

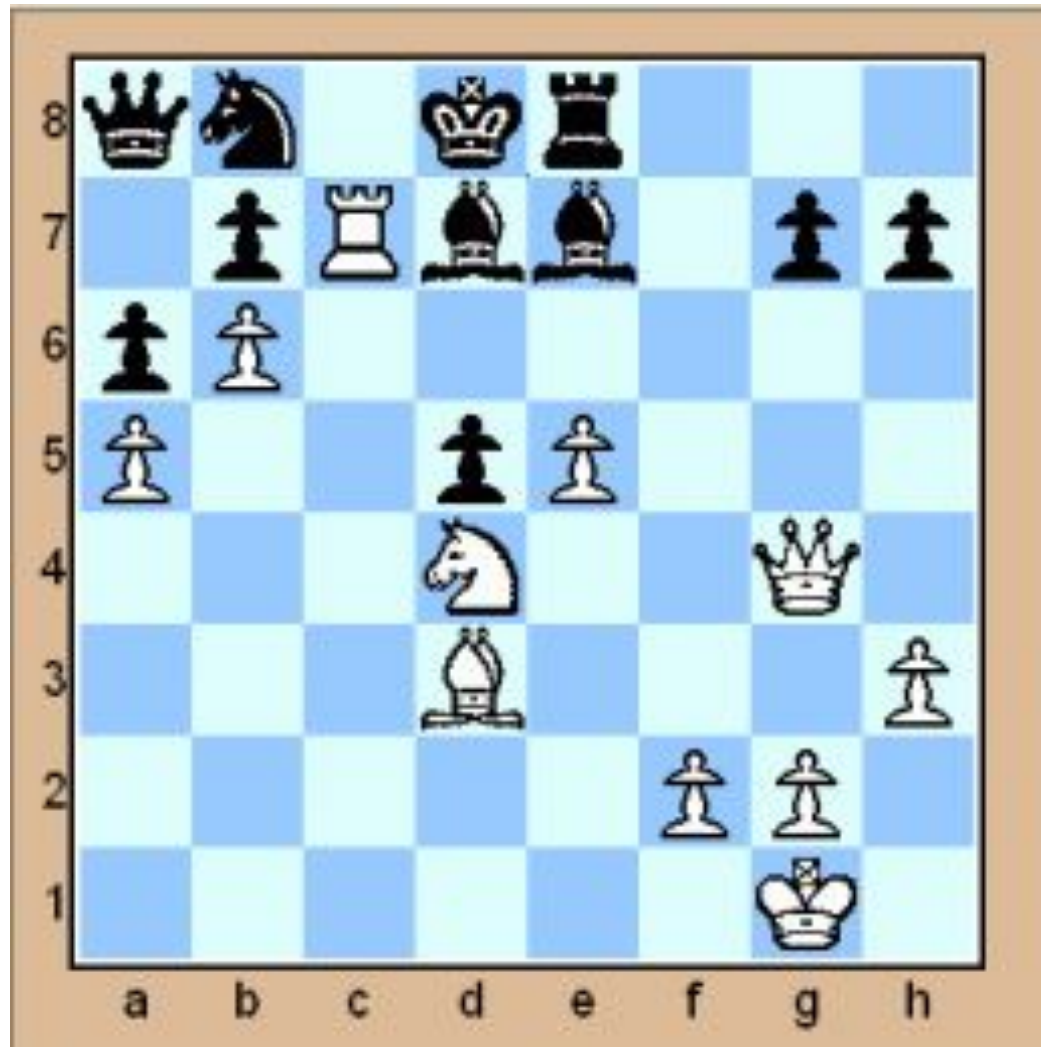
Life: play and win in 20 trillion moves

Stuart Russell

Computer Science Division, UC Berkeley

Joint work with Ron Parr, David Andre, Andy Zimdars, Carlos Guestrin, Bhaskara Marthi, and Jason Wolfe

White to play and win in 2 moves



Life

- 100 years x 365 days x 24 hrs x 3600 seconds x 640 muscles x 10/second = 20 trillion actions
- (Not to mention choosing brain activities!)
- And the world has a very large, partially observable, uncertain, unknown state space
- So, being intelligent is “provably” hard
- How on earth do we manage?

Hierarchical structure!!

(among other things)

- Deeply nested behaviors give **modularity**
 - E.g., **tongue controls for [t]** independent of everything else given **decision to say “tongue”**
- High-level decisions give **scale**
 - E.g., “go to SARA” \approx 3,000,000,000 actions
 - Look further ahead, reduce computation exponentially

[NIPS 97, ICML 99, NIPS 00, AAAI 02, ICML 03, IJCAI 05, ICAPS 07, ICAPS 08, ICAPS 10, IJCAI 11, UAI 12]

Outline

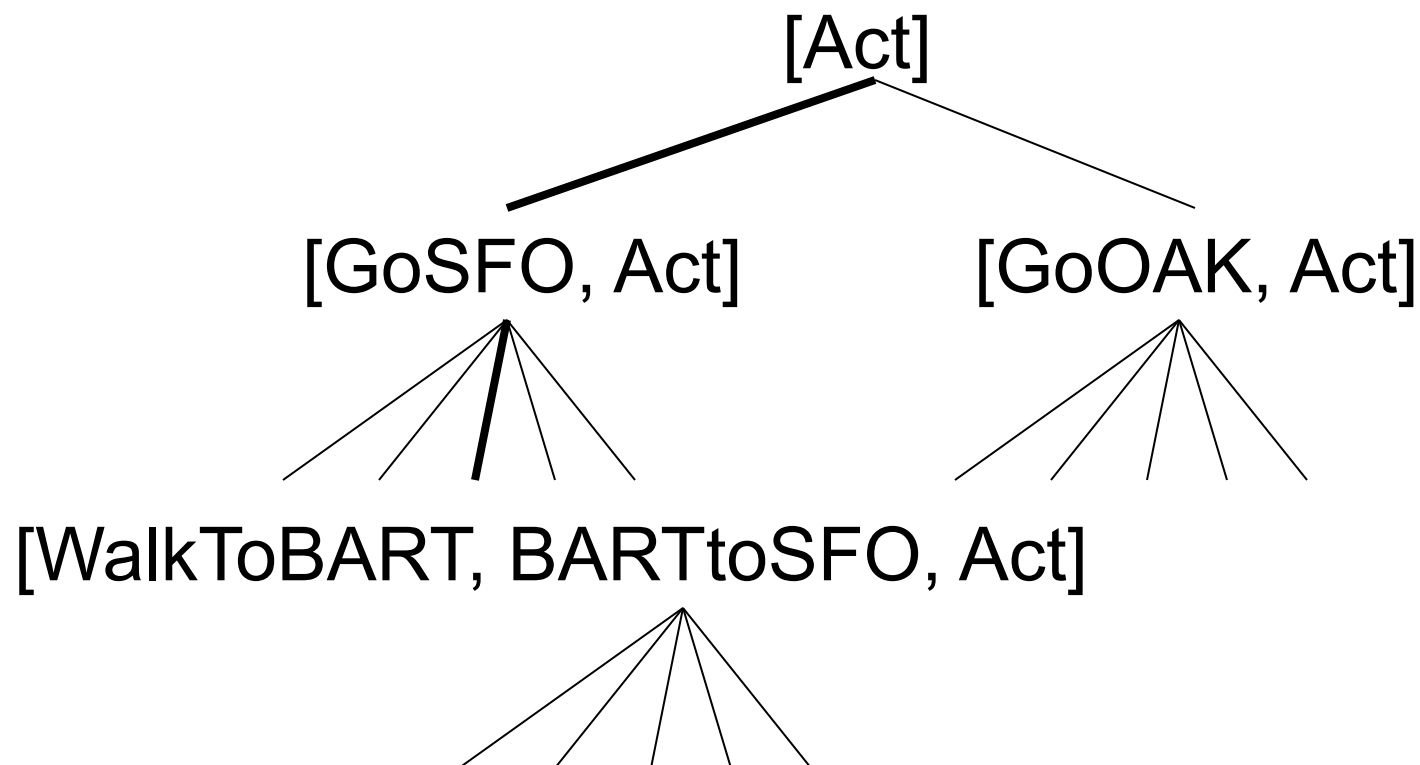
- Efficient hierarchical planning
- Hierarchical RL with partial programs
- Next steps

Outline

- Efficient hierarchical planning
- Hierarchical RL with partial programs
- Next steps

High-level actions

- Classical HTN (Hierarchical Task Network)
- A **high-level action** (HLA) has a set of possible **refinements** into **sequences** of actions, primitive or high-level
- **Hierarchical optimality** = best **primitive refinement** of **Act**



Semantics of HLAs

- To do offline or online planning with HLAs, need a **model** describing outcome of each HLA

Semantics of HLAs

- To do offline or online planning with HLAs, need a **model** describing outcome of each HLA
- Drew McDermott (AI Magazine, 2000):
 - *The semantics of hierarchical planning have never been clarified ... no one has ever figured out how to reconcile the semantics of hierarchical plans with the semantics of primitive actions*

Finding correct high-level plans

- Downward refinement property (DRP)
 - Every apparently successful high-level plan has a successful primitive refinement
- “Holy Grail” of HTN planning: allows commitment to abstract plans without backtracking

Finding correct high-level plans

- Downward refinement property (DRP)
 - Every apparently successful high-level plan has a successful primitive refinement
- “Holy Grail” of HTN planning: allows commitment to abstract plans without backtracking
- Bacchus and Yang, 1991: *It is naïve to expect DRP to hold in general*

Finding correct high-level plans

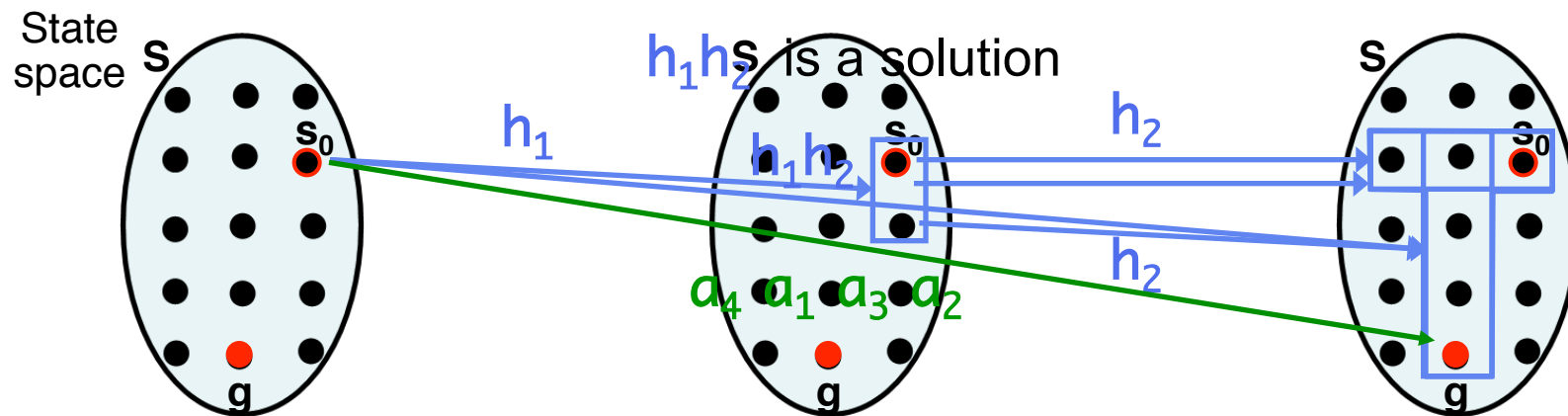
- Downward refinement property (DRP)
 - Every apparently successful high-level plan has a successful primitive refinement
- “Holy Grail” of HTN planning: allows commitment to abstract plans without backtracking
- Bacchus and Yang, 1991: *It is naïve to expect DRP to hold in general*
- MRW, 2007: Theorem: If assertions about HLAs are **true**, then DRP **always holds**

Finding correct high-level plans

- Downward refinement property (DRP)
 - Every apparently successful high-level plan has a successful primitive refinement
- “Holy Grail” of HTN planning: allows commitment to abstract plans without backtracking
- Bacchus and Yang, 1991: *It is naïve to expect DRP to hold in general*
- MRW, 2007: Theorem: If assertions about HLAs are **true**, then DRP **always holds**
- Problem: how to say true things about effects of HLAs?

Angelic semantics for HLAs

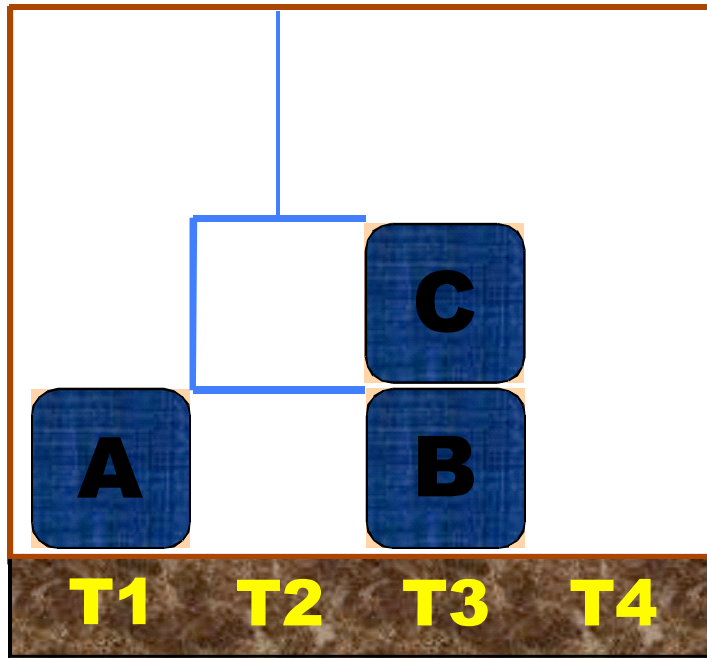
- Start with atomic state-space view, start in s_0 , goal state g
- Central idea is the **reachable set** of an HLA from each state
 - When extended to sequences of actions, allows proving that a plan can or cannot possibly reach the goal
- May seem related to nondeterminism
 - But the nondeterminism is **angelic**: the “uncertainty” will be resolved by the agent, not an adversary or nature



Technical development

- NCSTRIPS to describe reachable sets
 - STRIPS add/delete plus “possibly add”, “possibly delete”, etc
 - E.g., GoSFO adds AtSFO, *possibly adds* CarAtSFO
- Reachable sets may be too hard to describe exactly
- Upper and lower descriptions bound the exact reachable set (and its cost) above and below
 - Still support proofs of plan success/failure
 - *Possibly-successful* plans must be refined
- Sound, complete, optimal offline planning algorithms
- Complete, eventually-optimal online (real-time) search

Example – Warehouse World

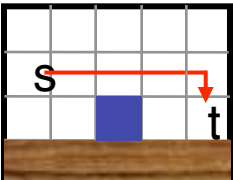
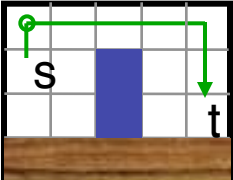
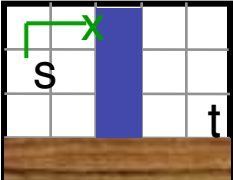


Left, Down, Pickup,
Up, Turn, Down, Putdown,
Right, Right, Down,
Pickup, Left, Put, Up,
Left, Pickup, Up, Turn,
Right, Down, Down,
Putdown

- Has similarities to blocks and taxi domains, but more choices and constraints
 - Gripper must stay in bounds
 - Can't pass through blocks
 - Can only turn around at top row
- Goal: have C on T4
 - Can't just move directly
 - Final plan has 22 steps

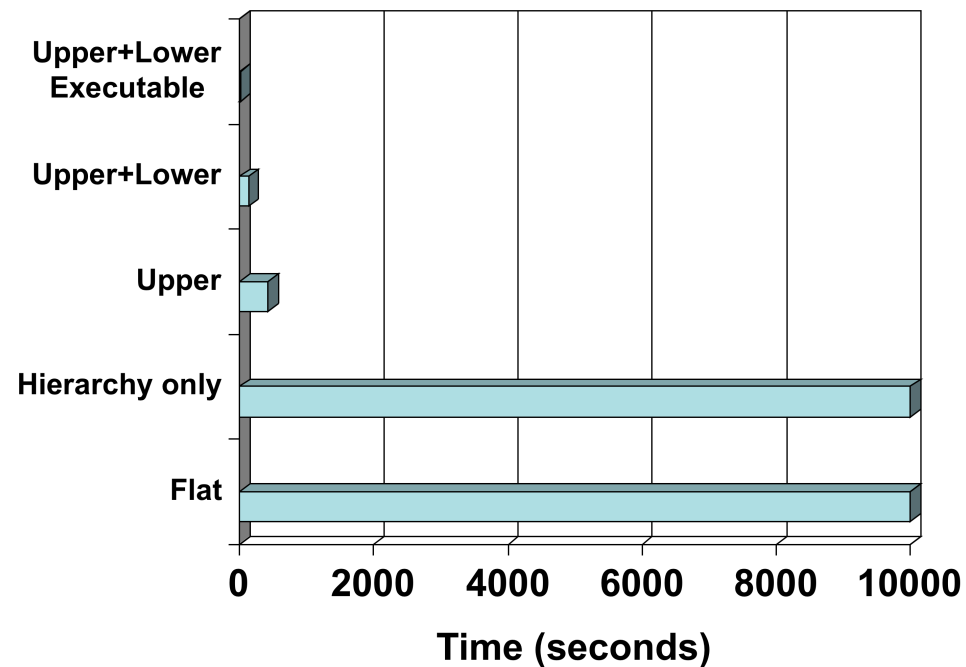
NCSTRIPS for warehouse world

- An efficient algorithm exists to **progress state sets** (represented as DNF formulae) through descriptions

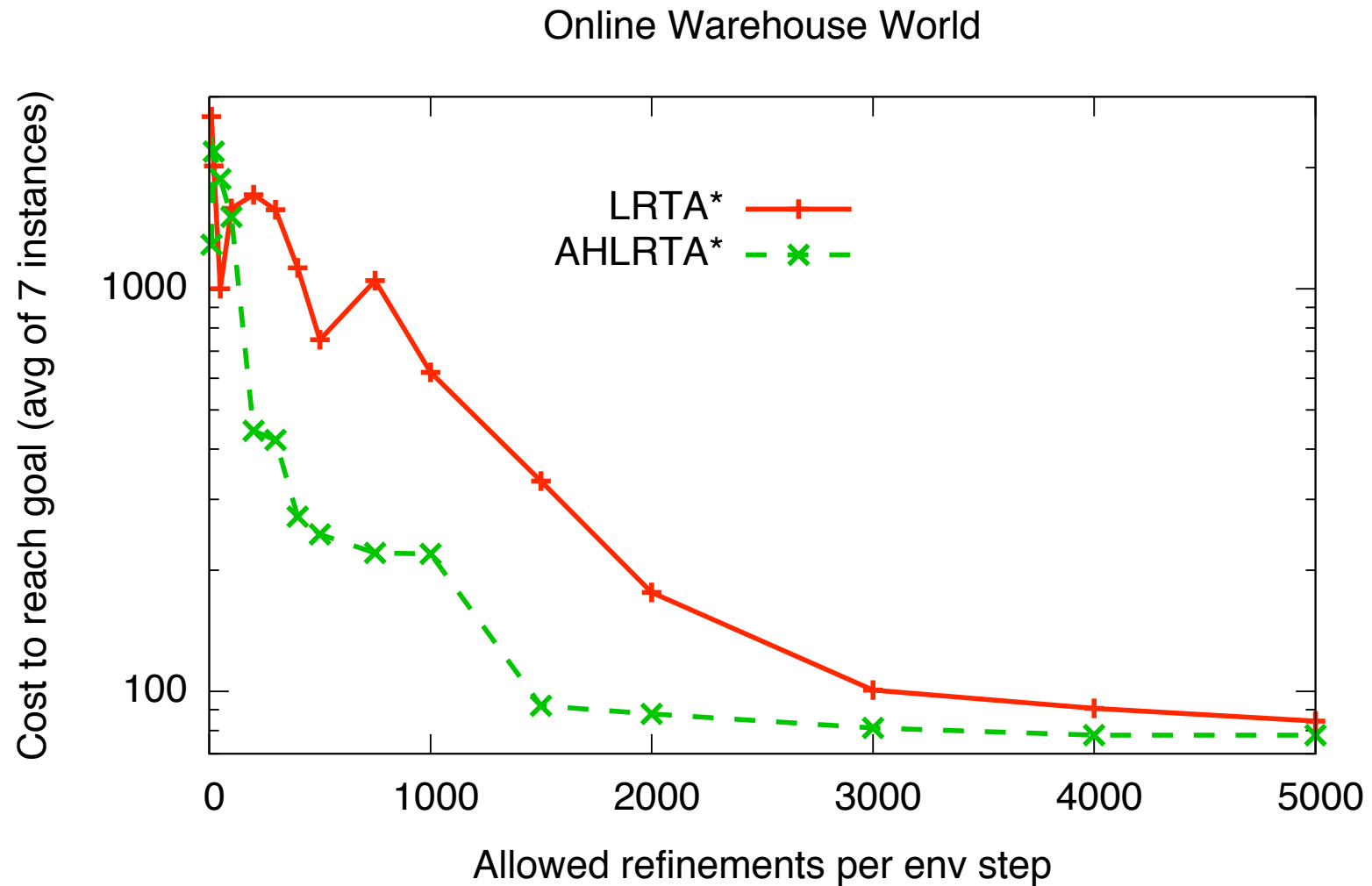
Navigate(x_t, y_t)	(Pre: $At(x_s, y_s)$)
Upper: $-At(x_s, y_s), +At(x_t, y_t), \exists FacingRight$	
Lower: IF ($Free(x_t, y_t) \wedge \forall x Free(x, y_{max})$): $-At(x_s, y_s), +At(x_t, y_t), \exists FacingRight,$	
ELSE: nil	

Experiment

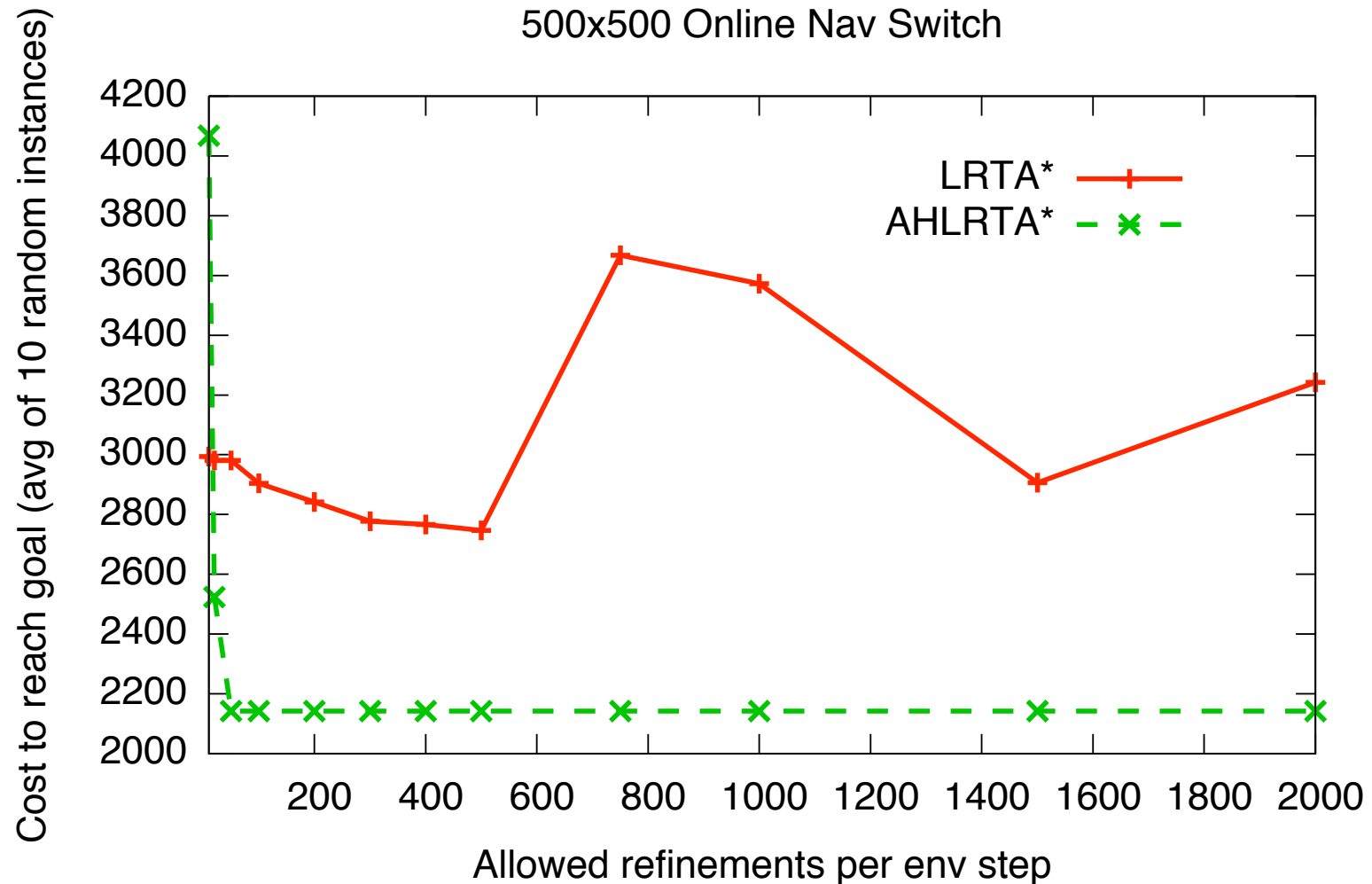
- Instance 3
 - 5x8 world
 - 90-step plan
- Flat/hierarchical without descriptions did not terminate within 10,000 seconds

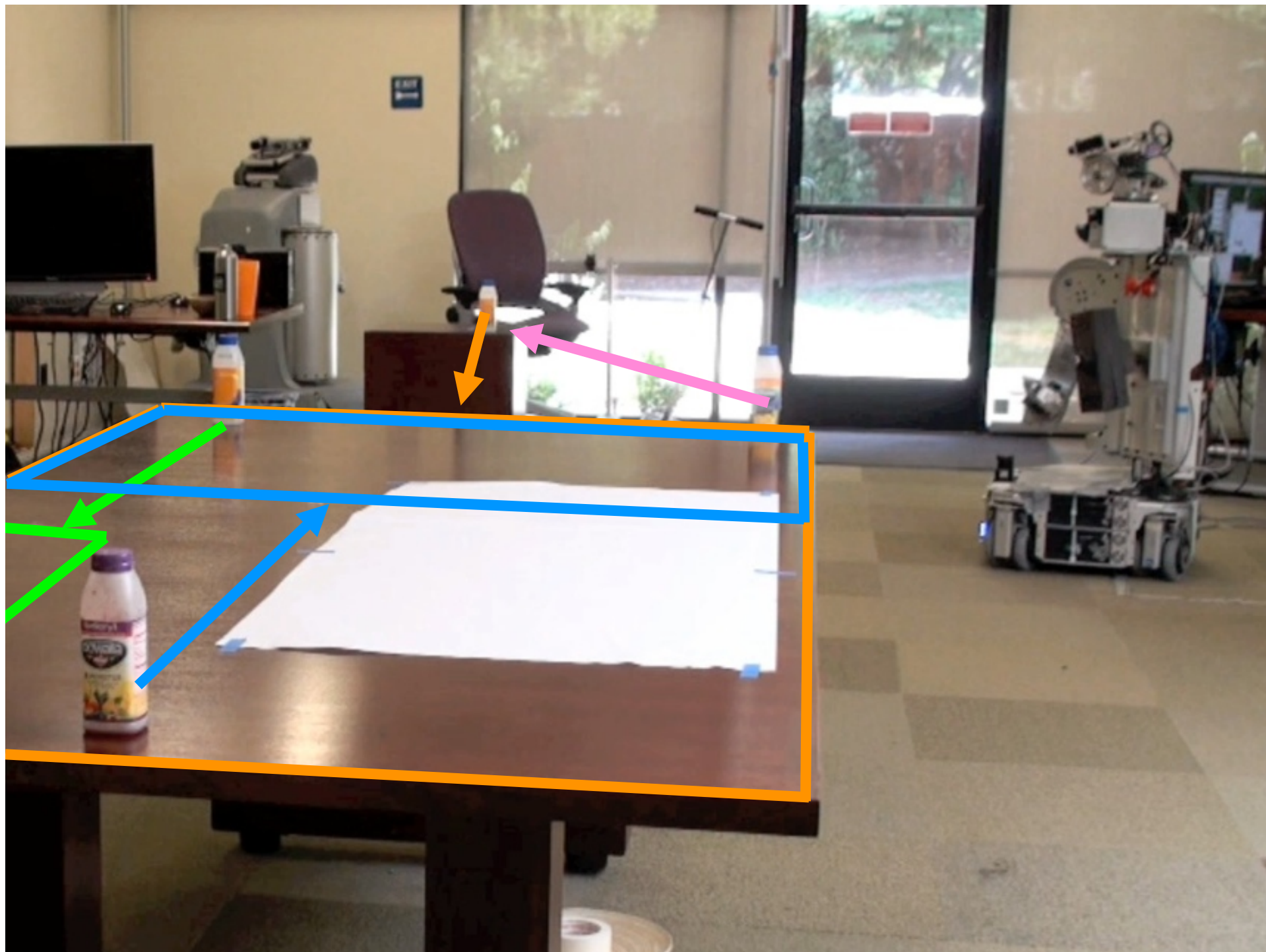


Online search: warehouse world



Online search: 500x500 nav-switch





Summary of Part 1

- Temporal abstraction is crucial for behaving well over long time scales
- Hierarchical planning was designed to take advantage of temporal abstraction
- Angelic descriptions of reachable sets ...
 - Capture the inherent *flexibility* of abstract plans
 - Support provably correct/optimal abstract planning

Outline

- Efficient hierarchical planning
- **Hierarchical RL with partial programs**
- Next steps

Temporal abstraction in RL

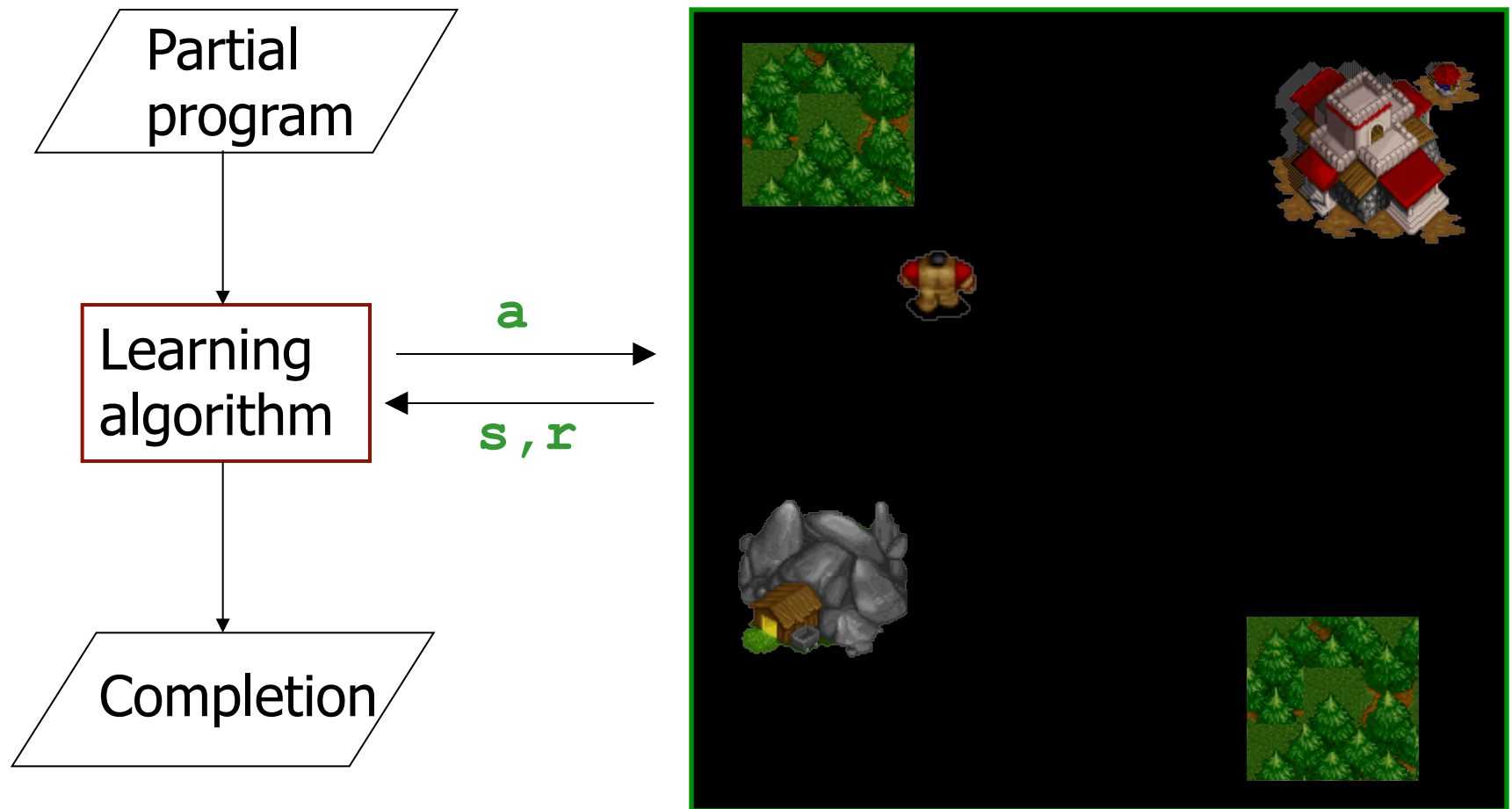
- Basic theorem (Forestier & Varaiya 78; Parr & Russell 98):
 - Given an underlying Markov decision process with primitive actions
 - Define **temporally extended choice-free actions**
 - Agent is in a **choice state** whenever an extended action terminates
 - *Choice states + extended actions form a semi-Markov decision process*
- Hierarchical structures with unspecified choices = **know-how**
 - Hierarchical Abstract Machines [Parr & Russell 98] (= recursive NDFAs)
 - Options [Sutton & Precup 98] (= extended choice-free actions)
 - MAXQ [Dietterich 98] (= HTN hierarchy)
- General **partial programs**: agent program falls in a designated restricted subset of arbitrary programs [Genesereth & Hsu, 1991]
 - Alisp [Andre & Russell 02]
 - Concurrent Alisp [Marthi et al 05]

Running example

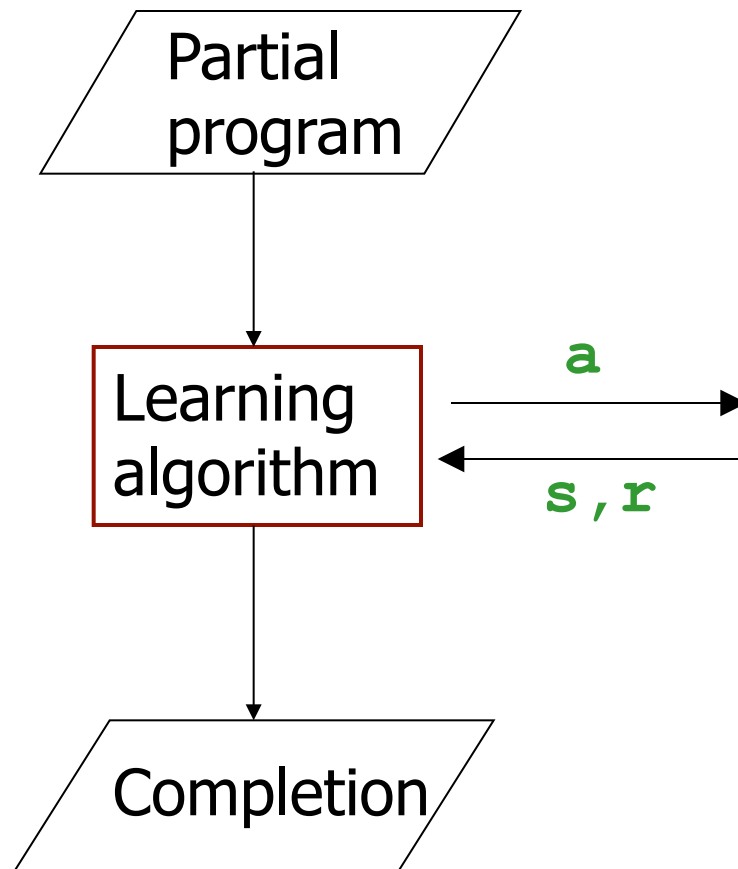
- Peasants can move, pickup and dropoff
- Penalty for collision
- Cost-of-living each step
- Reward for dropping off resources
- Goal : gather 10 gold + 10 wood
- $(3L)^n$ states **s**
- 7^n primitive actions **a**



RL and partial programs



RL and partial programs



Hierarchically optimal
for all terminating programs

Standard MDP in ALisp

```
(defun top ()  
  (loop do  
    (choose '(N S E W NoOp PickUp DropOff) ) )
```

An example Single threaded AI program

```
(defun top ()
  (loop do
    (choose (my-effectors)
      (gather-wood)
      (gather-gold)))
    (setf (first (my-effectors))
      (choose 'top-choice
        (spawn gather-wood peas)
        (spawn gather-gold peas)))))
```

```
(defun gather-wood ()
  (with-choice 'forest-choice
    (dest *forest-list*)
    (nav dest)
    (action 'get-wood)
    (nav *base-loc*)
    (action 'dropoff)))
```

```
(defun gather-gold ()
  (with-choice 'mine-choice
    (dest *goldmine-list*)
    (nav dest)
    (action 'get-gold)
    (nav *base-loc*)
    (action 'dropoff)))
```

```
(defun nav (dest)
  (until (= (my-pos) dest)
    (with-choice 'nav-choice
      (move '(N S E W NOOP))
      (action move))))
```

Technical development

Technical development

- Decisions based on *internal state*
 - Joint state $\omega = [s, \theta]$ environment state + program state
(cf. [Russell & Wefald 1989])

Technical development

- Decisions based on *internal state*
 - Joint state $\omega = [s, \theta]$ environment state + program state (cf. [Russell & Wefald 1989])
 - MDP + partial program = SMDP over $\{\omega\}$, learn $Q^\pi(\omega, u)$

Technical development

- Decisions based on *internal state*
 - Joint state $\omega = [s, \theta]$ environment state + program state (cf. [Russell & Wefald 1989])
 - MDP + partial program = SMDP over $\{\omega\}$, learn $Q^\pi(\omega, u)$
- Additive *decomposition* of value functions

Technical development

- Decisions based on *internal state*
 - Joint state $\omega = [s, \theta]$ environment state + program state (cf. [Russell & Wefald 1989])
 - MDP + partial program = SMDP over $\{\omega\}$, learn $Q^\pi(\omega, u)$
- Additive *decomposition* of value functions
 - by *subroutine structure* [Dietterich 00, Andre & Russell 02]
Q is a sum of sub-Q functions per subroutine

Technical development

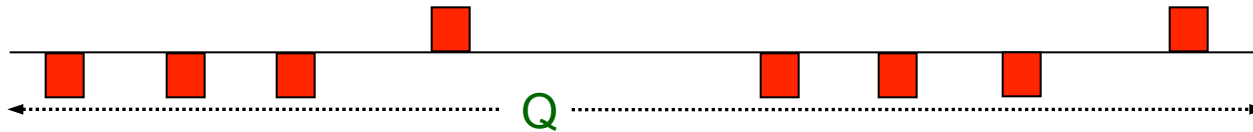
- Decisions based on *internal state*
 - Joint state $\omega = [s, \theta]$ environment state + program state (cf. [Russell & Wefald 1989])
 - MDP + partial program = SMDP over $\{\omega\}$, learn $Q^\pi(\omega, u)$
- Additive *decomposition* of value functions
 - by *subroutine structure* [Dietterich 00, Andre & Russell 02]
Q is a sum of sub-Q functions per subroutine
 - across *concurrent threads* [Russell & Zimdars 03]
Q is a sum of sub-Q functions per thread, with decomposed reward signal

Internal state

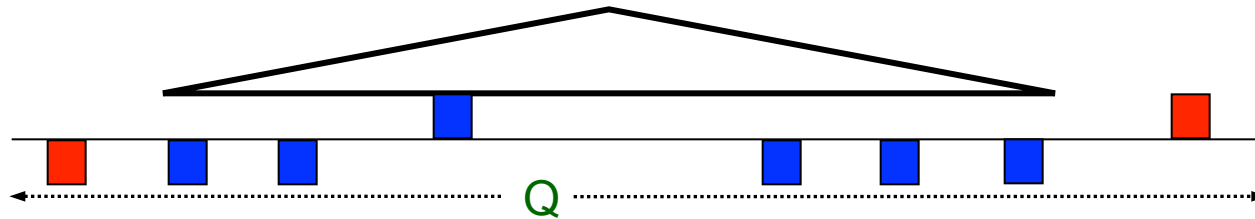
- Availability of internal state (e.g., goal stack) can greatly simplify value functions and policies
- E.g., **while navigating to (x,y)**, moving towards (x,y) is a good idea
- “**while navigating to (x,y)**” is not a state of the world; it’s purely internal!!
- Natural heuristic (distance from **destination**) impossible to express in external terms

Temporal decomposition and state abstraction

Standard Q predicts sum of rewards over all time



Temporal decomposition and state abstraction

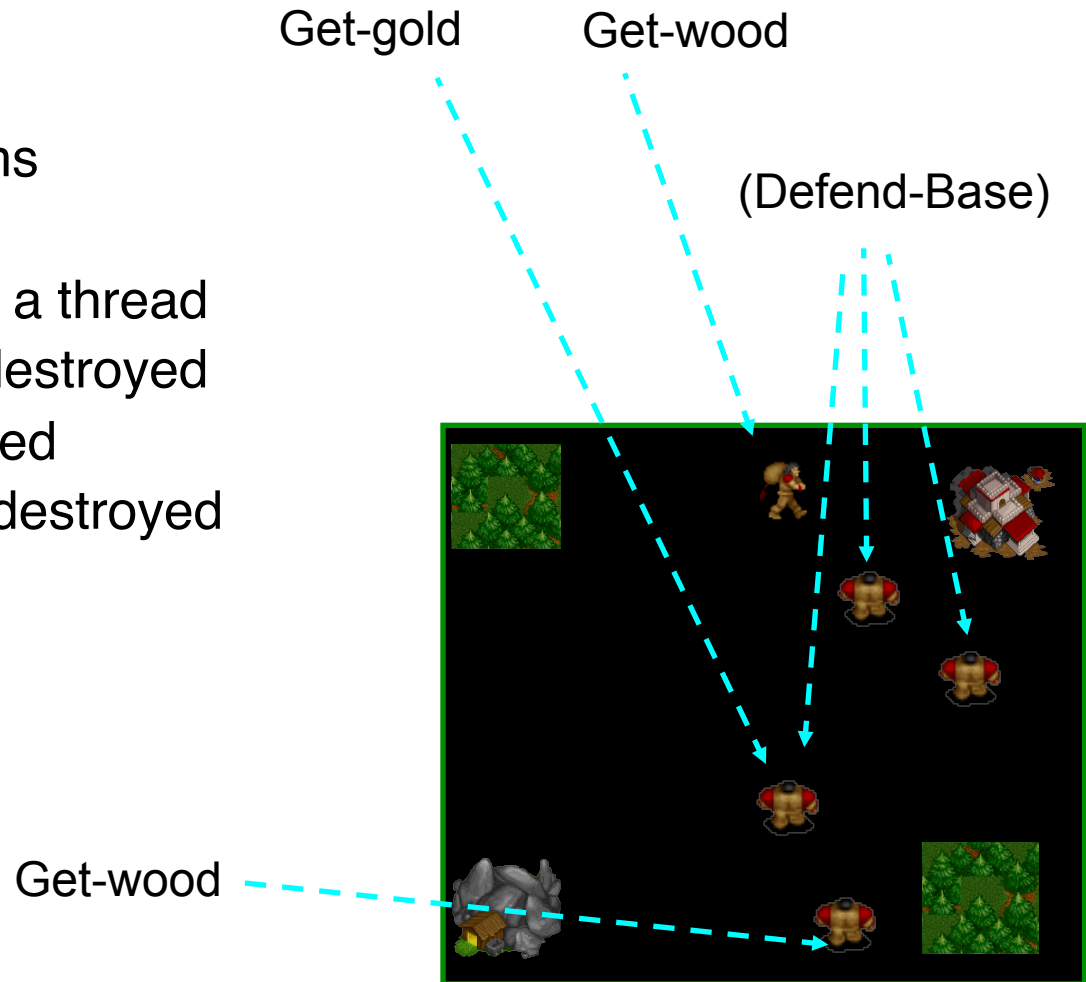


- Temporal decomposition of Q-function: local components capture sum-of-rewards *per subroutine* [Dietterich 00, Andre & Russell 02]
- => State abstraction - e.g., when navigating, local Q independent of gold reserves
- Small local Q-components => *fast learning*

Handling multiple effectors

Multithreaded agent programs

- Threads = tasks
- Each effector assigned to a thread
- Threads can be created/destroyed
- Effectors can be reassigned
- Effectors can be created/destroyed



An example Single threaded AI is a program

```
(defun top ()  
  (loop do  
    (choose (my-choices)  
      ((gather-gold))  
      ((gather-wood)))  
    (first (my-effectors))  
    (choose 'top-choice  
      (spawn gather-wood peas)  
      (spawn gather-gold peas))))
```

```
(defun gather-wood ()  
  (with-choice 'forest-choice  
    (dest *forest-list*)  
    (nav dest)  
    (action 'get-wood)  
    (nav *base-loc*)  
    (action 'dropoff)))
```

```
(defun gather-gold ()  
  (with-choice 'mine-choice  
    (dest *goldmine-list*)  
    (nav dest)  
    (action 'get-gold)  
    (nav *base-loc*)  
    (action 'dropoff)))
```

```
(defun nav (dest)  
  (until (= (my-pos) dest)  
    (with-choice 'nav-choice  
      (move '(N S E W NOOP))  
      (action move))))
```

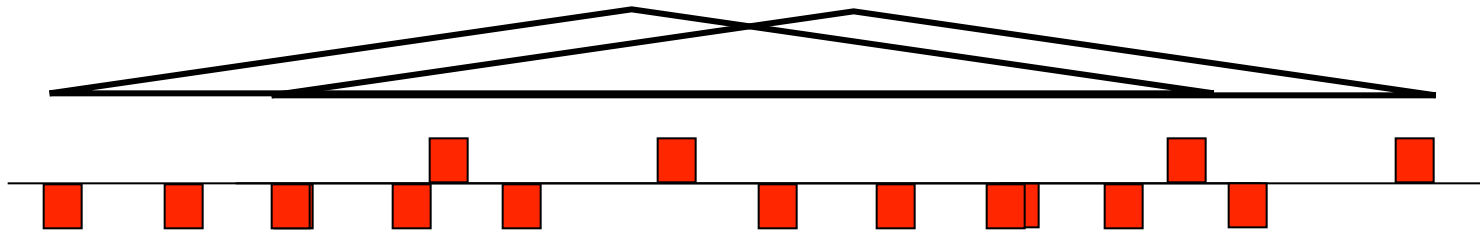

Q-functions

ω	u	$Q(\omega, u)$
...
Peas1 at NavChoice, Peas2 at DropoffGold, Peas3 at ForestChoice, Pos1=(2,3), Pos3=(7,4), Gold=12, Wood=14	(Peas1:East, Peas3:Forest2)	15.7
...

Example Q-function

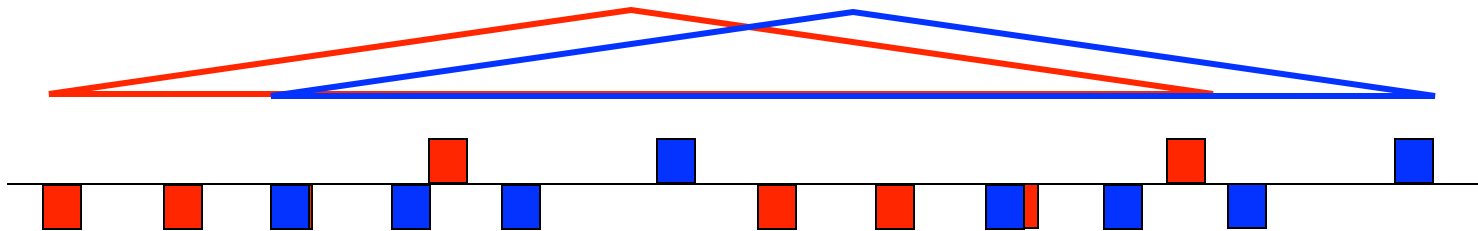
- To complete partial program, at each choice state ω , need to specify choices for all choosing threads
- So $Q(\omega, u)$ as before, except u is a **joint** choice
- Suitable SMDP Q-learning gives optimal completion

Q-decomposition w/ concurrency?



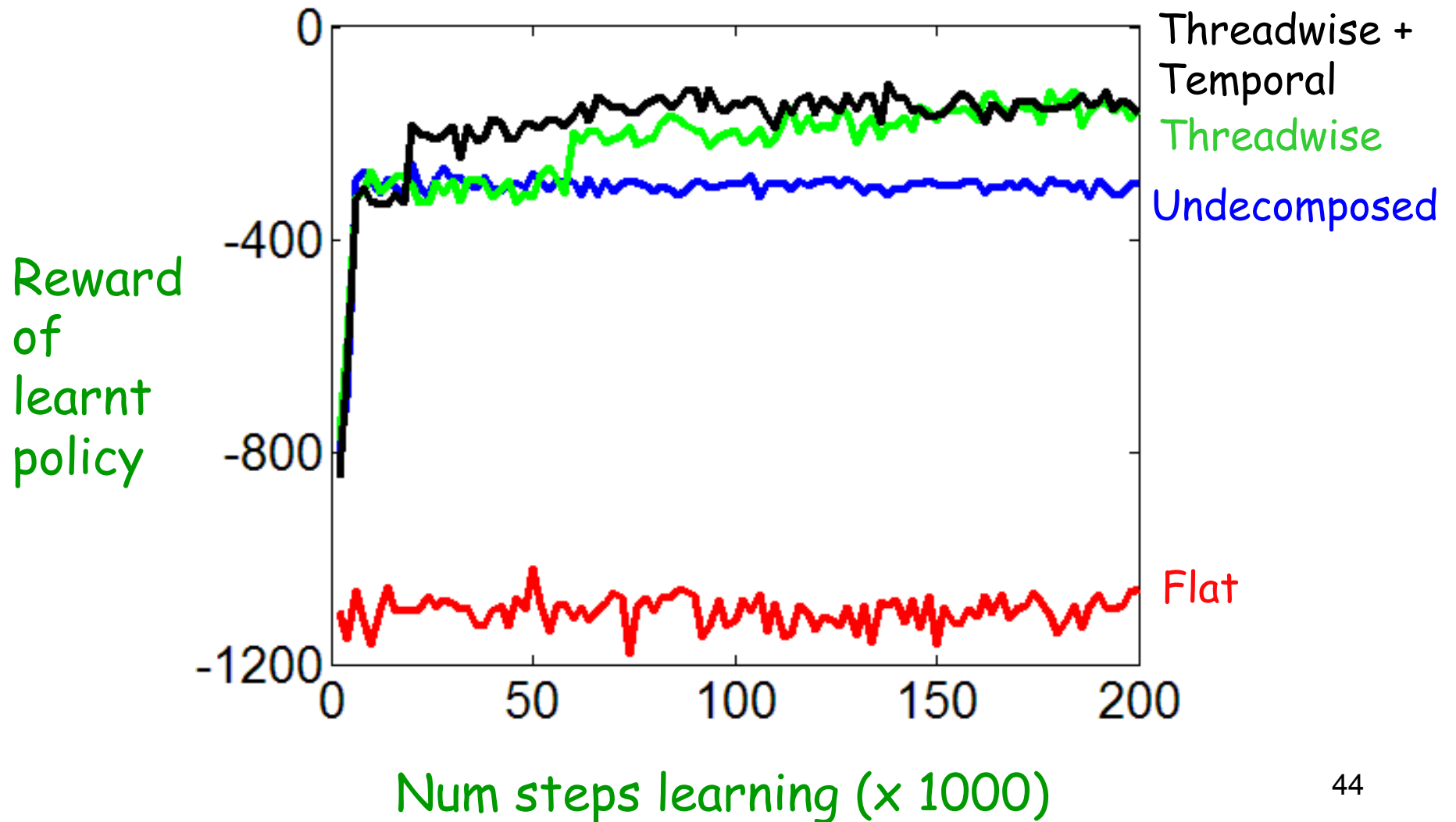
- Temporal decomposition of Q-function lost!!
- No credit assignment among threads
 - Peasant 1 brings back gold, Peasant 2 twiddles thumbs
 - Peasant 2 thinks he's done very well!!
 - => learning is hopelessly slow with many peasants

Threadwise decomposition



- Idea : decompose reward among threads [Russell & Zimdars 03]
- E.g., rewards for thread j only when peasant j drops off resources or collides with other peasants
- $Q_j^\pi(\omega, u)$ = “Expected total reward received by thread j if we make joint choice u and then do π ”
- Threadwise Q-decomposition $Q = Q_1 + \dots + Q_n$
- Recursively distributed SARSA => global optimality

Resource gathering with 15 peasants



Stratagus

Menu (F10)

100

1,000

1,000

1/21

0

Stratagus

Cycle: 550 150

0 Person 0

1 Computer 0

2 Computer 0



Summary of Part 2

- Structure in behavior seems essential for scaling up
- Partial programs
 - Provide natural structural constraints on policies
 - Decompose value functions into simple components
 - Include internal state (e.g., “goals”) that further simplifies value functions, shaping rewards
- Concurrency
 - Simplifies description of multieffector behavior
 - Messes up temporal decomposition and credit assignment (but threadwise reward decomposition restores it)

Outline

- Efficient hierarchical planning
- Hierarchical RL with partial programs
- **Next steps**

Extending angelic semantics

- For most real “hierarchically interesting” domains, need more **expressive power**
 - nondeterminism / probabilistic uncertainty / conditioning
 - partial observability
 - partial ordering / concurrency

Learning high-level behavioral structures

- Where do the HLAs and Alisp programs come from?
 - Many researchers have been looking for tabula rasa solutions to this problem
 - Cumulative learning of partial knowledge structures
 - Even very weak theories support derivations of very loose abstract plans and descriptions [Ryan, 2004]
 - Unifying know-how and know-what
- Basal ganglia!!!

Controlling expensive deliberation

- Rational metareasoning:
 - View computations as actions; agent chooses the ones providing maximal utility
 - \approx expected decision improvement minus time cost
 - Myopic metalevel control somewhat effective for search and games [Russell and Wefald, 1989, MoGo 2009]
 - NB *selection theory* not *bandit theory*!
 - *Can metalevel RL provide effective non-myopic control for deliberation over long time scales?*

Implementing metalevel RL

What if the agent program is allowed to deliberate?

E.g., an agent that does tree search to choose actions:

```
(defun top ()  
  (let ((nodes (list (make-node (get-state)))))  
    (loop until (choose '(nil t))  
      ...  
      (setq leaf-to-expand (choose nodes))  
      ...
```

Tricky issues

- Program state θ is a tree of arbitrary size
 - Need recursively defined Q-function?
- Very long computation sequences occur before each external reward
 - Provide metalevel shaping rewards for “progress”
 - Make the deliberation hierarchical!
 - High-level, long-time-scale decisions have a big and reasonably predictable effect on expected utility
 - Planning needs to be detailed for the near future, can remain abstract for the long term

Mar. 29, 1976

THE NEW YORKER

Price 75 cents

