

CS4618 Artificial Intelligence I

Today: Uninformed Search

Thomas Jansen

September 28th

Plans for Today

- 1 Introduction
Overview
- 2 Intelligent Agents
Introduction and Task Environments
- 3 Rational Agents
Agents
- 4 Searching
Introduction
Examples and Notation
- 5 Summary
Summary & Take Home Message

Looking at 'Dimensions of Task Environments'

Remember task environment encompasses
 performance measure, environment, actuators, sensors

- fully observable vs. partially observable
- single agent vs. multiagent
- deterministic vs. stochastic
- episodic vs. sequential
- static vs. dynamic
- discrete vs. continuous
- known vs. unknown

now examples



Seven Examples of Task Environments

Task Environment	observable	agents	determ.	episodic	static	discrete
Crossword puzzle	fully	single	determ.	sequential	static	discrete
Chess with clock	fully	multi	determ.	sequential	semi	discrete
Backgammon	fully	multi	stochastic	sequential	static	discrete
Poker	partially	multi	stochastic	sequential	static	discrete
Taxi driving	partially	multi	stochastic	sequential	dynamic	continuous
Medical diagnosis	partially	single	stochastic	sequential	dynamic	continuous
Image analysis	fully	single	determ.	episodic	static	discrete

Note

- in many examples different properties possible
- **known vs. unknown** both possible in all examples
- in particular for complex task environments, **simulators** useful

Structure of Agents

Agents consist of

- architecture
- program

Agent Architecture

- physical sensors
- physical actuators
- computing platform

Agent Program

- run on computing platform

Note agent function **vs.** agent program

- **agent function** maps **sequences** of percepts ($\hat{=}$ percept history) to action
- **agent program** maps **single** percept (and current state of the agent) to action



Agent Programs

in general maps percept to action
based on agent's state

Classes of agent programs

- table-driven agent
- simple reflex agent
- model-based reflex agent
- goal-based agent
- utility-based agent

Table-Driven Agent

Idea direct implementation of agent function

Table-Driven Agent

```
function Table-Driven-Agent(percept) returns action
  persistent sequence of percepts, initially empty
               table of actions, initially fully specified
  append percept to end of percepts
  return lookup(table, percepts)
```

- + very simple
- + completely flexible
- requires very large (maybe infinite) table
- not very practical

Simple Reflex Agent

Idea choose action based on current percept (ignoring history)

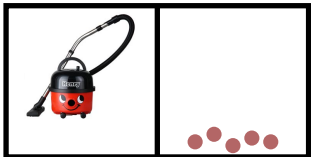
Simple Reflex Agent

```
function Simple-Reflex-Agent(percept) returns action
  persistent a set of condition-action rules, initially fully specified
  state = interpret-input(percept)
  rule = rule-match(state, rules)
  return rule.action
```

- + very simple
- + easy to understand
- + requires almost no memory
- limited flexibility

Two Tiny Examples

Vacuum Cleaner World 1



A

B

- initial set-up never changes
- sensors perceive percept
 $p \in \{A, B\} \times \{\text{dirty}, \text{clean}\}$
- action $\in \{\text{clean}, \text{left}, \text{right}\}$
- performance measure

$$m = \sum_{t=1}^{100} \# \text{clean fields in } t\text{-th step}$$

Vacuum Cleaner World 2



A

B

- initial set-up never changes
- sensors perceive percept
 $p \in \{\text{dirty}, \text{clean}\}$
- action $\in \{\text{clean}, \text{left}, \text{right}\}$
- performance measure

$$m = \sum_{t=1}^{100} \# \text{clean fields in } t\text{-th step}$$

Simple Reflex Agents for the Vacuum Cleaner Worlds

Remember action depends only on current percept

Rules for Vacuum Cleaner World 1

Current percept $p = (p_1, p_2) \in \{A, B\} \times \{\text{dirty}, \text{clean}\}$

- if $p_2 = \text{dirty}$ then clean
- if $p_2 = \text{clean}$ and $p_1 = A$ then right
- if $p_2 = \text{clean}$ and $p_1 = B$ then left

Rules for Vacuum Cleaner World 2

Current percept $p \in \{\text{dirty}, \text{clean}\}$

- if $p = \text{dirty}$ then clean
- if $p = \text{clean}$ then ?

Observation depending on starting state
any rule may lead to **infinite loop** and **bad performance**

Note randomisation **helps**

if $p = \text{clean}$ then with equal probability go either left or right

Model-Based Reflex Agent

Idea choose action based on current percept & model of world

Model-Based Reflex Agent

```
function Model-Based-Reflex-Agent(percept) returns action
  persistent current conception of world's state
               a model describing how next state depends on
               current state and action
               a set of condition-action rules, initially fully specified
               most recent action, initially empty
  state = update-state(state, action, percept, model)
  rule = rule-match(state, rules)
  return rule.action
```

- + more flexible
- + rules-part similar to simple reflex agent
- + requires limited memory
- highly dependent on model quality

Goal-Based Agent

Idea improve model-based reflex agent with respect to flexibility

Modifications

- explicitly incorporate goal
- allow goal to be modified
- incorporate **searching** to achieve goals in multiple steps if necessary

Observation

- + increased flexibility
- more complex
- less efficient



Utility-Based Agent

Idea improve goal-based agent with respect to flexibility

More concrete goals allow for binary distinction
goal achieved? yes/no
instead differentiate by means of utility measure

Modifications

- incorporate utility measure
- let choice of action additionally depend on (estimated) utility

Observation

- + increased flexibility
- more complex
- less efficient

Designing Agents

How do we implement agents?

- ① Pick a model.
- ② Implement the agent as concrete instance of the model.

Observation implementation by hand often **not feasible**
too complex, too tedious, too difficult

Alternative **learning agents**

Crucial

- distinguish **performance element** from **learning element**
- incorporate **critic** to facilitate learning
- incorporate **problem generator** to explore

Solving Problems by Searching

General Situation agent aims at maximising performance measure ($\hat{=}$ task)

- 1 Formulate **goal**.
- 2 Decide **problem formulation**, i. e., decide about states and actions to be considered.

For now, some (simplifying) assumptions

- environment fully observable
- environment discrete
- environment known
- environment deterministic

Consequence solution to problem is **fixed sequence of actions** for **any** problem

Problems in a Formal Sense

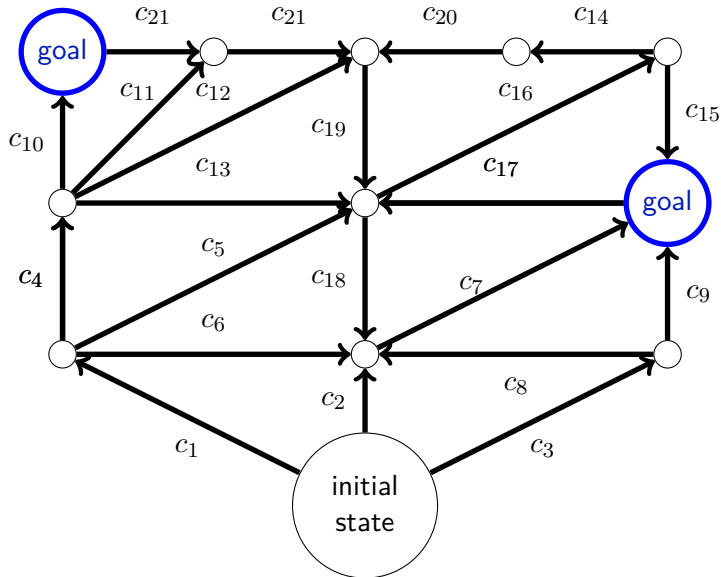
Definition (Problem)

A problem is defined by

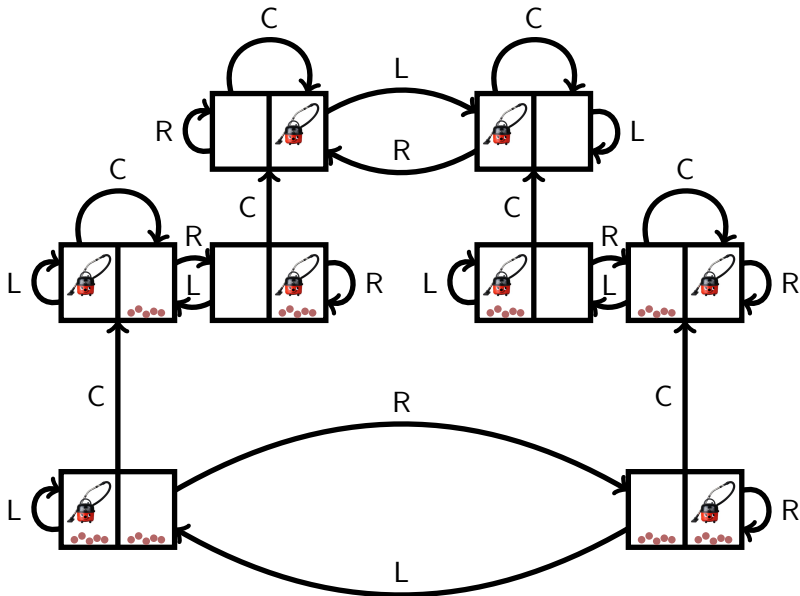
- the agent's **initial state**,
- the set of possible **actions** (applicability depending on the current state),
- a **transition model** mapping (state, action) to resulting state
- a **goal test**, indicating if a state is a goal state,
- a **path cost** assigning numerical costs to any sequence of actions.

Here $\text{path cost}(a_1, a_2, \dots, a_l) = \sum_{i=1}^l \text{path cost}(a_i)$
 and $\forall a: \text{path cost}(a) \geq 0$

Problem Visualisation



Visualisation Vacuum Cleaner World



Another Example Problem: 8-Puzzle

Initial State

7	2	4
5		6
8	3	1

Goal State

1	2	3
4	5	6
7	8	

Actions

left, right, up, down

for example ('defines' transition model)

7	2	4	$\xrightarrow{\text{up}}$	7	2	4
5		6		5	3	6
8	3	1		8		1

Cost

1 for each action



Final Example: A Number Game (Knuth (1964))

Initial State	4
Goal State	some positive integer (e. g. 5)
Actions	apply factorial (!), square root ($\sqrt{\quad}$), floor ($\lfloor \quad \rfloor$)
Transition Model	as defined by mathematical operations
Cost	1 for each action

Applying a Search Algorithm

- ① **Start** in initial state.
- ② **Expand** current state: Apply each applicable action, generating more states.
Visualisation Connect current state with each state, label connecting with corresponding action.
Observe Current state $\hat{=}$ node in tree, generated states $\hat{=}$ child nodes in this tree.
Notation All not-yet expanded notes are called **frontier**.
- ③ Select state from the frontier and continue at ②.

Remarks

- State graph may have loops, search tree contains no loops.
- Loops in state graph may lead to infinite search trees.
- Removing loops is **advantageous** since costs are non-negative.

Towards Concrete Search Algorithms

Notation for some node v in the search tree

- $v.state \hat{=}$ state in state space corresponding to v
- $v.parent \hat{=}$ node in search tree that generated v
- $v.action \hat{=}$ action that applied to $v.parent$ generated v
- $v.path-cost \hat{=}$ cost of the path from the initial state to v

Observe v easy to compute during node expansion

Data Structures for storing the frontier

- **Queue** (FIFO: first in first out)
 - $Insert(v, q)$ inserts v at the end of queue q
 - $Remove(q)$ removes first element from queue q and returns it
 - $Empty?(q)$ returns true if the queue q is empty
- **Stack** (LIFO: last in first out)
 - $Push(v, s)$ inserts v at the beginning of stack s
 - $Pop(s)$ removes first element from stack s and returns it
 - $Empty?(s)$ returns true if the stack s is empty

Measuring the Performance of Search Algorithms

- **Completeness**

Is the algorithm guaranteed to find a solution if one exists?

- **Optimality**

Is the algorithm guaranteed to find a solution with minimal past cost if one exists?

- **Time Complexity**

How long does the algorithm take to find a solution?

Remark Often proportional to the number of states generated.

- **Space Complexity**

How much memory does the algorithm need to find a solution?

Measuring with respect to what?

- **branching factor** $\hat{=}$ maximum number of applicable actions
- **depth** $\hat{=}$ minimal number of steps to a goal state
- m $\hat{=}$ maximum length of path in the state space

Summary & Take Home Message

Things to remember

- task environments: PEAS description and categorisation
- types of agents: table-driven, simple reflex, model-based reflex, goal-based, utility-based
- uninformed search

Take Home Message

- Intelligent agents and their environments are a flexible and general framework for AI (applications).
- Even restricted search problems are interesting and relevant.