Planning

Gal A. Kaminka

The Intelligent Random Walker

Reminder: The basic agent algorithm

```
W is knowledge base
g is an unsatisfied goal in W
B set of actions available
P set of percepts available

while g not satisfied:
   PERCEIVE() using percepts in P to update W
   CHOOSE() action b (from B) that advances towards g
   EXECUTE() action b
```

- Knowledge base W starts with initial knowledge

The (really stupid) Random Walker

```
W knowledge base, g goal, B actions, P percepts

while g not satisfied:
   PERCEIVE() using percepts in P to update W
   CHOOSE() next action b (from B) RANDOMLY
   EXECUTE() action b
```

- Guaranteed to (*slowly*) reach goal if actions are reversible
- CHOOSE() randomly from **all** actions
  - even those that are impossible

Not very goal-oriented

The (slightly less stupid) Random Walker

```
W knowledge base, g goal, B actions, P percepts

while g not satisfied:
   PERCEIVE() using percepts in P to update W

   Let C be the empty set
   For all actions b in B:
     if APPLICABLE(b, W) add b to C
   CHOOSE() next action c from C RANDOMLY
   EXECUTE() action c
```

- Guaranteed to reach goal if actions are reversible
- CHOOSE() randomly from **possible** actions

Requires **Action Model** knowledge (when **applicable**)

The (slightly less stupid) Random Walker

```
W knowledge base, g goal, B actions, P percepts
while g not satisfied:
   PERCEIVE() using percepts in P to update W
   Let C be the empty set
   For all actions b in B:
     if APPLICABLE(b, W) add b to C
   CHOOSE() next action c from C RANDOMLY
   EXECUTE() action c
```

- Guaranteed to reach goal if actions are reversible
- CHOOSE() randomly from **possible** actions

Requires **Action Model** knowledge (when **applicable**)

But, actions may cause unnecessary loops

The (less stupid) Random Walker

```
W knowledge base, g goal, B actions, P percepts

while g not satisfied:
   s = PERCEIVE() // new state
   REMEMBER!(s, W)
   Let C be the empty set
   For all actions b in B:
     if APPLICABLE(b, W) add b to C
   For all actions c in C:
     if REMEMBER?(EFFECTS(c), W):
        remove c from C, add c to C*
     if C is not empty, CHOOSE() next action c from C
     else CHOOSE() next action c from C*
    EXECUTE() action c
```

The (less stupid) Random Walker

- Action model expanded: (predicting **action effects**)
- CHOOSE() actions with non-uniform probability
- Return to previous state only if nothing else to do
  - "avoid the past" heuristic[1]
- **Remember past states**

[1] {Balch and Arkin, Avoiding the Past: A Simple but Effective Strategy for Reactive Navigation, ICRA 1993.}

Knowledge vs intelligence

- Intelligence and knowledge complement each other
- Allow agent to work correctly at **the knowledge level**
- Random walker versions:
  - No knowledge, reversible actions: goal will be reached
  - Faster if knows effects of actions
  - Faster if can also remember past states and retrieve
- **Episodic memory** (what have I seen?)
  - REMEMBER? (retrieval) and REMEMBER! (storage)

Episodic Memory

Open questions: REMEMBER!()

- What to remember
- When to remember
- In humans, lots of research on cognitive biases

Open questions: REMEMBER?()

- Associative memory (e.g., spreading activation)

Research areas: Analogy, case-based reasoning

Representing Knowledge for Planning

Knowledge Representation

- Whole area of AI devoted to knowledge representation
- Commonly known as **KR**
- Own conferences
  - KRR, knowledge representation and reasoning
- Lots of thought on how to reason about knowledge
  - Look up *ontology*, *description logics*, etc.

Simple KR used (*here* and *now*)

- Keep track of states: *beliefs* (previous lecture)
- We saw need to represent **action models**:
  - APPLICABLE?(action)
  - EFFECTS(action)
- **Domain**: states, and actions allowed in them
  - Actions: transitions between states
  - State: a combination of grounded fluent literals
    - **factored state representation**

Actions

- Take agent from one state to another
- Specified using action models
  - How state will change
- Approach by STRIPS planner (1970s) still the basis today
- Action model:
  - APPLICABLE (precondition): partial state where action can be **applied**
  - EFFECTS: changes dictated by transition
    - Delete list (fluent literals not in target)
    - Add list (fluent literals in target)

STRIPS Actions (formally)

- Action a has three associated fluent (partial) sets
  - PRE_a: preconditions
  - DEL_a: delete effects
  - ADD_a: add effects
- Given state s, new state $ss \leftarrow \delta(s, a)$
  - if $s \cap PRE_a = PRE_a$ then $\delta(s, a) = (s/DEL_a) \cup ADD_a$
  - otherwise not defined (action not applicable)

[2] {Ghallab, Nau, & Traverso. Automated Planning and Acting. Cambridge University Press, 2016.}

STRIPS Actions (formally)

- Action a has three associated fluent (partial) sets
  - PRE_a: preconditions
  - DEL_a: delete effects
  - ADD_a: add effects
- Given state s, new state $ss \leftarrow \delta(s, a)$
  - if $s \cap PRE_a = PRE_a$ then $\delta(s, a) = (s/DEL_a) \cup ADD_a$
  - otherwise not defined (action not applicable)
- There are many extensions to this in planning literature[2]
  - Continuous effects, conditional effects, sensing actions
  - Action durations, costs, existential qualifiers $(\forall, \exists)$

[2] {Ghallab, Nau, & Traverso. Automated Planning and Acting. Cambridge University Press, 2016.}

PDDL (Planning Domain Description Language)

- Language used in AI as abstraction to describe domains
- PDDL 1.0: Boolean-valued fluents, mostly STRIPS
  - In later versions, numeric valued fluents
  - Extensions support contingency and probabilistic planning, etc.
- Used extensively by planning community
- Agent perceives "PDDL fluents"
- Used in the exercises in this course
  - Several variants used today in AI planning

PDDL Domain Example (from slides by Manuela Veloso)

```
(define (domain gripper-strips)
  (:predicates  (room ?r)
                (ball ?b)
                (gripper ?g)
                (at-robby ?r)
                (at ?b ?r)
                (free ?g)
                (carry ?o ?g))
  (:action ..... ))
```

Note declaration of predicates (boolean fluents)

PDDL Action Example (from slides by Manuela Veloso)

```
(:action move
   :parameters (?from ?to)
   :precondition (and (room ?from)
                      (room ?to)
                      (at-robby ?from))
   :effect (and (at-robby ?to)
                (not (at-robby ?from))))
```

In depth:

Dana Nau's presentation about planning language representations

An Agent that can Plan

Random Walking is Hardly Enough

- The “intelligent” random walker is not very intelligent
- Some general capabilities make it better
  - Episodic Memory
  - Prediction of action effects (action models)
- Ultimately, still needs to choose

The Plan-Dispatch Agent

- Observation: if agent has action models, can consider k steps ahead

  "If I apply b_1 I will be in state s_1, where I could apply b_2 to reach s_2, . . . until I apply b_k and reach the goal state g"

The Plan-Dispatch Agent

- Observation: if agent has action models, can consider k steps ahead

  "If I apply b_1 I will be in state s_1, where I could apply b_2 to reach s_2, ... until I apply b_k and reach the goal state g"

- Planning: finding an ordered set of actions to reach a goal
- PLANNER(i, g, B):
  - i: initial state (known in W)
  - g: one or more possible goal states (known in W)
  - B: set of possible actions
  - Returns: p (a plan)

The Simplest Plan Dispatch Agent

```
W knowledge base, g goal, B actions, P percepts

s = PERCEIVE() // initial state
p = PLANNER(s,g,B) // (b1,b2, ... )
while g not satisfied:
    let b = next action in p
    EXECUTE() action c
    s = PERCEIVE() // new state
```

The Simplest Plan Dispatch Agent

```
W knowledge base, g goal, B actions, P percepts

s = PERCEIVE() // initial state
p = PLANNER(s,g,B) // (b1,b2, ... )
while g not satisfied:
    let b = next action in p
    EXECUTE() action c
    s = PERCEIVE() // new state
```

- Obviously bad: why perceive?
- Ignores changes in environment, action failures
- **May not reach goal!**

A Replanning Plan Dispatch Agent

```
W knowledge base, g goal, B actions, P percepts

s = PERCEIVE() // new state
while g not satisfied:
    p = PLANNER(s,g,B) // (b1,b2, ... )
    let b = first action in p
    EXECUTE() action b
    s = PERCEIVE() // new state
```

- Better: Now guaranteed to reach goal
- But always replans, even if not necessary

When to (re)Plan?

- When there is a choice
- When in an unexpected state
- These need to be checked in execution!

Some Basic Terms

- Plan: ordered set of actions
  - Sequence: Complete order
  - Partial-order: Alternatives, parallelization
- Policy: Plan from every possible initial state
  - Also called *Universal Plan*
- In principle: shortest (optimal) path problem
  - Initial state to goal state
- In practice: unsolvable with Dijkstra
  - Lots of different approaches and representations
  - Heuristic search (A*), plan-space vs state-space, etc.

Entire area in AI: Open and interesting

- ICAPS conference, planning competitions, probabilistic planning, scheduling, ...

Planning Approaches

Planning as search

- State-space search
- Plan-space search

Dana Nau's presentation about plan-space search