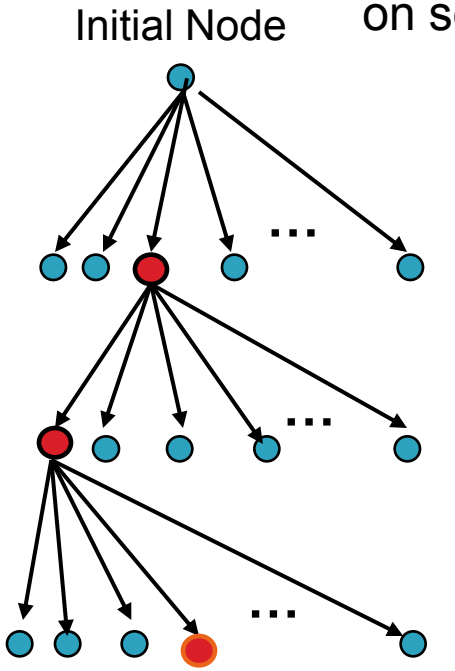
- **Our Goal:** learn efficient planners from prior planning experience in a target domain



- What type of efficient planner should we learn?

Greedy Search

Assume we have greedy control knowledge on search nodes



Select the best node according to the greedy control knowledge

Goal found!

- How to represent the control knowledge?
- How to learn the control knowledge?

Rule based Policies

- Decision-lists of action selection rules

IF *holding*(*x*) & *gontable*(*x*) THEN *PutOnTable*(*x*)
ELSE IF *clear*(*x*) & *handempty* THEN *PickUp(x)*
ELSE IF *clear*(*x*) & *holding*(*y*) & *gon** (*x,y*) THEN *PutDown*(*y,x*)

- In many domains, the learned policies are often imperfect and result in poor performance
- Here we address three reasons for this lack of robustness

Problem I

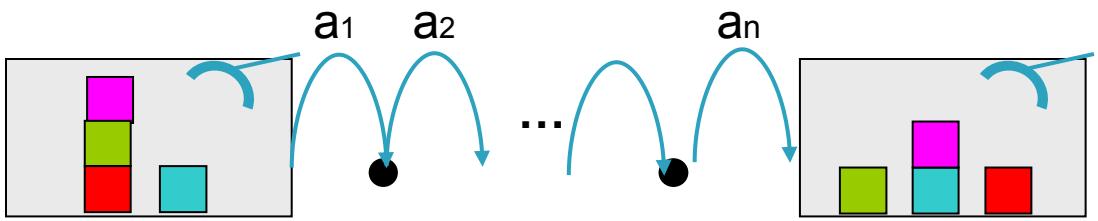
- Each decision of a policy is made by a single rule in the list

IF *holding*(*x*) & *gontable*(*x*) THEN *PutOnTable*(*x*)
ELSE IF *clear*(*x*) & *handempty* THEN *PickUp(x)*
ELSE IF *clear*(*x*) & *holding*(*y*) & *gon** (*x,y*) THEN *PutDown*(*y,x*)

- Could we use many rules and let them vote to make the decisions?
  - Machine learning has shown voting often helps

Problem II

- The learning approaches forces policies to copy the arbitrary action sequence given in the training data

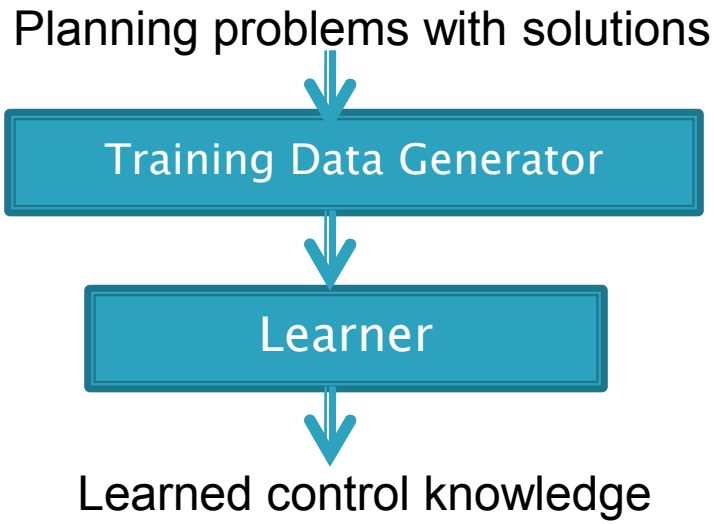


- There are often many good ordering of these actions that are not included in the training set

- Can make learning problem harder

Problem III

- The learning approach doesn't consider the actual search performance of the learned policy



- Learner never tests control knowledge

Our Work

- We are going to present our work that solves all these three problems
  - Introduce a new form of control knowledge that allows multiple rules to vote
  - Define the learning goal as of forcing greedy search to remain consistent with partially ordered plans
  - Give a learning algorithm that tightly integrates learning with search

Remainder of the Talk

- Knowledge Representation
- Ranking Problems and RankBoost
- Boosting Style Algorithm for Learning Rules and Weights
- Experimental Results

Remainder of the Talk

- Knowledge Representation
- Ranking Problems and RankBoost
- Boosting Style Algorithm for Learning Rules and Weights
- Experimental Results

Action Selection Rules

- Often action selection rules are used to define reactive policies or hard action-selection constraints
  - IF *holding*(*x*) & *gontable*(*x*) THEN *PutOnTable*(*x*)
  - IF *clear*(*x*) & *holding*(*y*) & *gon**(*x*,*y*) THEN *PutDown*(*y*,*x*)
- Previous work has shown there are learners that learn good action-selection rules
- How can we use such rules in a more robust way?

Weighted Rule Sets

- Each possible rule *R* defines a binary feature *f* on state-goal-action triples:
  - $f(s,g,a) = 1$ iff *R* "suggests" *a* in state *s* for goal *g*
  - Can be viewed as features of state transitions

- A linear combination of such features defines a ranking function on state transitions:

$$F(s,g,a) = \sum_i w_i \cdot f_i(s,g,a)$$

  - Rank of a transition is sum of weights of rules that suggest transition
  - Can use to guide greedy search

Remainder of the Talk

- Knowledge Representation
- Ranking Problems and RankBoost
- Boosting Style Algorithm for Learning Rules and Weights
- Experimental Results

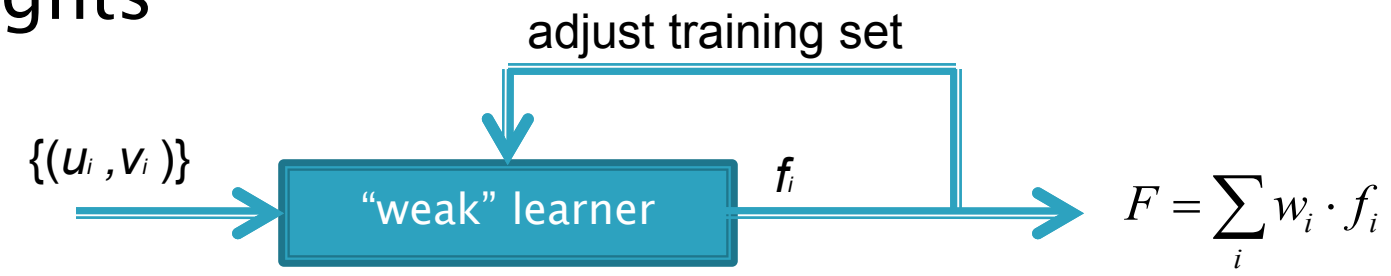
Traditional Ranking Problems

- **Input:** a set of ordered instances $\{(u_i, v_i)\}$
  - (u_i, v_i) indicates that v_i should be ranked higher than u_i
- **Output:** A ranking function F that minimizes the number of misranked pairs
- RankBoost is an algorithm for learning linear ranking functions

$$F(v) = \sum_i w_i \cdot f_i(v)$$

RankBoost

- Assume a “weak” feature learner that returns a feature with “non-trivial” performance
- Iteratively create new features and learn the weights

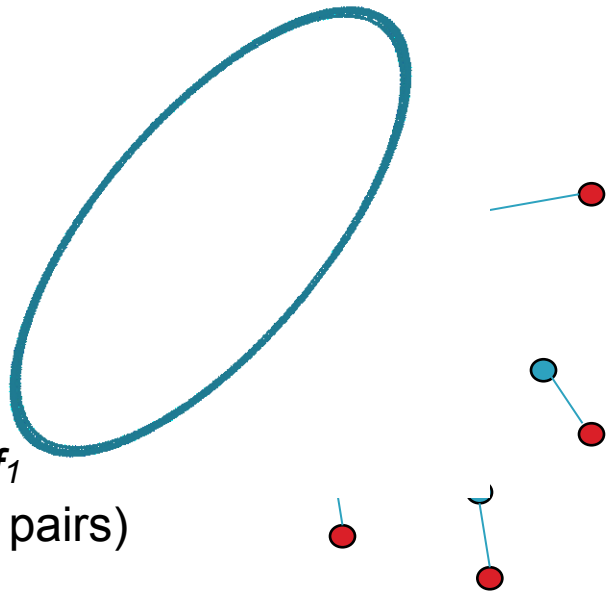


- Strong theoretical guarantees

RankBoost Example

Each ranking instance is a pair of points: v_1 v_2

Initially, all pairs have the same weight



Ranking feature f_1
(correctly ranked pairs)

RankBoost Example

Each ranking instance is a pair of points: v_1 v_2

The weight of incorrectly ranked pairs will increase and the weight of correctly ranked pairs will decrease

Ranking feature f_1

Ranking feature f_2
(emphasizes misranked pairs)

RankBoost Example

Each ranking instance is a pair of points: v_1 v_2

Finally, the learned ranking function is a linear combination of all ranking features

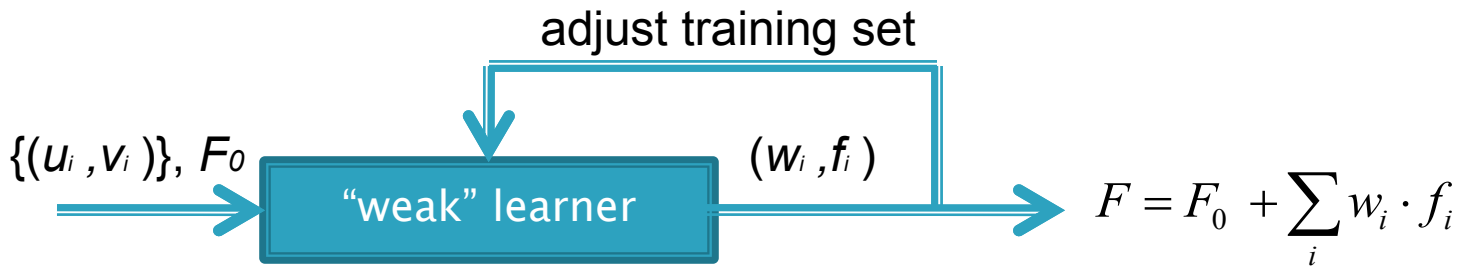
nking feature f_3

Ranking featu

Ranking feature f_2

RB-prior: RankBoost with Prior Knowledge

- We extend RankBoost to allow for prior knowledge

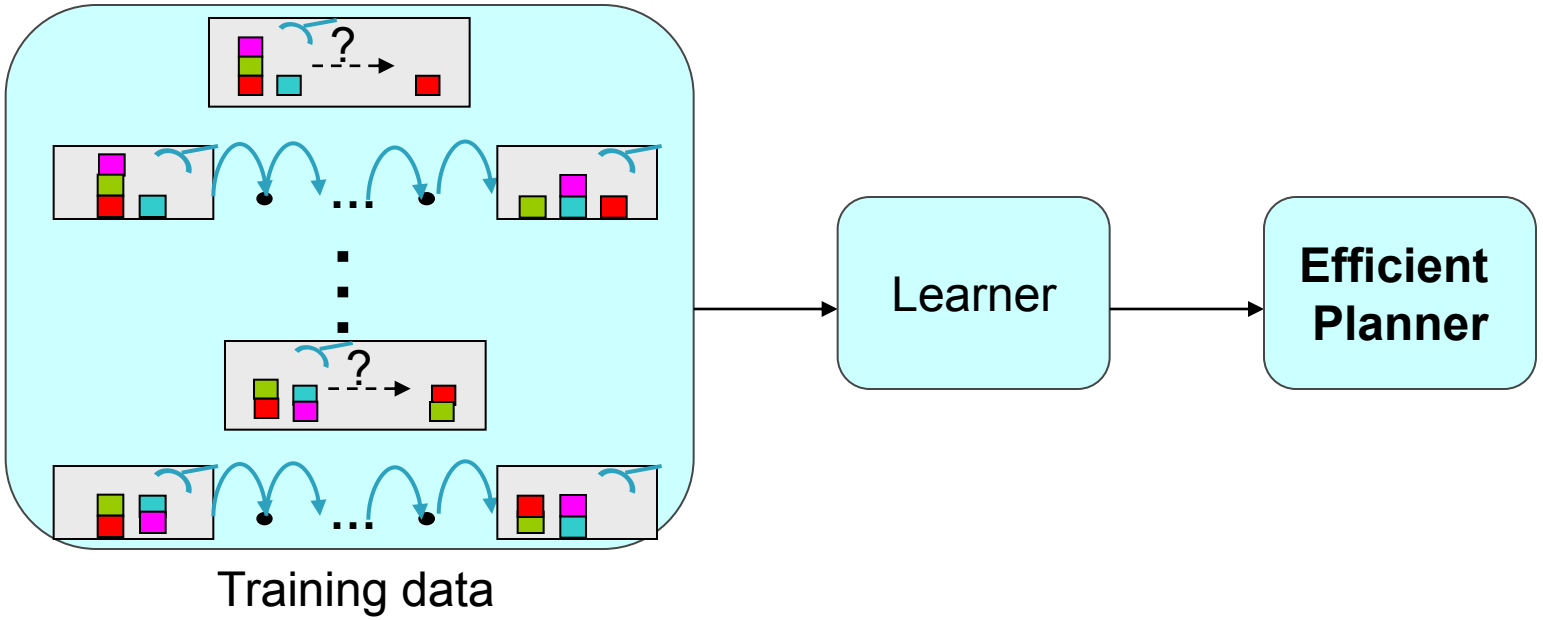


- Strong theoretical properties as RankBoost

Remainder of the Talk

- Knowledge Representation
- Ranking Problems and RankBoost
- Boosting Style Algorithm for Learning Rules and Weights
- Experimental Results

Back to Planning

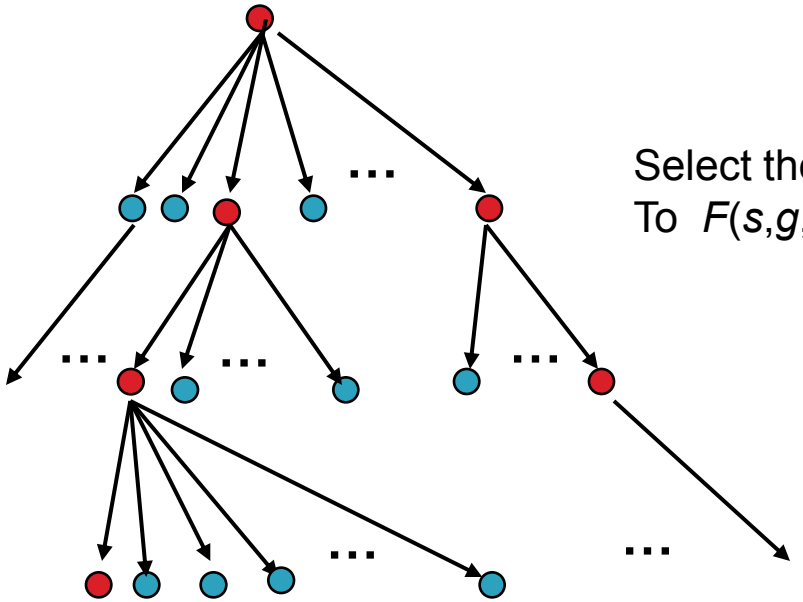


- We consider greedy search and learn linear ranking functions that can guide greedy search

$$F(s,g,a) = \sum_i w_i \cdot f_i(s,g,a)$$

- We convert the sequential plans to partially ordered plans

Learning Problem

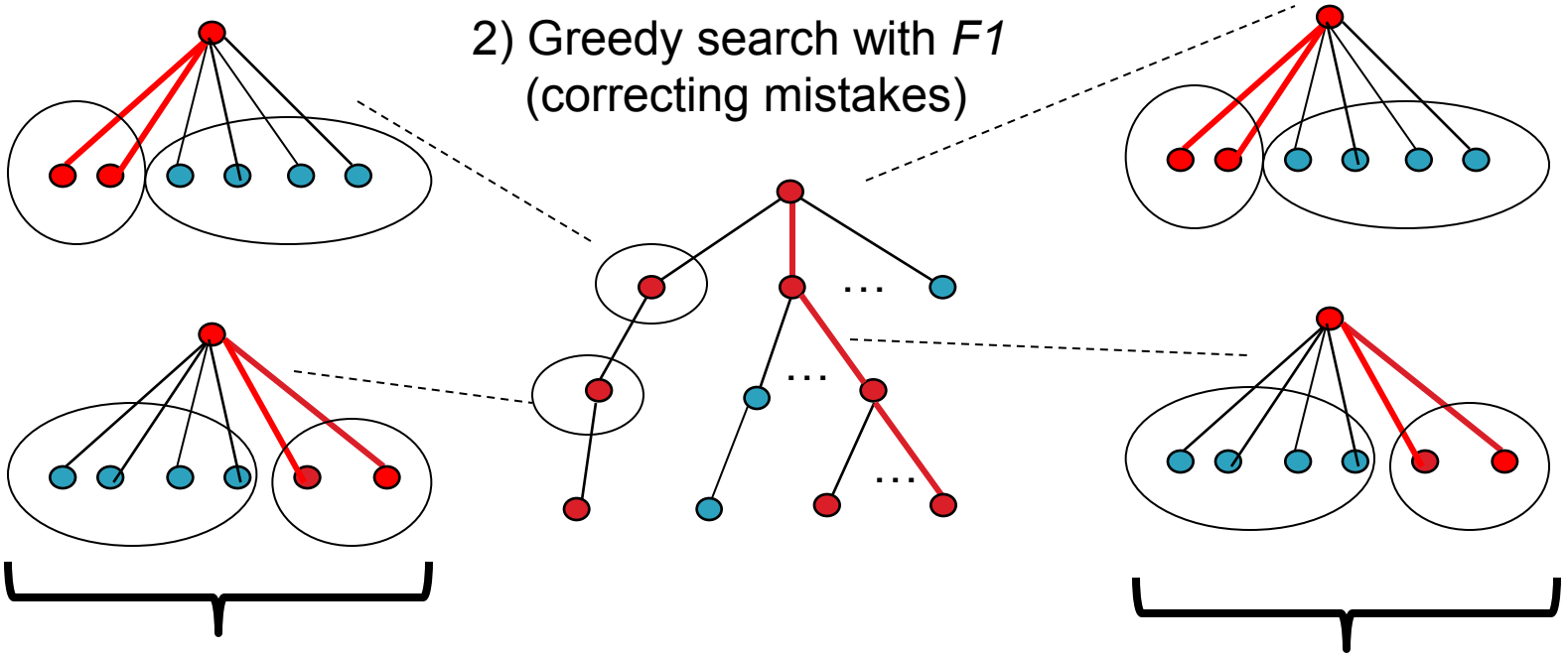


Select the best node according
To $F(s,g,a)$

- **Our Goal**: learn weighted rule set so that greedy search according to $F(s,g,a)$ remains consistent with the partially ordered plan

Iterative Learning in Search

- **Rough idea**: Iteratively formulate ranking problems according to the current search process



Convergence Properties

- Under certain assumptions, each call to RB-prior is guaranteed to terminate in a finite amount of time with zero rank loss
- The number of calls to RB-prior is bounded
- To our knowledge, first convergence result.
  - Most work doesn't even check whether it converges

Remainder of the Talk

- Knowledge Representation
- Ranking Problems and RankBoost
- Boosting Style Algorithm for Learning Rules and Weights
- Experimental Results

Experiment Setup

- Domains:
  Blocks world, Pipesworld, Pipesworld-with-tankage, Philosopher, DriverLog, Depots, FreeCell
- Evaluated greedy search using different ranking functions
- Learning approaches:
  - ITR – our iterative learning algorithm
  - RB-prior – non-iterative RB-prior
  - Xu'09 – prior work on weight learning with provided features
  - Yoon'08 – prior work on learning rule based policies
  - RPL– FF's relaxed plan length heuristic;

Experimental Results

Problem Solved (Median plan length)	RPL	Yoon'08 (Rule-based Policies)	Xu'09 (greedy)	RB-prior (non-iterative method)	ITR (iterative method)
Blocksworld	13 (3318)	N/A	27 (840)	30 (166)	**30 (118)**
Depots	1 (462)	N/A	4 (1526)	11 (129)	**23 (433)**
DriverLog	0 (–)	N/A	0 (–)	3 (2852)	**4 (544)**
FreeCell	5 (96)	N/A	7 (132)	7 (96)	**9 (92)**
Pipesworld	11 (114)	0 (–)	16 (1803)	17 (1063)	**17 (579)**
Pipesworld-with-tankage	**6 (119)**	0 (–)	5 (55)	6 (152)	5 (206)
Philosopher	0 (–)	0 (–)	6 (589)	**33 (363)**	**33 (363)**

- Weighted rule sets much more effective than learned policies
- New approaches improve on Xu'09 while learning both features and weights
- Iterative approach can improve on non-iterative approach

More Observations

- As the learning goes on, we may fail to induce new features
  - The rule learner fails to adequately explore the space of possible rules
  - Limitation of our language of representing the rules
- The planning performance sometimes got worse when more rules are induced
  - Require us to study more on the learning approaches

Future Work

- More powerful weak learning algorithms
- More powerful control knowledge
  - Policy based features – Each feature consists of a list of action selection rules
- Find improved convergence bounds
  - Our current analysis is worst-case convergence
- Understand the fundamental limitations on learning for greedy search
  - Can one characterize when it is possible to practically compile search away via learning?

Thank you!

Learning to Rank [Xu, Fern, Yoon: JMLR'09]

- **Previous Work:** learn ranking function so that beam search solves training problems
- Assumed linear ranking function over a human provided set of features:

$$H(n) = \sum_i w_i \cdot f_i(n)$$

- Gave algorithm for learning weights

- Here we focus on greedy search and extend that work in two ways
  1) Consider new type of "rule-based" feature
  2) Learn features automatically