COMP3411/9814: Artificial Intelligence

Solving problems by searching

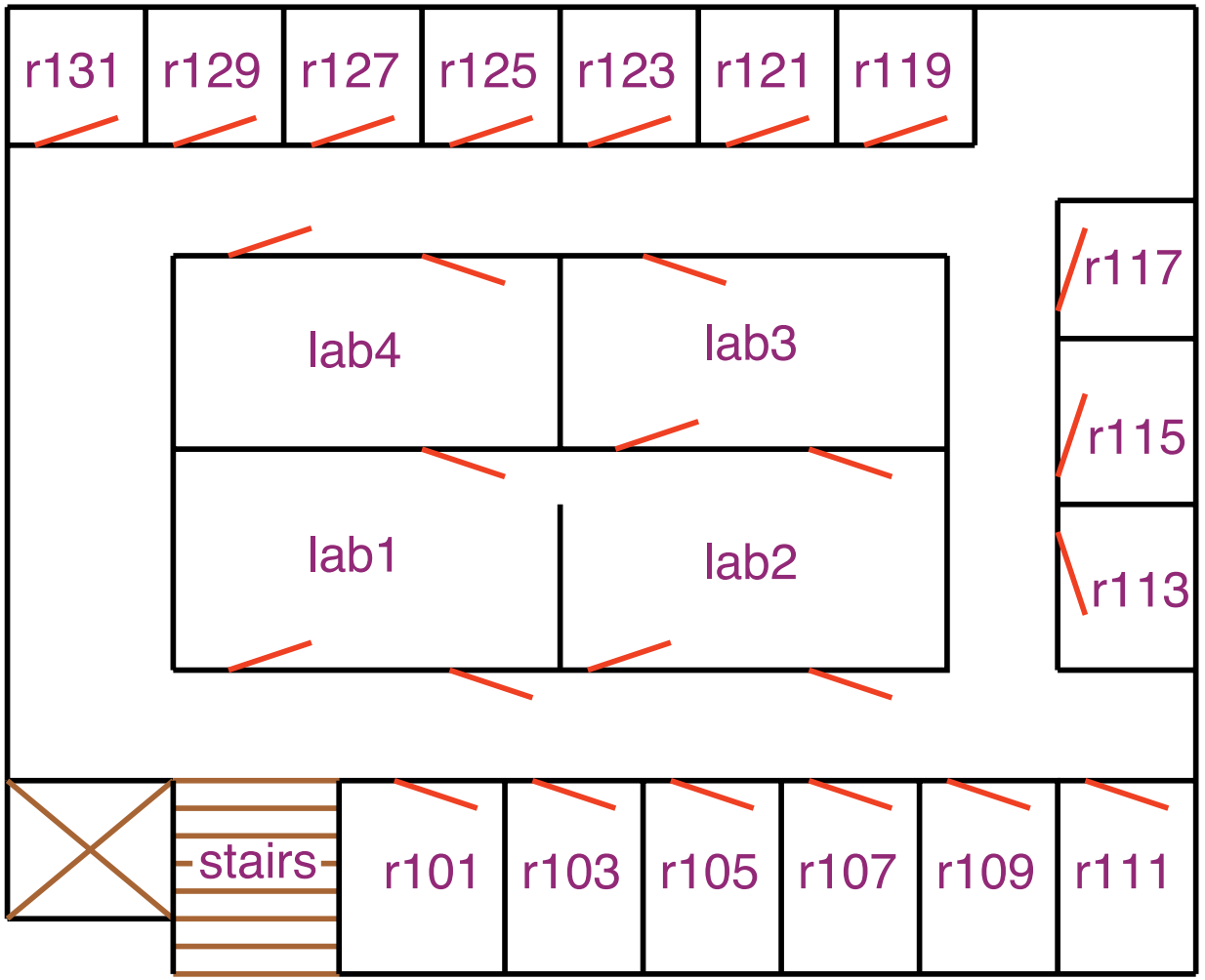
Lecture Overview

- Problem-solving agents
- Problem types
- Problem formulation
- Example problems
- Basic search algorithms

Four Example Application Domains

- Autonomous delivery robot roams around an office environment and delivers coffee, parcels,. . .
- Diagnostic assistant helps a human troubleshoot problems and suggests repairs or treatments. E.g., electrical problems, medical diagnosis.
- Intelligent tutoring system teaches students in some subject area.
- Trading agent buys goods and services on your behalf.

Domain for Delivery Robot



Autonomous Delivery Robot

Example inputs:

- Abilities: movement, speech, pickup and place objects.
- Prior knowledge: its capabilities, objects it may encounter, maps.
- Past experience: which actions are useful and when, what objects are there, how its actions affect its position.
- Goals: what it needs to deliver and when, tradeoffs between acting quickly and acting safely.
- Observations: about its environment from cameras, sonar, sound, laser range finders, or keyboards.

What does the Delivery Robot need to do?

- Determine where Craig's office is. Where coffee is. . .
- Find a path between locations.
- Plan how to carry out multiple tasks.
- Make default assumptions about where Craig is.
- Make tradeoffs under uncertainty: should it go near the stairs?
- Learn from experience.
- Sense the world, avoid obstacles, pickup and put down coffee.

Searching

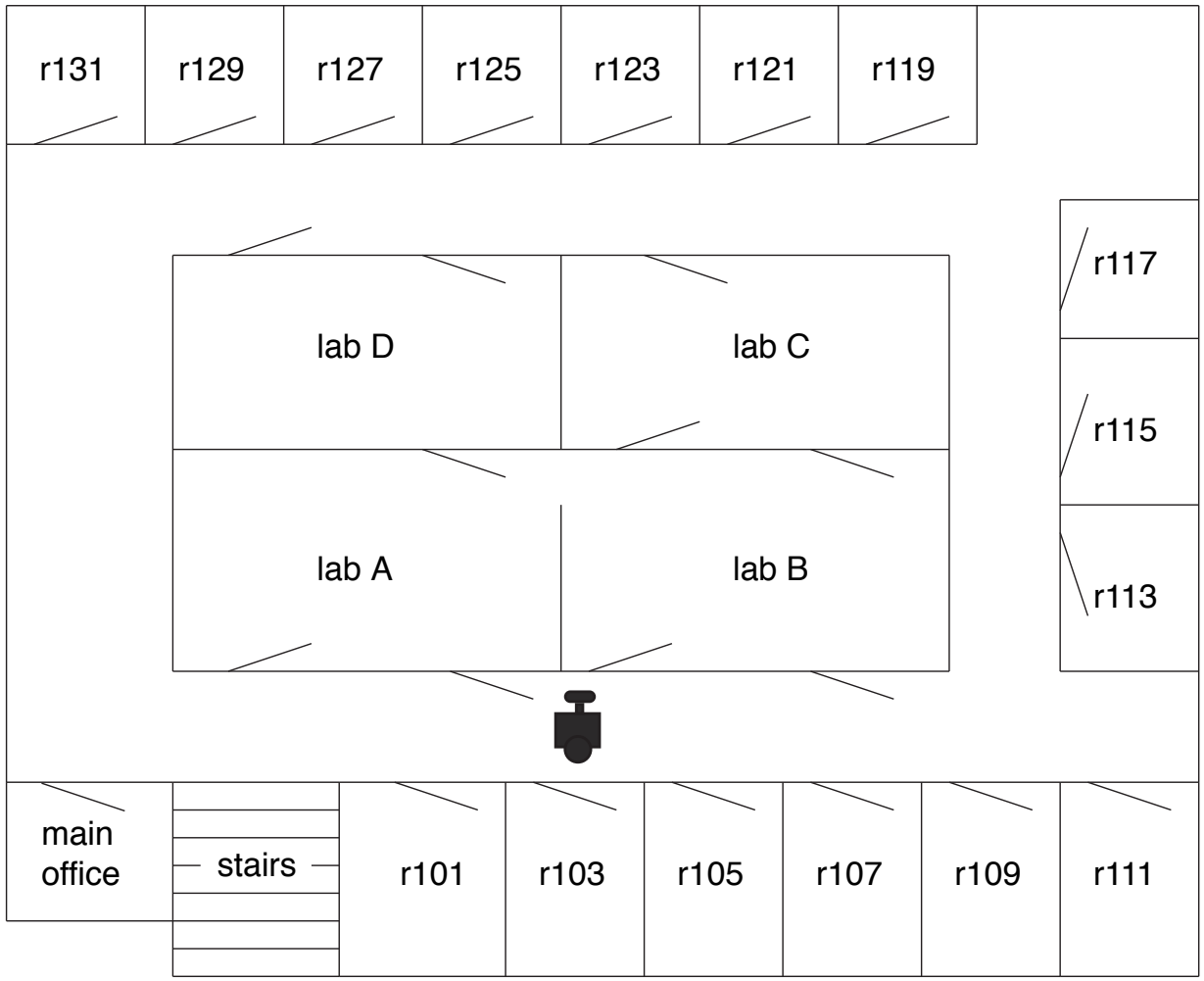
- Often we are not given an algorithm to solve a problem, but only a specification of what is a solution — we have to search for a solution.
- A typical problem is when the agent is in one state, it has a set of deterministic actions it can carry out, and wants to get to a goal state.
- Many AI problems can be abstracted into the problem of finding a path in a directed graph.
- Often there is more than one way to represent a problem as a graph.

Motivation

- Reactive and Model-Based Agents choose their actions based only on what they currently perceive, or have perceived in the past.
- A Planning Agent can use Search techniques to plan several steps ahead in order to achieve its goal(s).
- Two classes of search strategies:
  - Uninformed search strategies can only distinguish goal states from non-goal states
  - Informed search strategies use heuristics to try to get "closer" to the goal

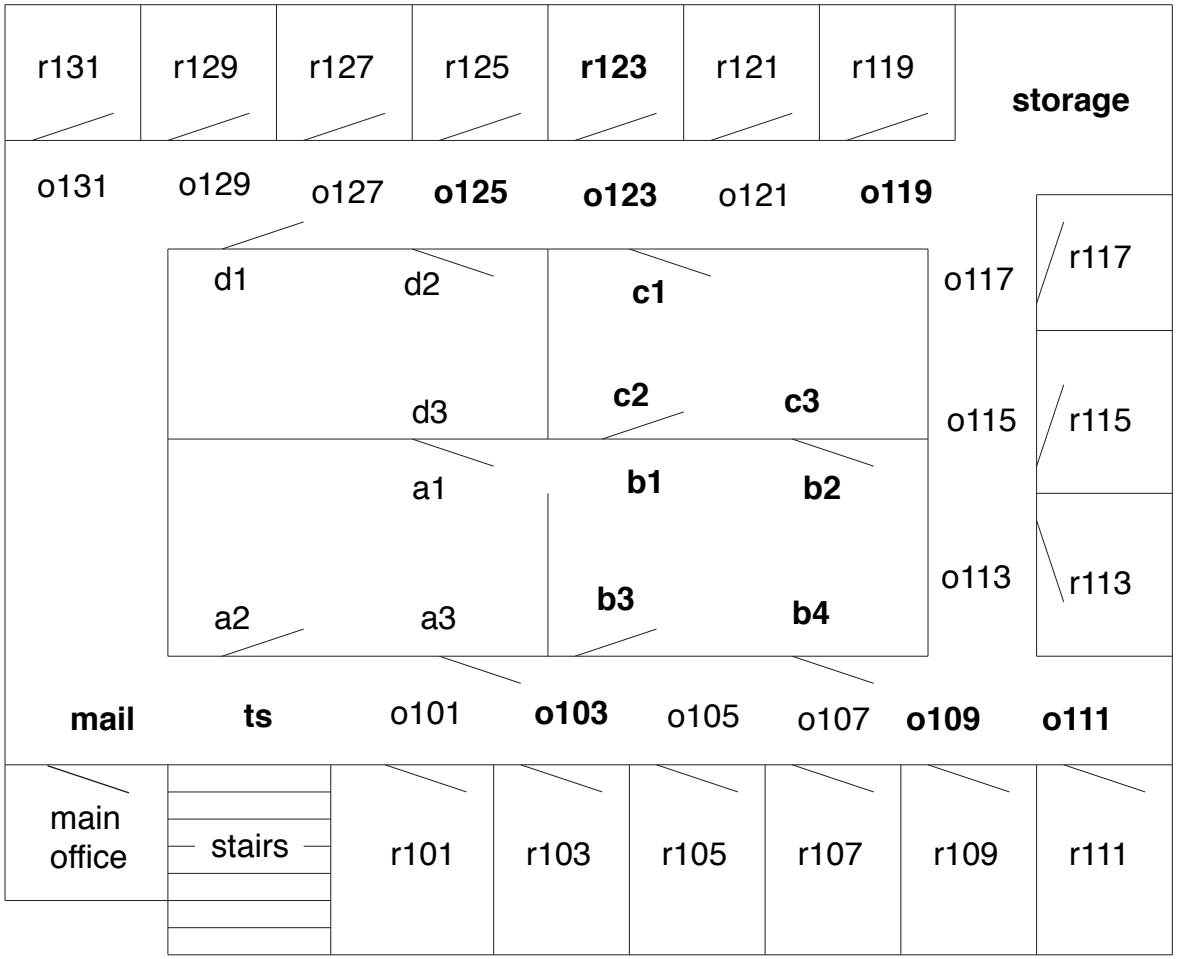
Example Problem for Delivery Robot

The robot wants to get from outside room 103 to the inside of room 123.



Example Problem for Delivery Robot

The robot wants to get from outside room 103 to the inside of room 123.



The delivery robot domain with interesting locations labelled

State Space

- One general formulation of intelligent action is in terms of a **state space**.
- A **state** contains all of the information necessary to predict the effects of an action and to determine whether a state satisfies the goal.
- State-space searching assumes:
  - The agent has perfect knowledge of the state space and is planning for the case where it observes what state it is in: there is full observability.
  - The agent has a set of actions that have known deterministic effects.
  - The agent can determine whether a state satisfies the goal.
- A **solution** is a sequence of actions that will get the agent from its current state to a state that satisfies the goal.

State-space Problem

A state-space problem consists of

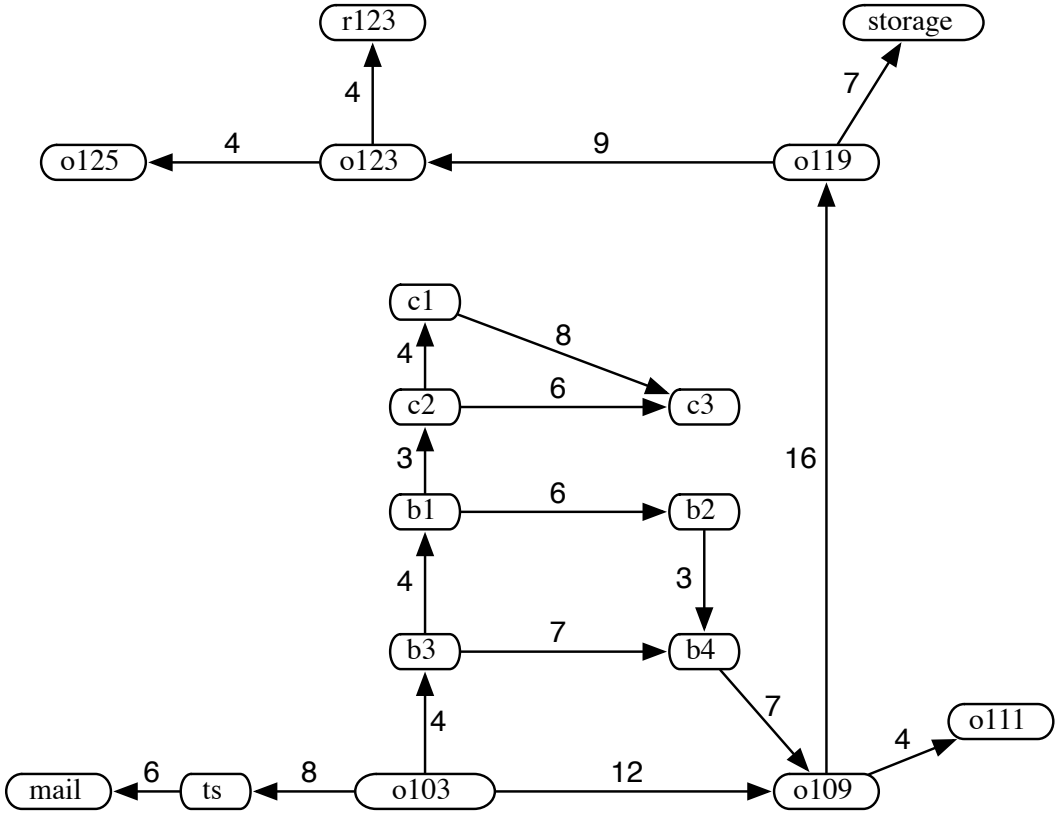
- a set of states
- a subset of states called the start states
- a set of actions
- an action function: given a state and an action, returns a new state
- a set of goal states, specified as function, *goal(s)*
- a criterion that specifies the quality of an acceptable solution.

Example Problem for Delivery Robot

The robot wants to get from outside room 103 to the inside of room 123.

- Step 1 Formulate goal:
  - be in Bucharest on time
- Step 2 Formulate problem - Specify task:
  - states: various cities
  - actions (operators) (= transitions between states): moving between rooms
- Step 3 Find solution (= action sequences): sequence of rooms
  - A solution is a sequence of actions that moves the robot from o103 to room r123
- Step 4 Execute: move through all locations given by the solution.

State-Space Graph for the Delivery Robot



This can be modeled as a state-space search problem, where the states are locations.

Directed Graphs

- A (directed) graph consists of a set N of nodes and a set A of ordered pairs of nodes, called arcs.
- Node n_2 is a neighbor of n_1 if there is an arc from n_1 to n_2. That is, if $\langle n_1, n_2 \rangle \in A$.
- A path is a sequence of nodes $\langle n_0, n_1, \ldots, n_k \rangle$ such that $\langle n_{i-1}, n_i \rangle \in A$.
- The length of path $\langle n_0, n_1, \ldots, n_k \rangle$ is k.
- Given a set of start nodes and goal nodes, a solution is a path from a start node to a goal node.

Single-state Task Specification

A task is specified by states and actions: :

- state space e.g. other rooms
- initial state e.g., ” room 103”
- actions or operators (or successor function *S(x)*) = set of action–state pairs
- goal test, check if a state is goal state
  In this case, there is only one goal specified (“inside of room 123. ”)
- path cost e.g. sum of distances, number of actions etc.
- A solution is a sequence of actions leading from the initial state to a goal state

A state-space problem

A **state-space problem** consists of

- a distinguished state called the **start state**
- for each state, a set of actions available to the agent in that state
- an **action function** that, given a state and an action, returns a new state
- a **goal** specified as a Boolean function goal(s) that is true when state s satisfies the goal, in which case we say that s a goal state
- a criterion that specifies the quality of an acceptable solution.
  - For example, any sequence of actions that gets the agent to the goal state may be acceptable, or there may be costs associated with actions and the agent may be required to find a sequence that has minimal total cost.
  - A solution that is best according to some criterion is called an **optimal solution**. We do not always need an optimal solution, for example, we may be satisfied with any solution that is within 10% of optimal.

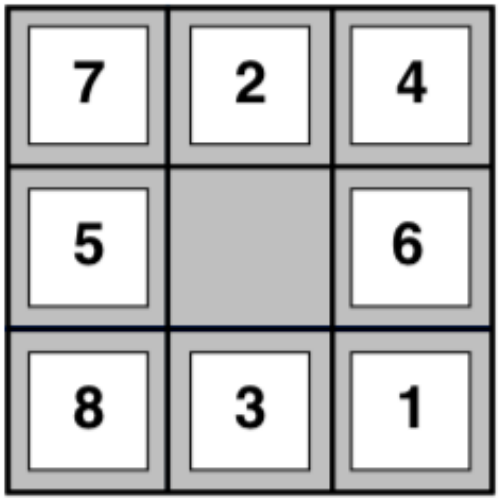
Choosing States and Actions

- Real world is absurdly complex
  → state space must be abstracted for problem solving
- (abstract) state = set of real states
- (abstract) action = complex combination of real actions
  - e.g., ”from o103 - to room 123" represents a complex set of possible routes
- (abstract) solution = set of real paths that are solutions in the real world
  - Each abstract action should be "easier" than the original problem

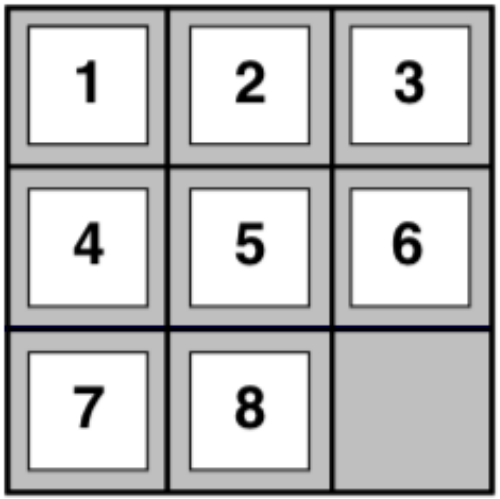
Example Problems

- Toy problems: concise exact description
- Real world problems: don’t have a single agreed description

Example: The 8-puzzle



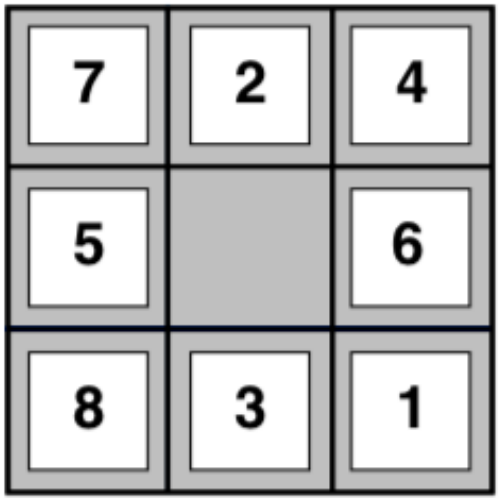
Start State



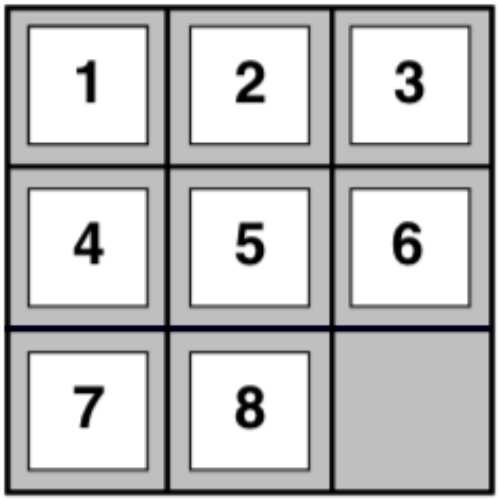
Goal State

- states?
- actions?
- goal test?
- path cost?

Example: The 8-puzzle



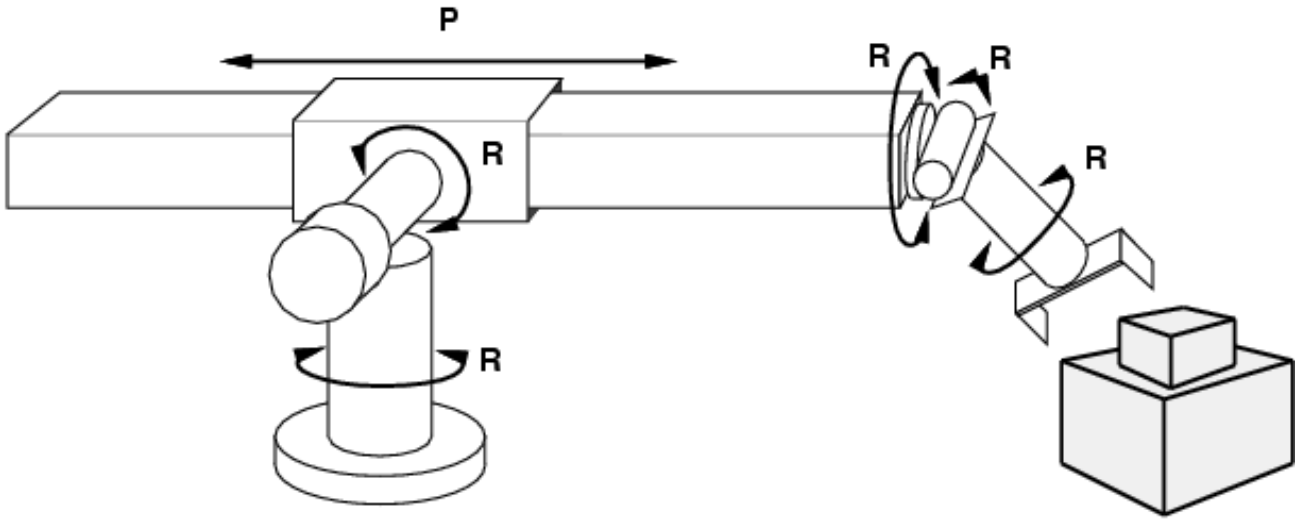
Start State



Goal State

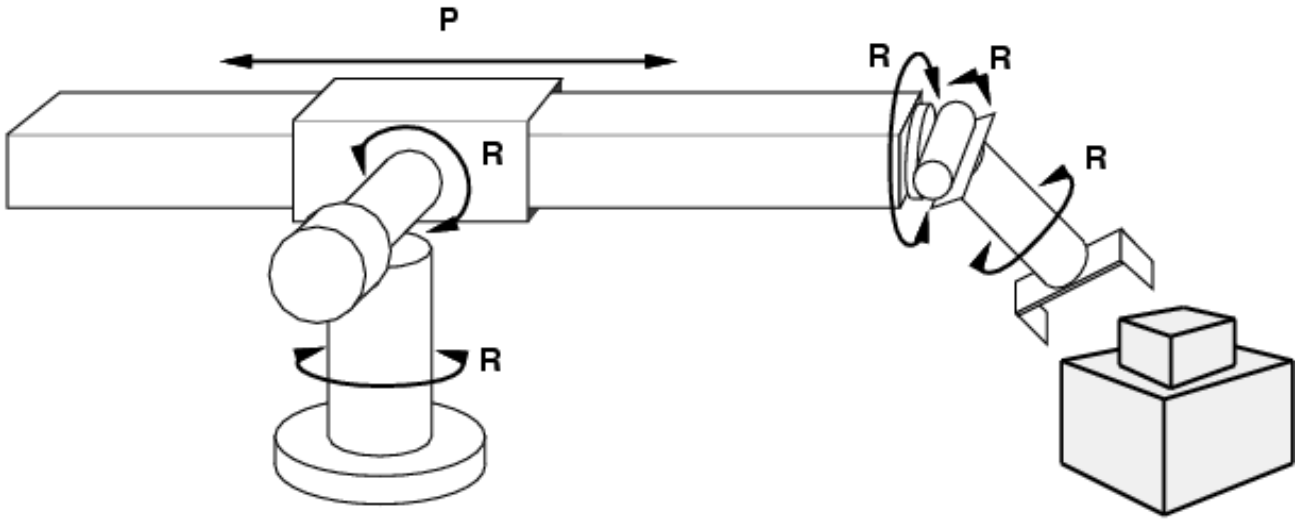
- states? integer locations of tiles (ignore intermediate positions)
- actions? move blank left, right, up, down (ignore unjamming etc.)
- goal test? = goal state (given)
- path cost? 1 per move

Example: robotic assembly



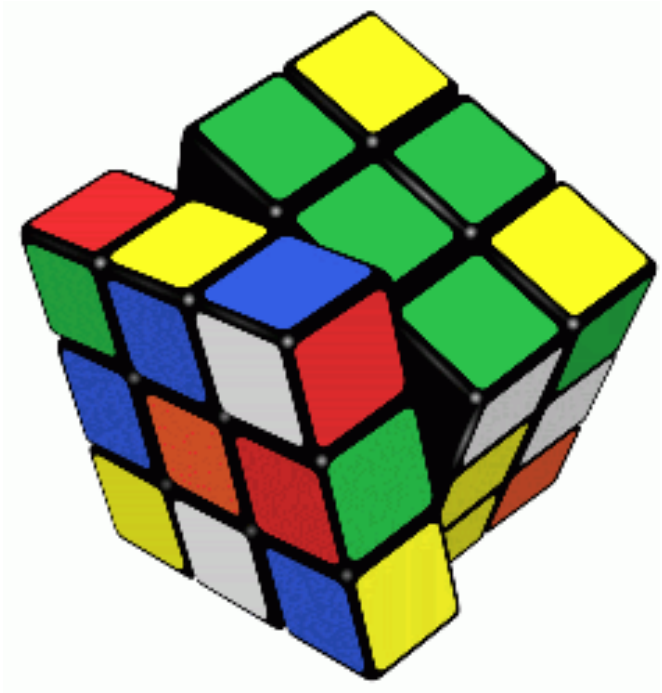
- states?
- actions?
- goal test?
- path cost?

Example: robotic assembly



- states?: real-valued coordinates of robot joint angles parts of the object to be assembled
- actions?: continuous motions of robot joints
- goal test?: complete assembly with no robot included
- path cost?: time to execute

Rubik's Cube



- states?
- actions?
- goal test?
- path cost?

Path Search Algorithms

Search: Finding state-action sequences that lead to desirable states. Search is a function

solution search(*task*)

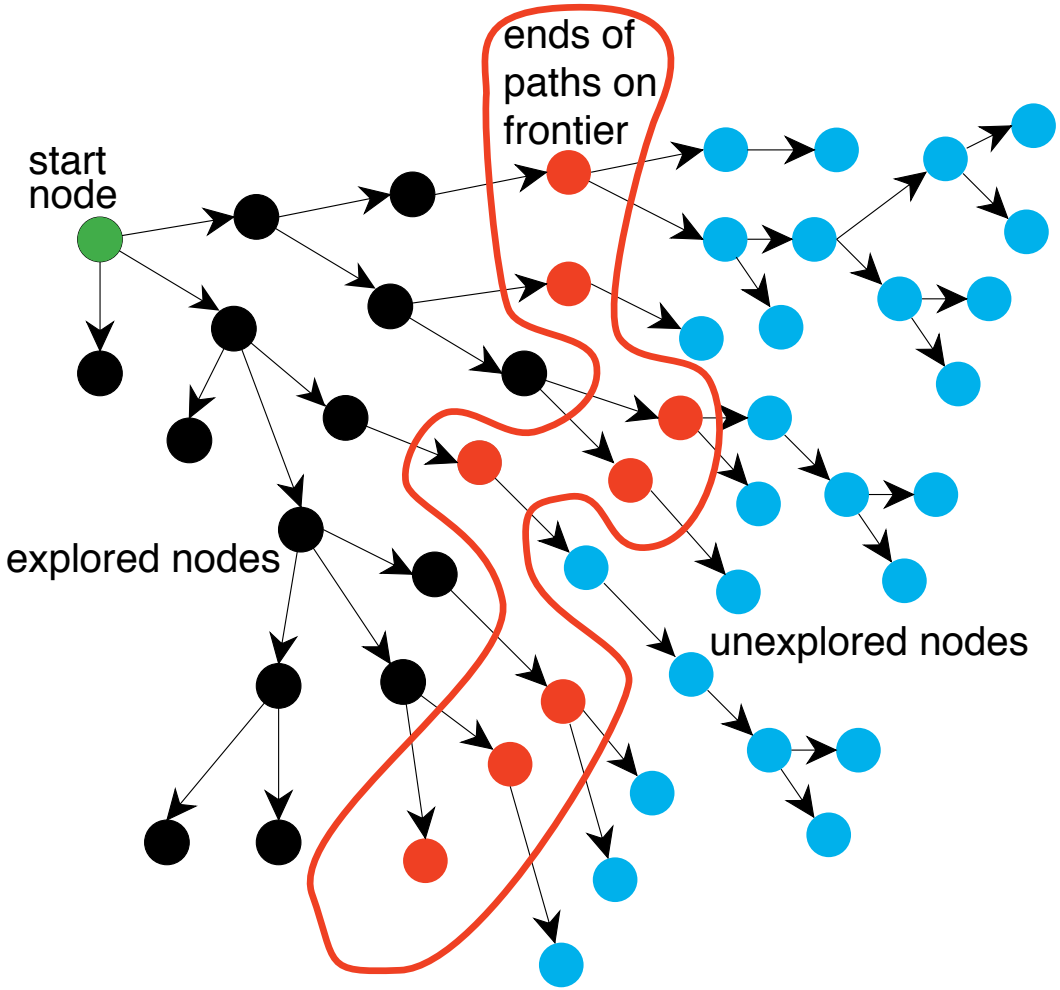
Basic idea:

Offline, simulated exploration of state space by generating successors of already-explored states (i.e. ”expanding” them)

Graph Searching

- Generic search algorithm: given a graph, start nodes, and goal nodes, incrementally explore paths from the start nodes.
- Maintain a frontier of paths from the start node that have been explored.
- As search proceeds, the frontier expands into the unexplored nodes until a goal node is encountered.
- The way in which the frontier is expanded defines the search strategy.

Problem Solving by Graph Searching



Graph Search Algorithm

The generic search algorithm:

- given a graph, a start node, and a goal predicate, is to explore paths incrementally from the start node.
- This is done by maintaining a **frontier** (or **fringe**) of paths from the start node.
  - The frontier contains all of the paths that could form initial segments of paths from the start node to a goal node (the frontier is the set of paths to the grey shaded nodes).
- Initially, the frontier contains the trivial path containing just the start node, and no arcs.
- As the search proceeds, the frontier expands into the unexplored nodes until a goal node is encountered.

Graph Search Algorithm

- Which value is selected from the frontier at each stage defines the search strategy.
- The neighbours define the graph.
- *goal* defines what is a solution.
- If more than one answer is required, the search can continue from the return.

Optimality Criteria

- Often, we don't want any solution, but the best solution or optimal solution.
- Costs on arcs give costs on paths. We want the least-cost path to a goal.

State-space Search

- flat or modular or hierarchical
- explicit states or features or individuals and relations
- static or finite stage or indefinite stage or infinite stage
- fully observable or partially observable
- deterministic or stochastic dynamics
- goals or complex preferences
- single agent or multiple agents
- knowledge is given or knowledge is learned
- perfect rationality or bounded rationality

Search Tree

- Search tree: superimposed over the state space.
- Root: search node corresponding to the initial state.
- Leaf nodes: correspond to states that have no successors in the tree because they were not expanded or generated no new nodes.

- *state space* is not the same as *search tree*
  - there are infinitely many paths!

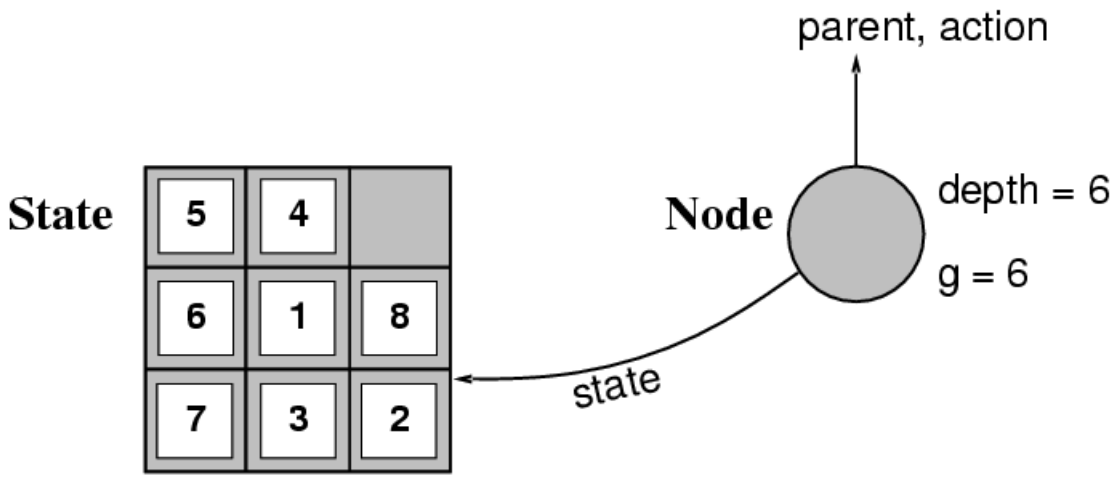
Data Structures for a Node

One possibility is to have a node data structure with five components:

1. Corresponding state
2. Parent node: the node which generated the current node.
3. Operator that was applied to generate the current node.
4. Depth: number of nodes from the root to the current node.
5. Path cost.

Implementation: states vs. nodes

- A state is a (representation of) a physical configuration
- A node is a data structure constituting part of a search tree includes parent node, children, depth, path cost $g(x)$



- Note: two different nodes can contain the same state.
- The `Expand` function creates new nodes, filling in the various fields and using the `SuccessorFn` of the problem to create the corresponding states.

Data Structures for Search Trees

- Frontier: collection of nodes waiting to be expanded
  It can be implemented as a *priority queue* with the following operations:
- `MAKE-QUEUE(ITEMS)` creates queue with given items.
- Boolean `EMPTY(QUEUE)` returns TRUE if no items in queue.
- `REMOVE FRONT(QUEUE)` removes the item at the front of the queue and returns it.
- `QUEUEING-FUNCTION(ITEMS, QUEUE)` inserts new items into the queue.

Search Strategies

- A strategy is defined by picking the order of node expansion
- Strategies are evaluated along the following dimensions:
  - completeness – does it always find a solution if one exists?
  - time complexity – number of nodes generated/expanded
  - space complexity – maximum number of nodes in memory
  - optimality – does it always find a least-cost solution?
- Time and space complexity are measured in terms of
  - b – maximum branching factor of the search tree
  - d – depth of the least-cost solution
  - m – maximum depth of the state space (may be ∞)

How Fast and How Much Memory ?

How to compare algorithms ? Two approaches:

1. Benchmarking: run both algorithms on a computer and measure speed

2. Analysis of algorithms: mathematical analysis of the algorithm

Benchmarking

- Run two algorithms on a computer and measure speed.
  - Depends on implementation, compiler, computer, data, network ...
- Measuring time
- Processor cycles
- Counting operations
- Statistical comparison, confidence intervals

Analysis of Algorithms

- $\mathrm{T}(n)$ is $O(f(n))$ means $\exists n_0, k : \forall n > n_0 \ \mathrm{T}(n) \leq kf(n)$
  - n = input size
  - $\mathrm{T}(n)$ = total number of step of the algorithm
- Independent of the implementation, compiler, ...
- Asymptotic analysis:
  For large n, an $O(n)$ algorithm is better than an $O(n^2)$ algorithm.
- $O()$ abstracts over constant factors
  - e.g. $\mathrm{T}(100 \cdot n + 1000)$ is better than $\mathrm{T}(n^2 + 1)$ only for $n > 110$.
- $O()$ notation is a good compromise between precision and ease of analysis.