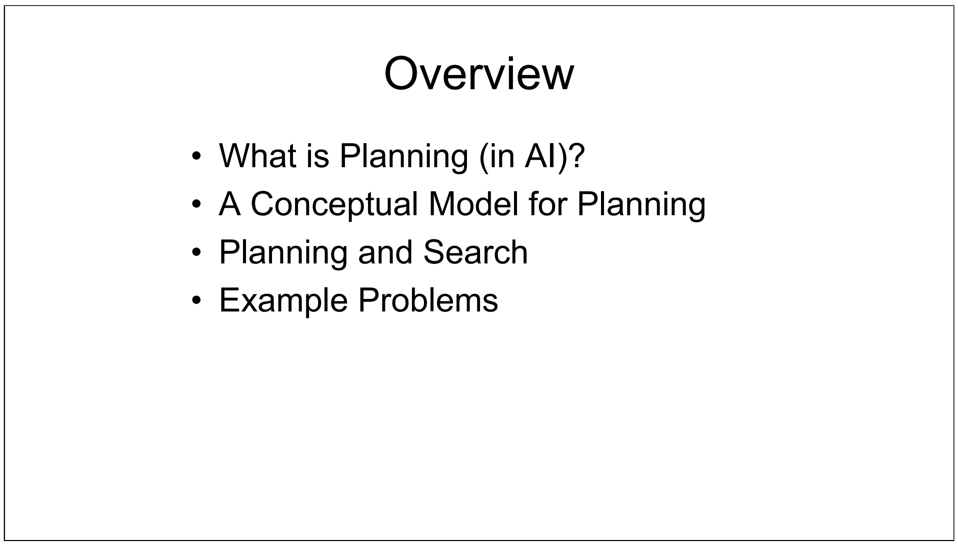
Artificial Intelligence Planning

Introduction and Overview

Artificial Intelligence Planning

•Introduction and Overview



Overview

- What is Planning (in AI)?
- A Conceptual Model for Planning
- Planning and Search
- Example Problems

Overview

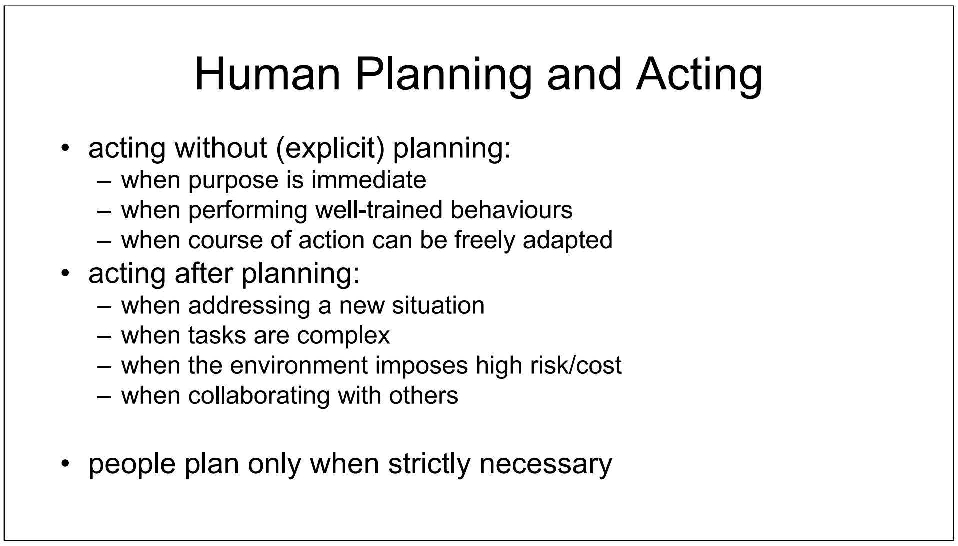
➢What is Planning (in AI)?

•now: what do we mean by (AI) planning?

•A Conceptual Model for Planning

•Planning and Search

•Example Problems



Human Planning and Acting

- acting without (explicit) planning:
  - when purpose is immediate
  - when performing well-trained behaviours
  - when course of action can be freely adapted
- acting after planning:
  - when addressing a new situation
  - when tasks are complex
  - when the environment imposes high risk/cost
  - when collaborating with others

- people plan only when strictly necessary

Human Planning and Acting

•humans rarely plan before acting in everyday situations

•**acting without (explicit) planning:** (may be subconscious)

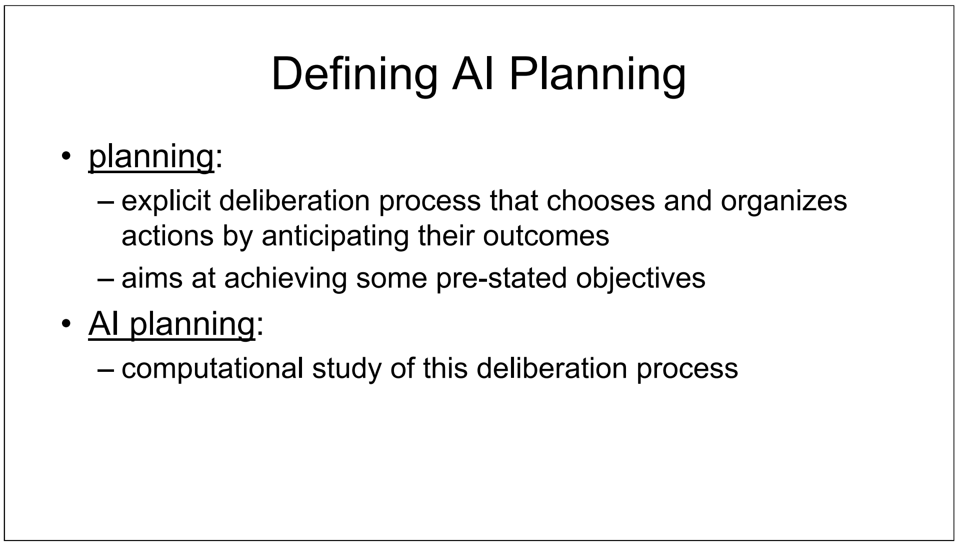
- •**when purpose is immediate** (e.g. switch on computer)
- •**when performing well-trained behaviours** (e.g. drive car)
- •**when course of action can be freely adapted** (e.g. shopping)

•**acting after planning:**

- •**when addressing a new situation** (e.g. move house)
- •**when tasks are complex** (e.g. plan this course)
- •**when the environment imposes high risk/cost** (e.g. manage nuclear power station)
- •**when collaborating with others** (e.g. build house)

•**people plan only when strictly necessary**

- •because planning is complicated and time-consuming (trade-off: cost vs. benefit)
- •often we seek only good rather than optimal plans



Defining AI Planning

- planning:
  - explicit deliberation process that chooses and organizes actions by anticipating their outcomes
  - aims at achieving some pre-stated objectives
- AI planning:
  - computational study of this deliberation process

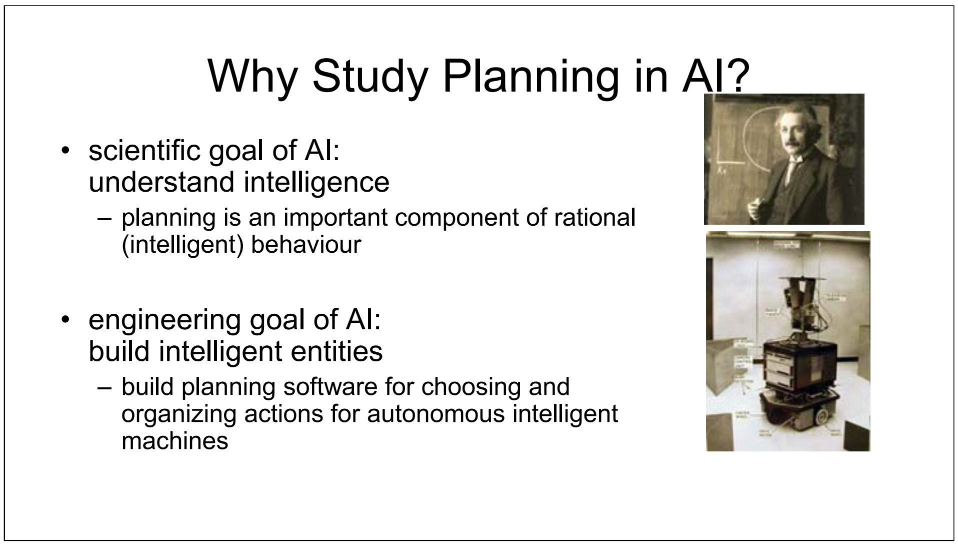
Defining AI Planning

•**planning**:

- **explicit deliberation process that chooses and organizes actions by anticipating their outcomes**
  - in short: planning is reasoning about actions
- **aims at achieving some pre-stated objectives**
  - or: achieving them as best as possible (planning as optimization problem)

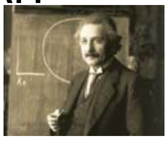
•**AI planning**:

- **computational study of this deliberation process**



Why Study Planning in AI?

- scientific goal of AI: understand intelligence
  - planning is an important component of rational (intelligent) behaviour
- engineering goal of AI: build intelligent entities
  - build planning software for choosing and organizing actions for autonomous intelligent machines



Why Study Planning in AI?

•**scientific goal of AI: understand intelligence**

•**planning is an important component of rational (intelligent) behaviour**

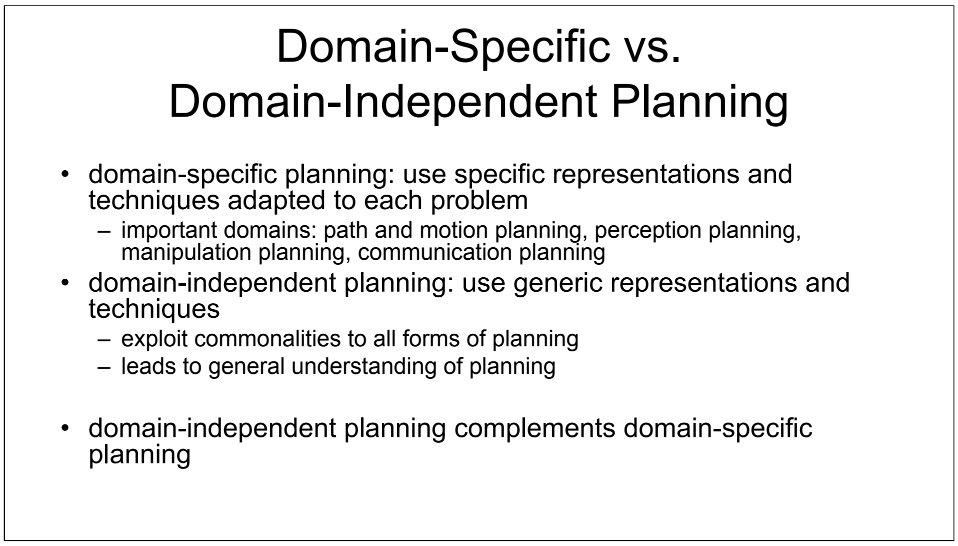
•planning is part of intelligent behaviour

•**engineering goal of AI: build intelligent entities**

•**build planning software for choosing and organizing actions for autonomous intelligent machines**

•example: Mars explorer (cannot be remotely operated)

•robot: Shakey, SRI 1968



Domain-Specific vs. Domain-Independent Planning

- domain-specific planning: use specific representations and techniques adapted to each problem
  - important domains: path and motion planning, perception planning, manipulation planning, communication planning
- domain-independent planning: use generic representations and techniques
  - exploit commonalities to all forms of planning
  - leads to general understanding of planning
- domain-independent planning complements domain-specific planning

Domain-Specific vs. Domain-Independent Planning

•domain-specific planning: use specific representations and techniques adapted to each problem

- **•important domains: path and motion planning, perception planning, manipulation planning, communication planning**

•domain-independent planning: use generic representations and techniques

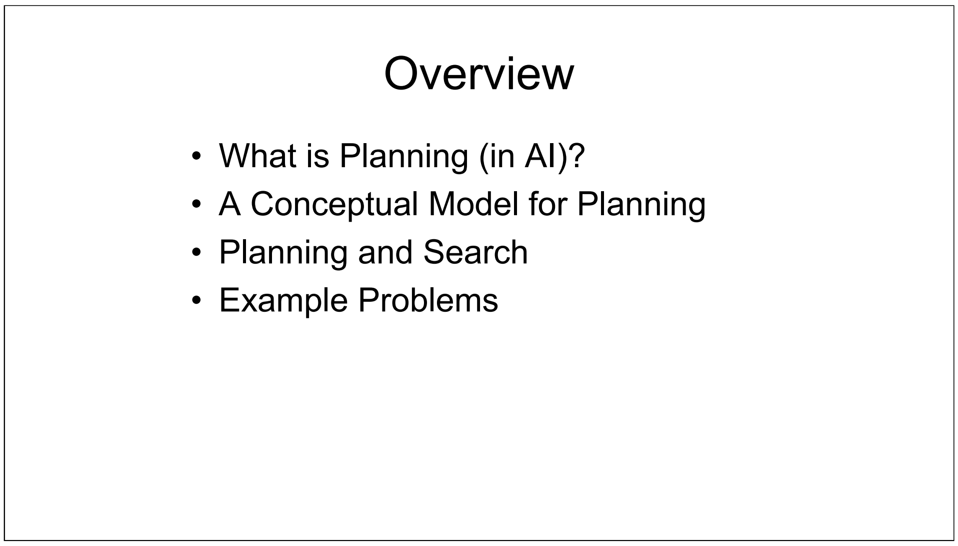
- **•exploit commonalities to all forms of planning**
  - •saves effort; no need to reinvent same techniques for different problems
- **•leads to general understanding of planning**
  - •contributes to scientific goal of AI

•domain-independent planning complements domain-specific planning

- •use domain-dependent planning where highly efficient solution is required

Quiz: True or False?

- people only plan *when they have to* because the benefit of an optimal plan does not always justify the effort of planning
- for humans, planning is a *subconscious process*, which is why computational planning is so hard
- planning involves a *mental simulation of actions* to foresee future world states and compare them to goals
- in Artificial Intelligence, planning is concerned with the search for *computationally optimal* plans
- domain-specific planning is used when efficiency is vital, whereas domain-independent planning is good for planning from first principles



Overview

- What is Planning (in AI)?
- A Conceptual Model for Planning
- Planning and Search
- Example Problems

Overview

•What is Planning (in AI)?

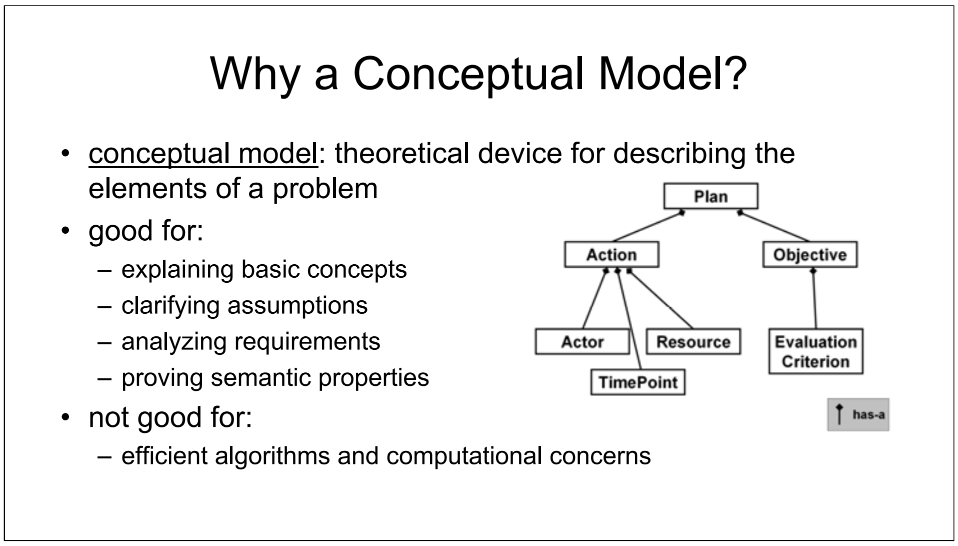
•just done: what do we mean by (AI) planning?

➢A Conceptual Model for Planning

•now: state-transition systems – formalizing the problem

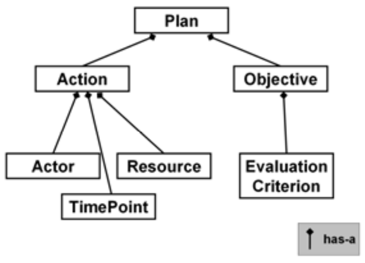
•Planning and Search

•Example Problems



Why a Conceptual Model?

- conceptual model: theoretical device for describing the elements of a problem
- good for:
  - explaining basic concepts
  - clarifying assumptions
  - analyzing requirements
  - proving semantic properties
- not good for:
  - efficient algorithms and computational concerns



Why a Conceptual Model?

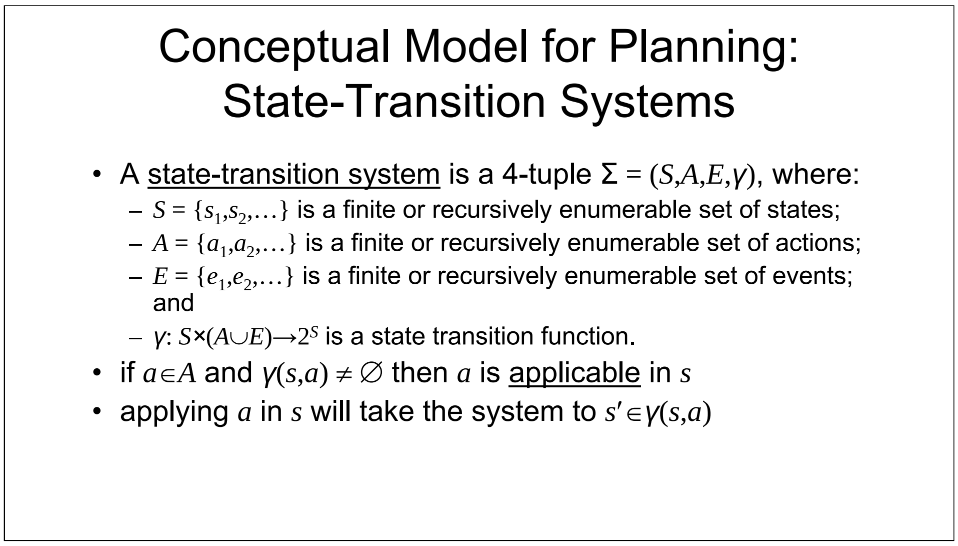
•**<u>conceptual model</u>: theoretical device for describing the elements of a problem**

•**good for:**

- •**explaining basic concepts**: what are the objects to be manipulated during problem-solving?
- •**clarifying assumptions**: what are the constraints imposed by this model?
- •**analyzing requirements**: what representations do we need for the objects?
- •**proving semantic properties**: when is an algorithm sound or complete?

•**not good for:**

- •**efficient algorithms and computational concerns**



Conceptual Model for Planning: State-Transition Systems

- A state-transition system is a 4-tuple $\Sigma = (S,A,E,\gamma)$, where:
  - $S = \{s_1,s_2,\ldots\}$ is a finite or recursively enumerable set of states;
  - $A = \{a_1,a_2,\ldots\}$ is a finite or recursively enumerable set of actions;
  - $E = \{e_1,e_2,\ldots\}$ is a finite or recursively enumerable set of events; and
  - $\gamma: S\times(A\cup E)\rightarrow 2^S$ is a state transition function.
- if $a\in A$ and $\gamma(s,a) \neq \varnothing$ then a is applicable in s
- applying a in s will take the system to $s'\in\gamma(s,a)$

Conceptual Model for Planning: State-Transition Systems

•**A state-transition system is a 4-tuple $\Sigma=(S,A,E,\gamma)$, where:**

•a general model for a dynamic system, common to other areas of computer science; aka. dynamic-event system

•**S = $\{s_1,s_2,\ldots\}$ is a finite or recursively enumerable set of states;**

•the possible states the world can be in

•**A = $\{a_1,a_2,\ldots\}$ is a finite or recursively enumerable set of actions;**

•the actions that can be performed by some agent in the world, transitions are controlled by the plan executor

•**E = $\{e_1,e_2,\ldots\}$ is a finite or recursively enumerable set of events; and**

•the events that can occur in the world, transitions that are contingent (correspond to the internal dynamics of the system)

•**γ: $S\times(A\cup E)\rightarrow 2^S$ is a state transition function.**

•notation: 2^S=powerset of S; maps to a set of states

•the function describing how the world evolves when actions or events occur

•note: model does not allow for parallelism between actions and/or events

•**if $a\in A$ and $\gamma(s,a) \neq \varnothing$ then a is applicable in s**

•**applying a in s will take the system to $s'\in\gamma(s,a)$**

State-Transition Systems as Graphs

- A state-transition system Σ = (S,A,E,γ) can be represented by a directed labelled graph $G = (N_G,E_G)$ where:
  - the nodes correspond to the states in S, i.e. $N_G=S$; and
  - there is an arc from $s \in N_G$ to $s' \in N_G$, i.e. $s \rightarrow s' \in E_G$, with label $u \in (A \cup E)$ if and only if $s' \in \gamma(s,u)$.

State-Transition Systems as Graphs

•**A state-transition system Σ=(S,A,E,γ) can be represented by a directed labelled graph G=(N_G,E_G) where:**

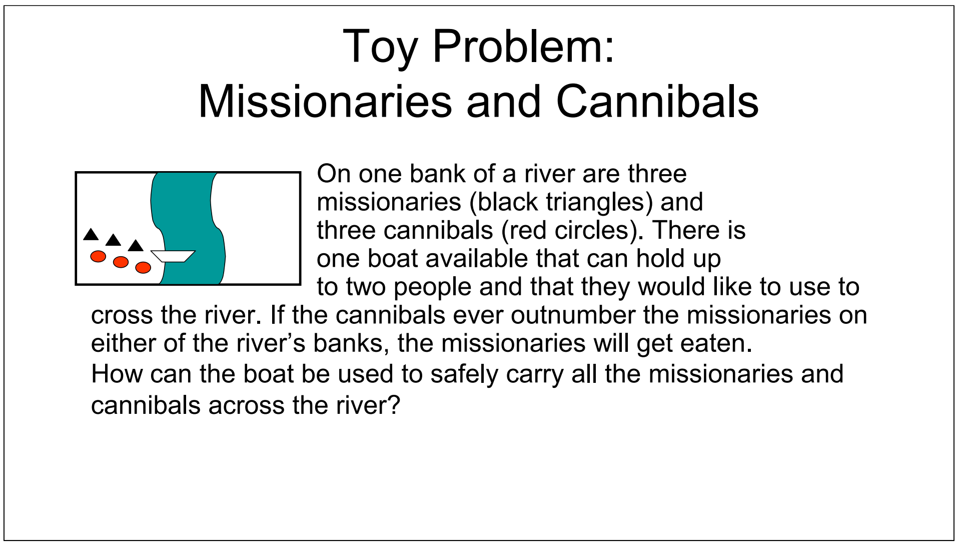
•**the nodes correspond to the states in S, i.e. N_G=S; and**

•nodes correspond to world states

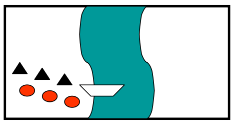
•**there is an arc from $s \in N_G$ to $s' \in N_G$, i.e. $s \rightarrow s' \in E_G$, with label $a \in (A \cup E)$ if and only if $s' \in \gamma(s,a)$.**

•there is an arc if there is an action or event that transforms one state into the other (called a state transition)

•the label of that arc is that action or event



Toy Problem: Missionaries and Cannibals



On one bank of a river are three missionaries (black triangles) and three cannibals (red circles). There is one boat available that can hold up to two people and that they would like to use to cross the river. If the cannibals ever outnumber the missionaries on either of the river's banks, the missionaries will get eaten.
How can the boat be used to safely carry all the missionaries and cannibals across the river?

Toy Problem: Missionaries and Cannibals

•On one bank of a river are three missionaries (black triangles) and three cannibals (red circles).

•description of initial state and graphical representation; other states must follow the same pattern

•There is one boat available that can hold up to two people and that they would like to use to cross the river.

•more state description: boat; implicitly: description of possible action: cross river using boat

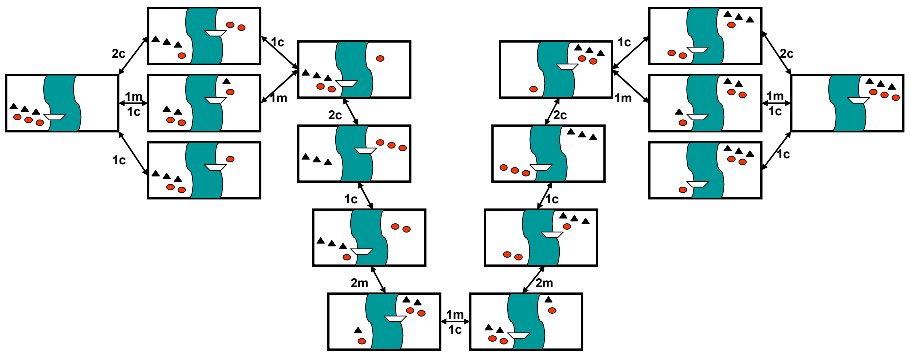
•If the cannibals ever outnumber the missionaries on either of the river's banks, the missionaries will get eaten.

•constraint on all states; must not be violated

•How can the boat be used to safely carry all the missionaries and cannibals across the river?

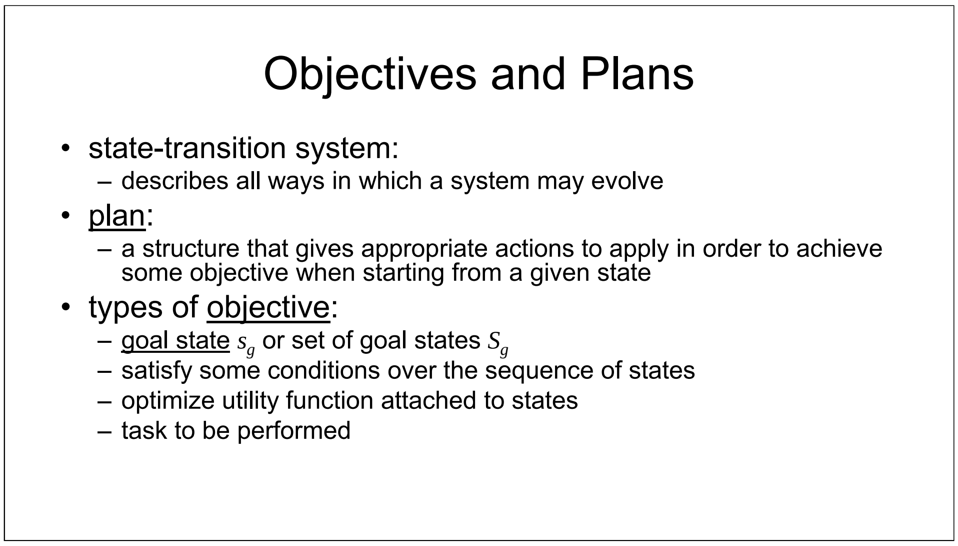
•problem: find a sequence of action that brings about another state

State-Transition Graph Example: Missionaries and Cannibals



State-Transition Graph Example: Missionaries and Cannibals

•On one bank of a river are three missionaries (black triangles) and three cannibals (red circles). There is one boat available that can hold up to two people and that they would like to use to cross the river. If the cannibals ever outnumber the missionaries on either of the river's banks, the missionaries will get eaten. How can the boat be used to safely carry all the missionaries and cannibals across the river?



Objectives and Plans

- state-transition system:
  - describes all ways in which a system may evolve
- plan:
  - a structure that gives appropriate actions to apply in order to achieve some objective when starting from a given state
- types of objective:
  - goal state s_g or set of goal states S_g
  - satisfy some conditions over the sequence of states
  - optimize utility function attached to states
  - task to be performed

Objectives and Plans

•state-transition system:

•describes all ways in which a system may evolve

•plan:

•a structure that gives appropriate actions to apply in order to achieve some objective when starting from a given state

•structure: e.g. sequential list of actions to be performed in order; function mapping states to actions

•describes a path through the state-transition graph

•types of objective:

•goal state s_g or set of goal states S_g

•simplest case; objective achieved by sequence of transitions that ends in one of the goal states

•satisfy some conditions over the sequence of states

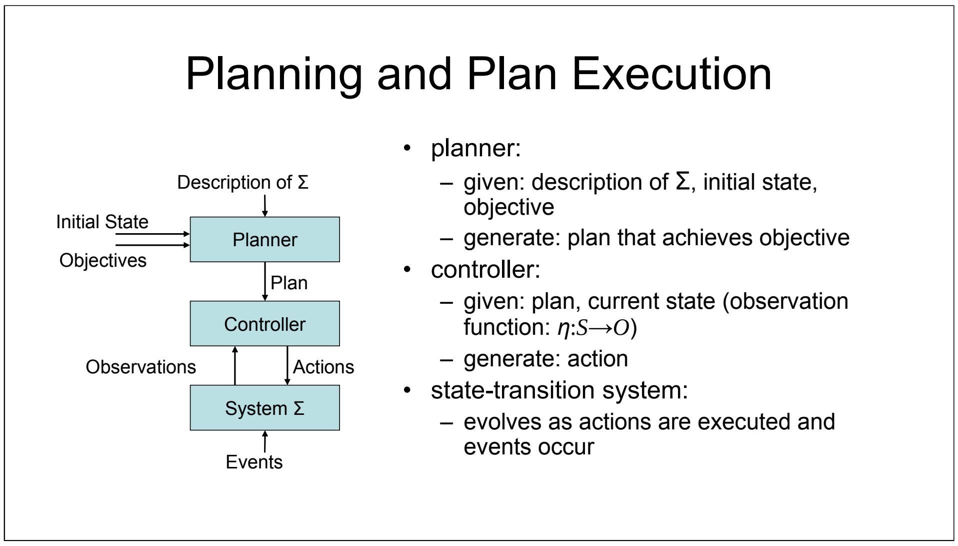
•example: states to be avoided or visited (goal state is special case)

•optimize utility function attached to states

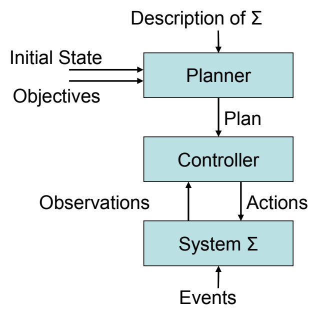
•optimize compound function (sum, max) of utilities of visited states

•task to be performed

•alternative approach: recursively defined set of actions and (other) tasks



Planning and Plan Execution



- planner:
  - given: description of Σ, initial state, objective
  - generate: plan that achieves objective
- controller:
  - given: plan, current state (observation function: η:$S \to O$)
  - generate: action
- state-transition system:
  - evolves as actions are executed and events occur

Planning and Plan Execution

•planner:

•given: description of Σ, initial state, objective

•generate: plan that achieves objective

•planner works offline: relies on formal description of Σ

•controller:

•given: plan, current state (observation function: η:$S \to O$)

•partial knowledge of controller about world modelled through observation function

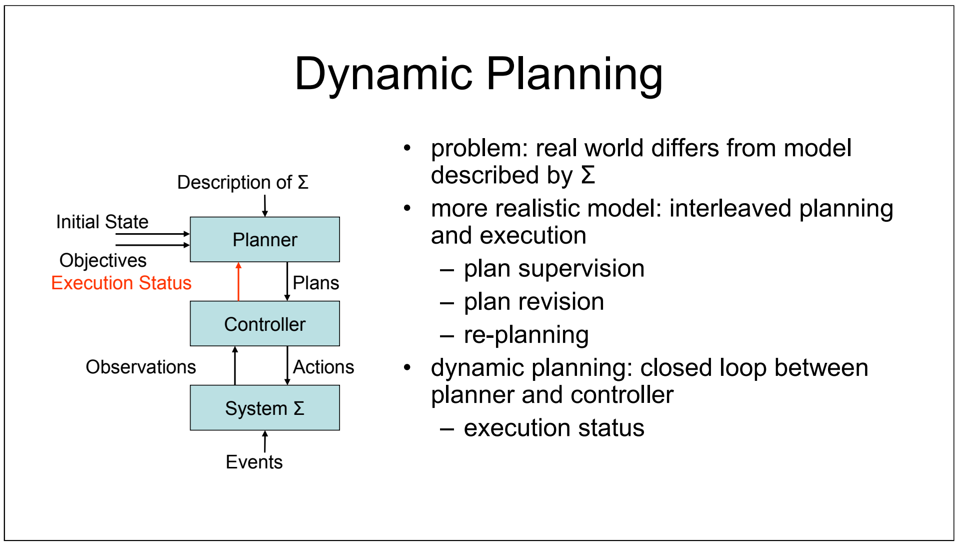
•*O* = set of possible observations (e.g. subset); input for controller

•generate: action

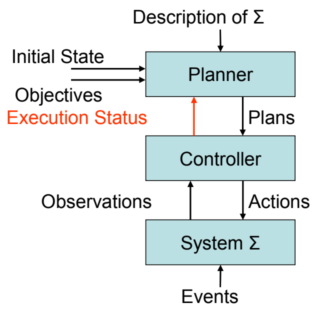
•controller works online: along with the dynamics of Σ

•state-transition system:

•evolves as actions are executed and events occur



Dynamic Planning



- problem: real world differs from model described by Σ
- more realistic model: interleaved planning and execution
  - plan supervision
  - plan revision
  - re-planning
- dynamic planning: closed loop between planner and controller
  - execution status

Dynamic Planning

•problem: physical system differs from model described by Σ

•planner only has access to model (description of Σ)

•controller must cope with differences between Σ and real world

•more realistic model: interleaved planning and execution

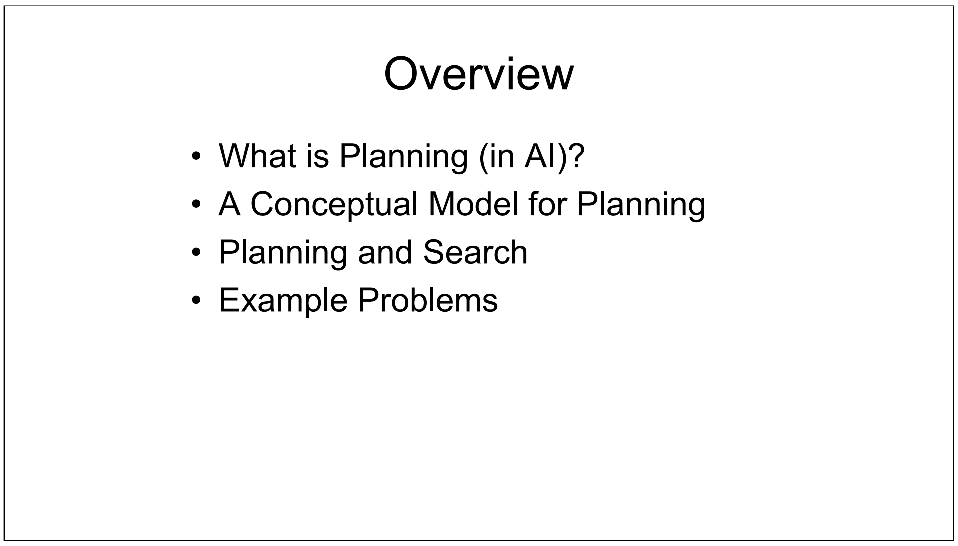
•**plan supervision**: detect when observations differ from expected results

•**plan revision**: adapt existing plan to new circumstances

•**re-planning**: generate a new plan from current (initial) state

•dynamic planning: closed loop between planner and controller

•execution status



Overview

- What is Planning (in AI)?
- A Conceptual Model for Planning
- Planning and Search
- Example Problems

Overview

•What is Planning (in AI)?

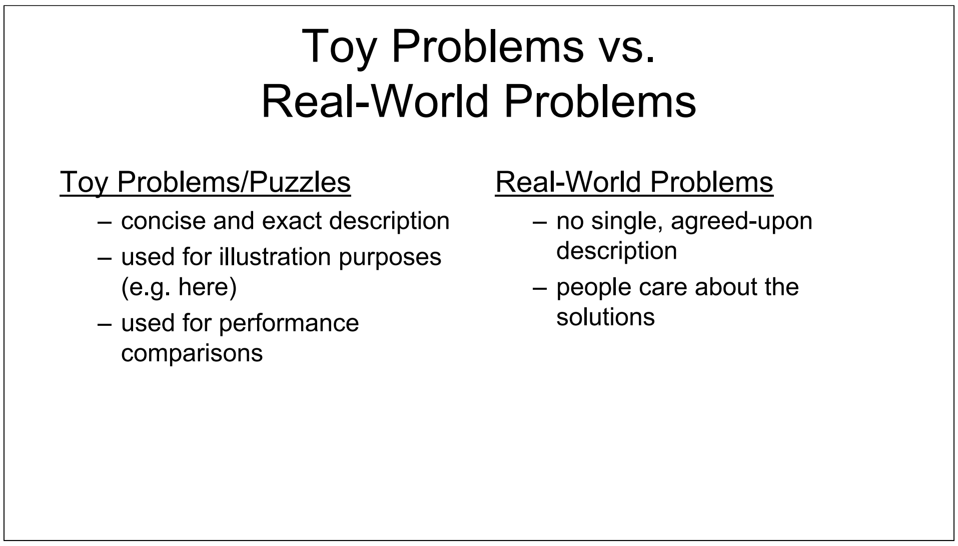
•A Conceptual Model for Planning

 •just done: state-transition systems – formalizing the problem

➢Planning and Search

 •now: problem formulation and search problems

•Example Problems



Toy Problems vs. Real-World Problems

Toy Problems/Puzzles
- concise and exact description
- used for illustration purposes (e.g. here)
- used for performance comparisons

Real-World Problems
- no single, agreed-upon description
- people care about the solutions

Toy Problems vs. Real-World Problems

•**<u>Toy Problems/Puzzles</u>**

- **•concise and exact description**
- **•used for illustration purposes (e.g. here)**
- **•used for performance comparisons**

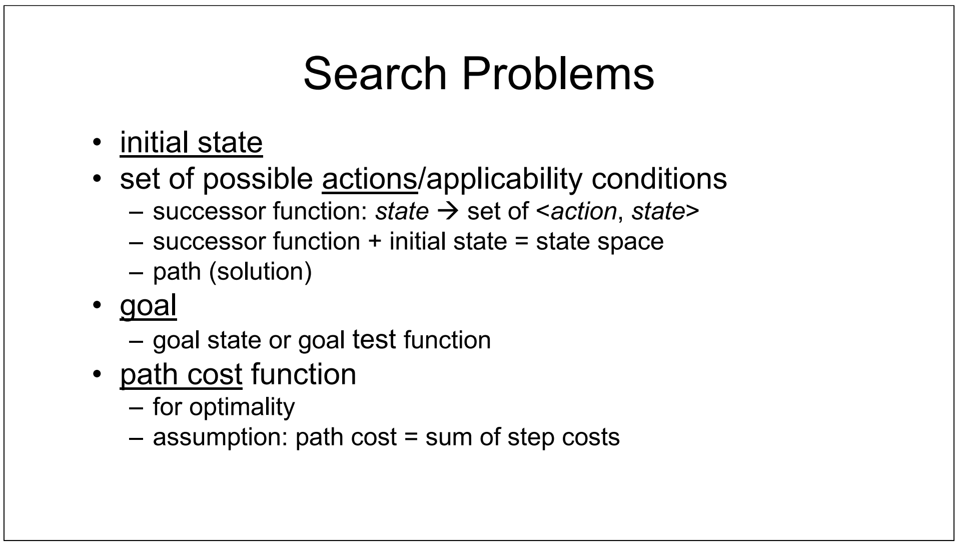
•toy problems combine

- •richness: they are anything but trivial to solve
- •simplicity: they are described easily and precisely

•**<u>Real-World Problems</u>**

- **•no single, agreed-upon description**
- **•people care about the solutions**

•most real-world problems would take hours to define



Search Problems

- <u>initial state</u>
- set of possible <u>actions</u>/applicability conditions
  - successor function: *state* → set of <*action*, *state*>
  - successor function + initial state = state space
  - path (solution)
- <u>goal</u>
  - goal state or goal test function
- <u>path cost</u> function
  - for optimality
  - assumption: path cost = sum of step costs

Search Problems

•**<u>initial state</u>**: current state the world is in (state = situation)

  •states: symbol structures representing real world objects and relations → physical symbols systems

•finite **set of possible <u>actions</u>** (aka. operators or production rules (problem formulation))**/applicability conditions**

  •**successor function: *state* → set of <*action*, *state*>**: action is applicable in given state; result of applying action in given state is paired state

  •**successor function + initial state = state space**: directed graph with states as nodes and actions as arcs

  •**path** (in the graph) **(solution)**

•**<u>goal</u>** (goal formulation)

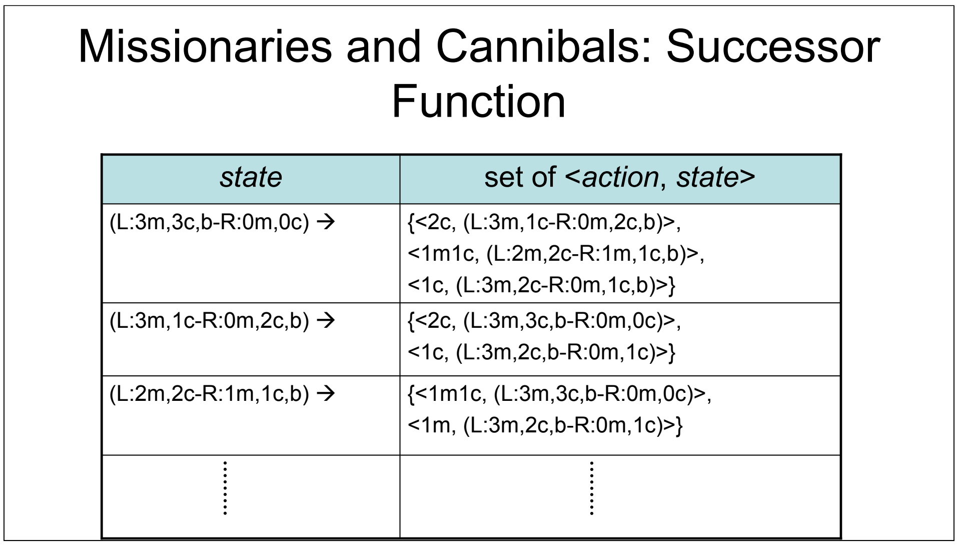
  •**goal state** (for unique goal state) **or goal test function** (for multiple goal states (e.g. in chess))

    •Solution: path in state space from initial state to goal state

•**<u>path cost</u> function**

  •**for optimality**: find solution path with minimal path cost

  •**assumption: path cost = sum of step costs** (cost of applying a given action in a given state)



Missionaries and Cannibals: Successor Function

state	set of <*action*, *state*>
(L:3m,3c,b-R:0m,0c) →	{<2c, (L:3m,1c-R:0m,2c,b)>, <1m1c, (L:2m,2c-R:1m,1c,b)>, <1c, (L:3m,2c-R:0m,1c,b)>}
(L:3m,1c-R:0m,2c,b) →	{<2c, (L:3m,3c,b-R:0m,0c)>, <1c, (L:3m,2c,b-R:0m,1c)>}
(L:2m,2c-R:1m,1c,b) →	{<1m1c, (L:3m,3c,b-R:0m,0c)>, <1m, (L:3m,2c,b-R:0m,1c)>}
⋮	⋮

Missionaries and Cannibals: Successor Function

•***state* → set of <*action*, *state*>** (domain and range: set of pairs)

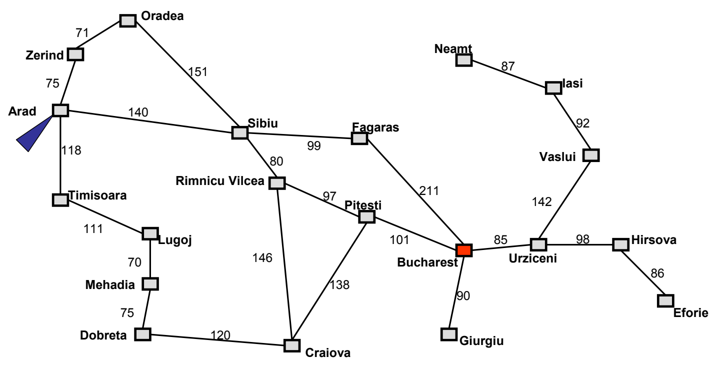
•**(L:3m,3c,b-R:0m,0c) → {<2c, (L:3m,1c-R:0m,2c,b)>, <1m1c, (L:2m,2c-R:1m,1c,b)>, <1c, (L:3m,2c-R:0m,1c,b)>}**

- •states:
  - •L/R: on left/right bank
  - •m/c: missionaries/cannibals (example: 3 missionaries and three cannibals on left bank, none on right bank)
  - •b: boat (example: boat on left bank)
- •actions:
  - •m/c: missionaries/cannibals crossing (example(s): 2 cannibals crossing (L to R), 1m and 1c crossing; 1c crossing)

•**(L:3m,1c-R:0m,2c,b) → {<2c, (L:3m,3c,b-R:0m,0c)>, <1c, (L:3m,2c,b-R:0m,1c)>}** (note: only two actions applicable)

•**(L:2m,2c-R:1m,1c,b) → {<1m1c, (L:3m,3c,b-R:0m,0c)>, <1m, (L:3m,2c,b-R:0m,1c)>}**

Real-World Problem: Touring in Romania

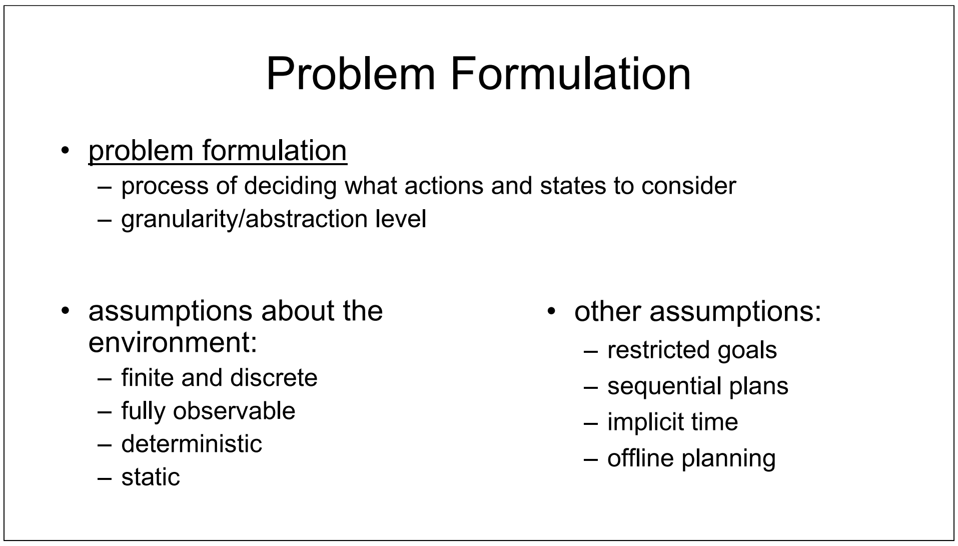


Real-World Problem: Touring in Romania

•shown: rough map of Romania

•initial state: on vacation in Arad, Romania

•goal? actions? -- “Touring Romania” cannot readily be described in terms of possible actions, goals, and path cost



Problem Formulation

- problem formulation
  - process of deciding what actions and states to consider
  - granularity/abstraction level
- assumptions about the environment:
  - finite and discrete
  - fully observable
  - deterministic
  - static
- other assumptions:
  - restricted goals
  - sequential plans
  - implicit time
  - offline planning

Problem Formulation

•**problem formulation**

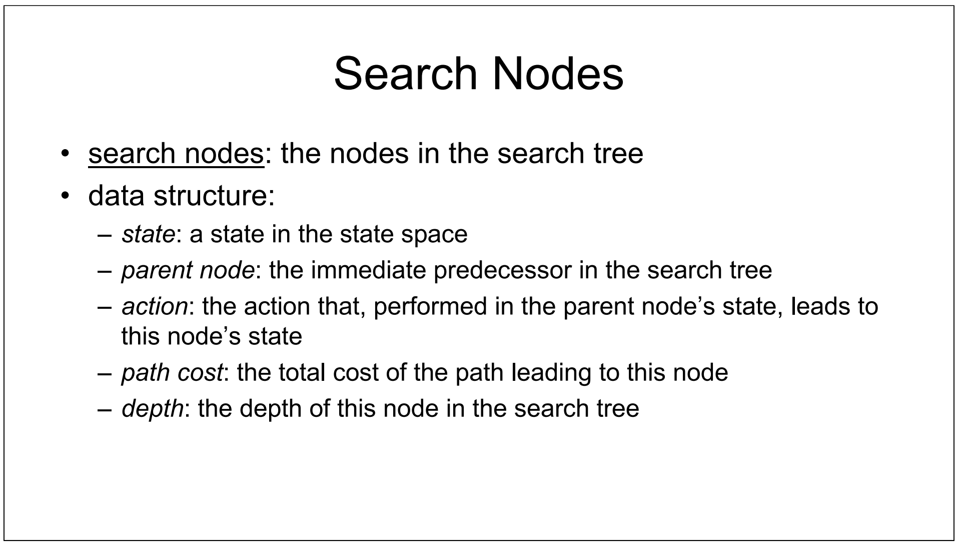
- •**process of deciding what actions and states to consider**
- •**granularity/abstraction level**
- •TR example:
  - •possible actions: turn steering wheel by one degree left/right; move left foot … → too much (irrelevant) detail, uncertainty, too many steps in solution
  - •possible action: drive to Bucharest → How to execute such an action?
  - •best level of granularity: drive to another major town to which there is a direct road

•**assumptions about the environment:**

- •**finite and discrete**: we can enumerate all the possible actions and states (else: continuous)
- •**fully observable**: agent can see what current state it is in
- •**deterministic**: applying an action in a given state leads to exactly one (usually different) state
- •**static**: environment does not change while we think (no events)

•**other assumptions:**

- •**restricted goals:** given as an explicit goal state s_g or set of goal states S_g
- •**sequential plans:** solution plan is a linearly ordered finite sequence of actions
- •**implicit time:** actions and events have no duration
- •**offline planning:** planner is not concerned with changes of Σ while planning



Search Nodes

- <u>search nodes</u>: the nodes in the search tree
- data structure:
  - *state*: a state in the state space
  - *parent node*: the immediate predecessor in the search tree
  - *action*: the action that, performed in the parent node’s state, leads to this node’s state
  - *path cost*: the total cost of the path leading to this node
  - *depth*: the depth of this node in the search tree

Search Nodes

•**<u>search nodes</u>: the nodes in the search tree**

  •node is a bookkeeping structure in a search tree

•**data structure:**

  •***state*: a state in the state space**

    •state (vs. node) corresponds to a configuration of the world

    •two nodes may contain equal states

  •***parent node*: the immediate predecessor in the search tree**

    •nodes are on paths (defined by parent nodes)

  •***action*: the action that, performed in the parent node’s state, leads to this node’s state**

  •***path cost*: the total cost of the path leading to this node**

  •***depth*: the depth of this node in the search tree**

•alternative: representing paths only (sequences of actions):

  •possible, but state provides direct access to valuable information that might be expensive to regenerate all the time

General Tree Search Algorithm

```
function treeSearch(problem, strategy)
  fringe ← { new searchNode(problem.initialState) }
  loop
    if empty(fringe) then return failure
    node ← selectFrom(fringe, strategy)
    if problem.goalTest(node.state) then
      return pathTo(node)
    fringe ← fringe + expand(problem, node)
```

General Tree Search Algorithm

function treeSearch(*problem*, *strategy*)

•find a solution to the given problem while expanding nodes according to the given strategy

***fringe* ← { new searchNode(*problem*.initialState) }**

•*fringe*: set of known states; initially just initial state

loop

•possibly infinite loop expands nodes

if empty(*fringe*) then return failure

•complete tree explored; no goal state found

***node* ← selectFrom(*fringe*, *strategy*)**

•select node from fringe according to search control strategy; the node will not be selected again

if *problem*.goalTest(*node.state*) then

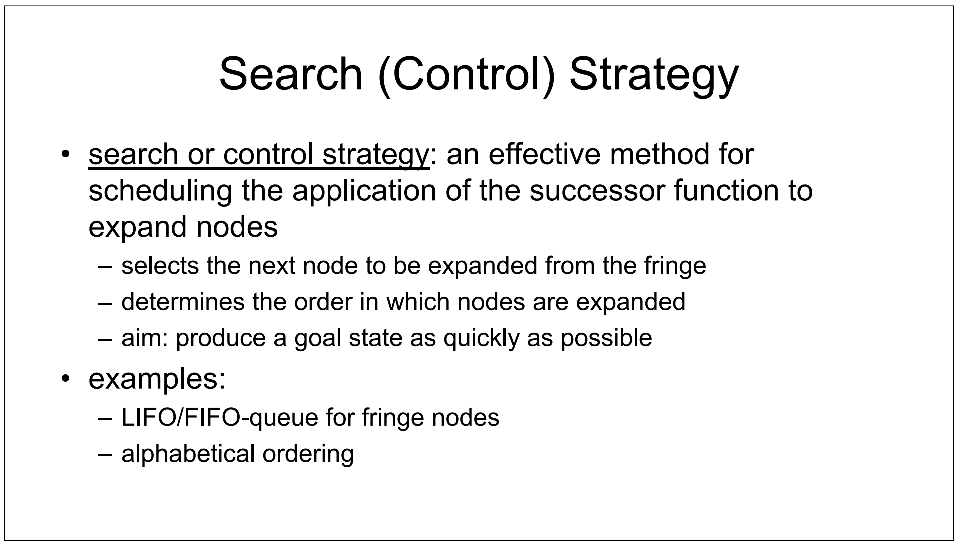
•goal test before expansion: to avoid trick problem like “get from Arad to Arad”

return pathTo(*node*)

•success: goal node found

***fringe* ← *fringe* + expand(*problem*, *node*)**

•otherwise: add new nodes to the fringe and continue loop



Search (Control) Strategy

- <u>search or control strategy</u>: an effective method for scheduling the application of the successor function to expand nodes
  - selects the next node to be expanded from the fringe
  - determines the order in which nodes are expanded
  - aim: produce a goal state as quickly as possible
- examples:
  - LIFO/FIFO-queue for fringe nodes
  - alphabetical ordering

Search (Control) Strategy

•<u>search or control strategy</u>: an effective method for scheduling the application of the successor function to expand nodes

•removes non-determinism from search method

•selects the next node to be expanded from the fringe

•closed nodes never need to be expanded again

•determines the order in which nodes are expanded

•exact order makes method deterministic

•aim: produce a goal state as quickly as possible

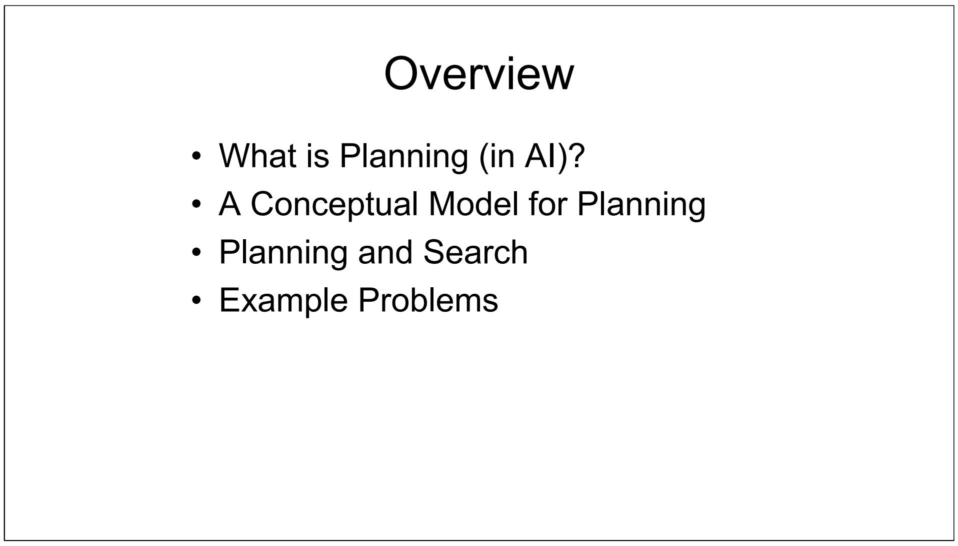
•strategy that produces goal state quicker is usually considered better

•examples:

•LIFO/FIFO-queue for fringe nodes (two fundamental search strategies)

•alphabetical ordering

•remark: complete search tree is usually too large to fit into memory, strategy determines which part to generate



Overview

- What is Planning (in AI)?
- A Conceptual Model for Planning
- Planning and Search
- Example Problems

Overview

•What is Planning (in AI)?

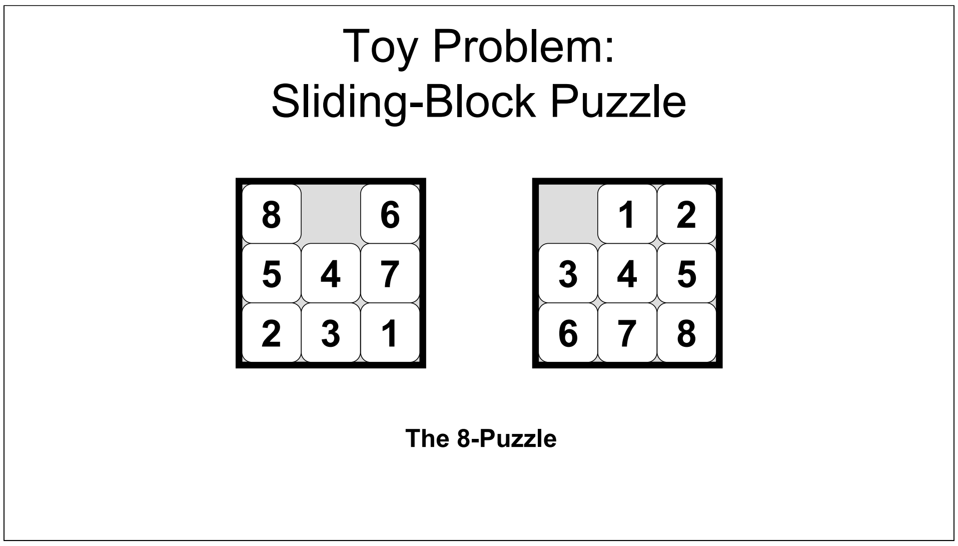
•A Conceptual Model for Planning

•Planning and Search

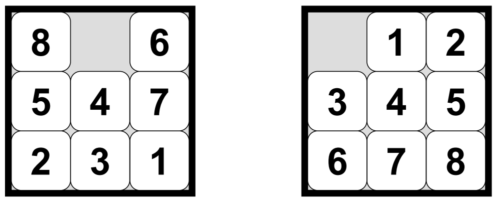
    •just done: problem formulation and search problems

➢Example Problems

    •now: more examples incl. DWR problem



Toy Problem: Sliding-Block Puzzle



The 8-Puzzle

Toy Problem: Sliding-Block Puzzle

•**[images]**

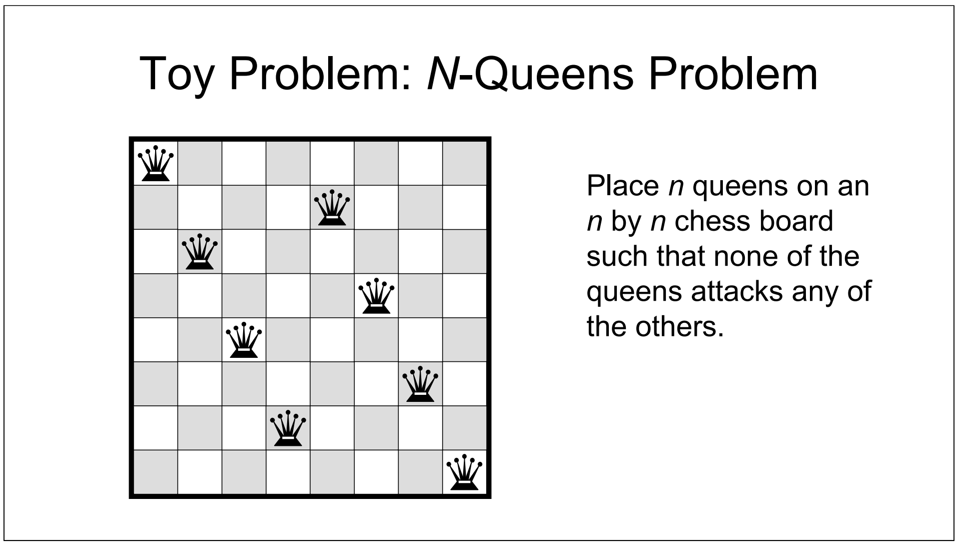
•**The 8-Puzzle**

•problem description:

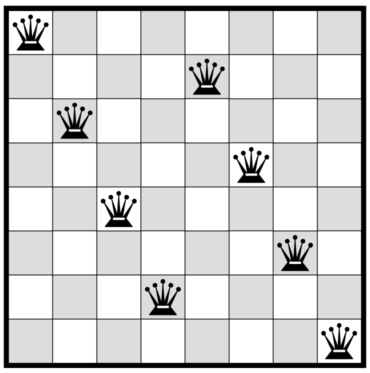
- •states: location of each tile and the blank (9!/2 = 181440 states)
- •initial state: any (reachable) state, e.g. one shown on left (worst case: solution requires at least 31 steps)
- •actions: good formulation: move blank left/right/up/down
- •goal test: goal state (shown right)
- •step cost: 1

•generalization: sliding-tile puzzles, e.g. 15-puzzle, 24-puzzle

•problem class is NP-complete



Toy Problem: *N*-Queens Problem



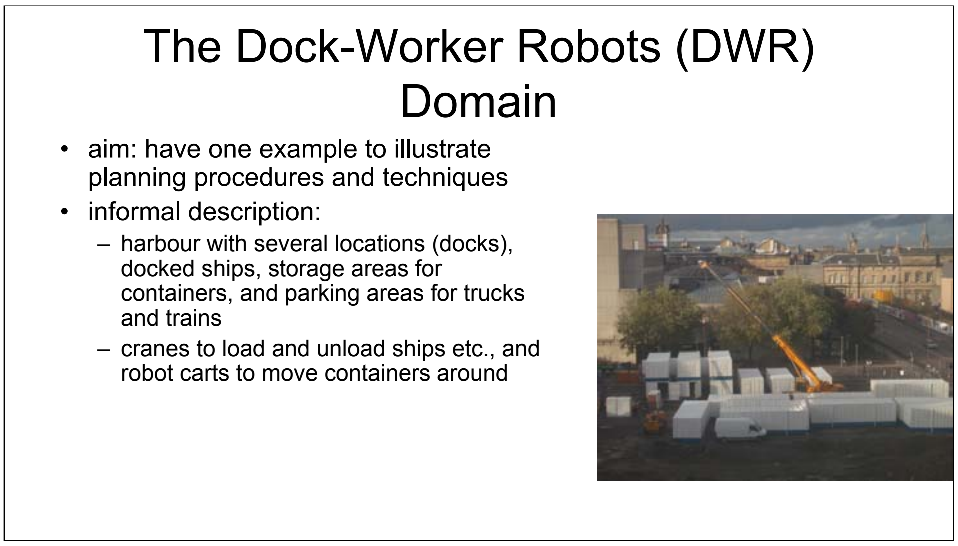
Place *n* queens on an *n* by *n* chess board such that none of the queens attacks any of the others.

Toy Problem: *N*-Queens Problem

•**Place *n* queens on an *n* by *n* chess board such that none of the queens attacks any of the others.**

- •problem: no row, column, or diagonal must contain more than one queen
- •path cost irrelevant; looking for goal state

•configuration shown is not a goal state



The Dock-Worker Robots (DWR) Domain

- aim: have one example to illustrate planning procedures and techniques
- informal description:
  - harbour with several locations (docks), docked ships, storage areas for containers, and parking areas for trucks and trains
  - cranes to load and unload ships etc., and robot carts to move containers around

The Dock-Worker Robots (DWR) Domain

•**aim: have one example to illustrate planning procedures and techniques**

 •problem must be nontrivial to be interesting, but not too much overhead to introduce problem

•**informal description:**

 •generalization of earlier example (state transition graph)

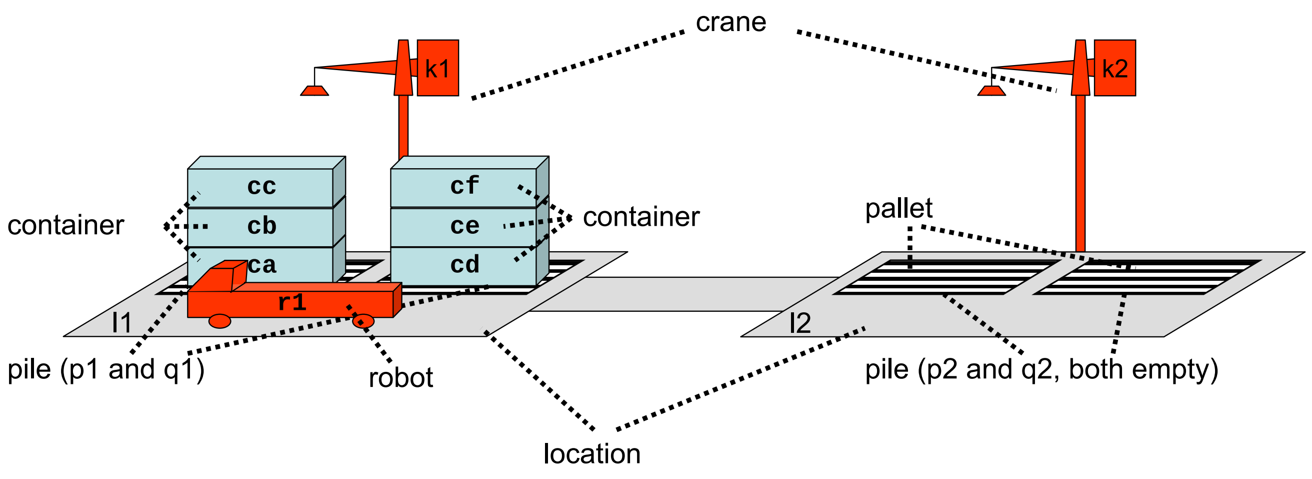
 •**harbour with several locations (docks), docked ships, storage areas for containers, and parking areas for trucks and trains**

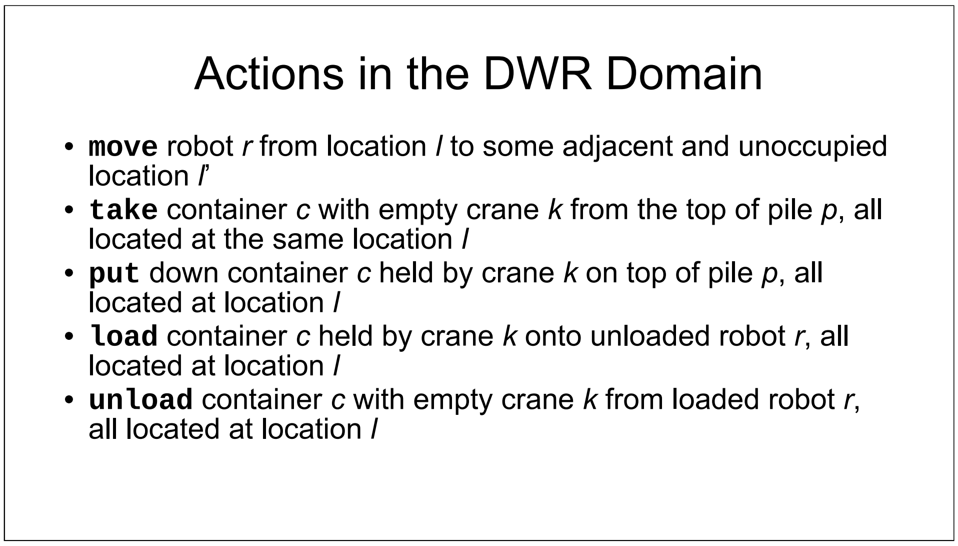
 •**cranes to load and unload ships etc., and robot carts to move containers around**

•simplified and enriched version of this domain will be introduced later

•approach: use first-order predicate logic as representation

DWR Example State





Actions in the DWR Domain

- **`move`** robot *r* from location *l* to some adjacent and unoccupied location *l'*
- **`take`** container *c* with empty crane *k* from the top of pile *p*, all located at the same location *l*
- **`put`** down container *c* held by crane *k* on top of pile *p*, all located at location *l*
- **`load`** container *c* held by crane *k* onto unloaded robot *r*, all located at location *l*
- **`unload`** container *c* with empty crane *k* from loaded robot *r*, all located at location *l*

Actions in the DWR Domain

•move robot *r* from location *l* to some adjacent and unoccupied location *l'*

•take container *c* with empty crane *k* from the top of pile *p*, all located at the same location *l*

•put down container *c* held by crane *k* on top of pile *p*, all located at location *l*

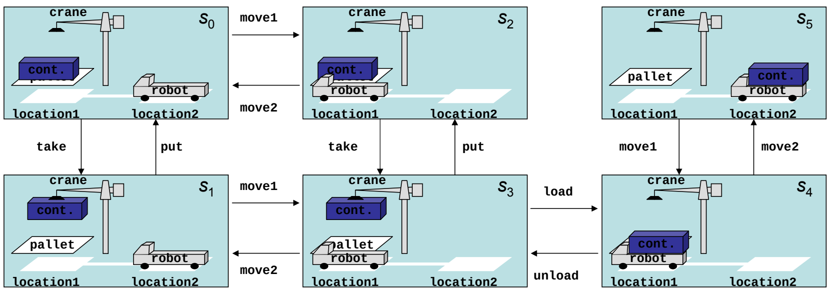
•load container *c* held by crane *k* onto unloaded robot *r*, all located at location *l*

•unload container *c* with empty crane *k* from loaded robot *r*, all located at location *l*

•formal specifications will follow when we have introduced a formal action description language

•problem: how to represent actions formally? first-order logic?

State-Transition Systems: Graph Example



State-Transition Systems: Graph Example

•states: s_0 to s_5

•objects: robot, crane, container, pallet, two locations

•actions:

•crane can take/put the container from the pallet/onto the pallet

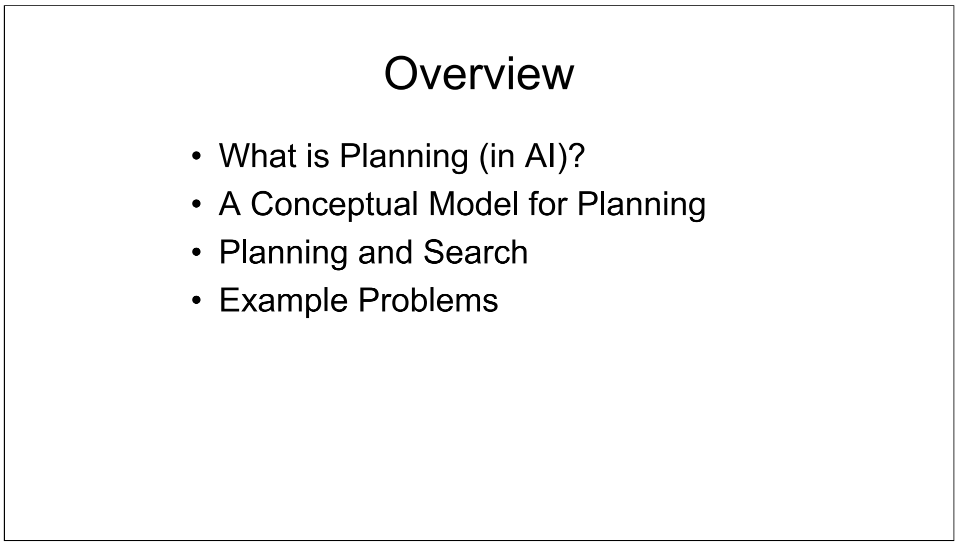
•crane can load/unload the container from the robot

•robot can drive to either location (with or without the container loaded)

•no events

•state transition function: arcs shown in graph

•note: state transitions are deterministic: each action leads to at most one other state



Overview

- What is Planning (in AI)?
- A Conceptual Model for Planning
- Planning and Search
- Example Problems

Overview

•What is Planning (in AI)?

•A Conceptual Model for Planning

•Planning and Search

•Example Problems

•just done : more examples incl. DWR problem