Seminar: Search and Optimization

2. Search Problems & Project Topics

Florian Pommerening

Universität Basel

September 26, 2013

Classical Search Problems

Informal Description

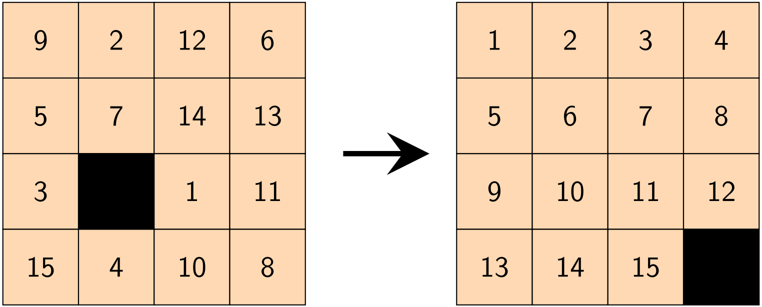
(Classical) search problems are one of the "easiest" and most important classes of AI problems.

Task of an agent:

- starting from an initial state
- apply actions
- to reach a goal state

Measure of performance: Minimize cost of actions

Motivating Example: 15-Puzzle



More examples later on

Classical Assumptions

"Classical" assumptions:

- only one agent in the environment (single agent)
- always knows the complete world state (full observability)
- only the agent can change the state (static)
- finite amount of possible states/actions (discrete)
- actions change the state deterministically

$\rightsquigarrow$ each assumption can be generalized
(not the focus of this seminar)

We omit "classical" in the following.

Formalization

State Spaces

To talk about algorithms for search problems we need a formal definition.

Definition (State Space)

A state space (or transition system) is a 6-tuple $\mathcal{S} = \langle S, A, cost, T, s_0, S_\star \rangle$ where

- S finite set of states
- A finite set of actions
- $cost : A \to \mathbb{R}_0^+$ action costs
- $T \subseteq S \times A \times S$ transition relation; deterministic in $\langle s, a \rangle$ (see next slide)
- $s_0 \in S$ initial state
- $S_\star \subseteq S$ set of goal states

State Spaces: Transitions, Determinism

Definition (Transition, deterministic)

Let $\mathcal{S} = \langle S, A, cost, T, s_0, S_\star \rangle$ be a state space.

The triples $\langle s, a, s' \rangle \in T$ are called transitions.

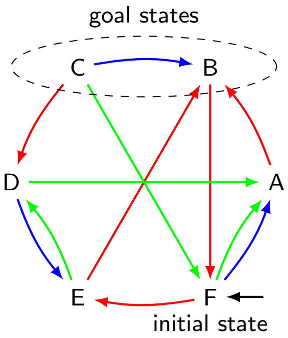
We say $\mathcal{S}$ has the transition $\langle s, a, s' \rangle$ if $\langle s, a, s' \rangle \in T$ and write $s \xrightarrow{a} s'$ ($s \rightarrow s'$, if we do not care about a).

Transitions are deterministic in $\langle s, a \rangle$: $s \xrightarrow{a} s_1$ and $s \xrightarrow{a} s_2$ with $s_1 \neq s_2$ is not allowed.

State Space: Example

State spaces are often visualized as directed graphs.

- states: nodes
- transitions: labeled edges (here: colors instead of labels)
- initial state: node marked with arrow
- goal states: marked (here: with ellipse)
- actions: edge labels
- action costs: given separately (or implicit $= 1$)
- paths to goal states correspond to solutions
- shortest paths correspond to optimal solutions



State Spaces: Terminology

We use common terminology from graph theory.

Definition (predecessor, successor, applicable action)

Let $\mathcal{S} = \langle S, A, cost, T, s_0, S_\star \rangle$ be a state space.

Let $s, s' \in S$ be states with $s \to s'$.

- s is a predecessor of s'
- s' is a successor of s

If we have $s \xrightarrow{a} s'$, action a is applicable in s.

State Spaces: Terminology

We use common terminology from graph theory.

Definition (path)

Let $\mathcal{S} = \langle S, A, cost, T, s_0, S_\star \rangle$ be a state space.

Let $s^{(0)}, \dots, s^{(n)} \in S$ be states and $\pi_1, \dots, \pi_n \in A$ actions, with $s^{(0)} \xrightarrow{\pi_1} s^{(1)}, \dots, s^{(n-1)} \xrightarrow{\pi_n} s^{(n)}$.

- $\pi = \langle \pi_1, \dots, \pi_n \rangle$ is a path from $s^{(0)}$ to $s^{(n)}$
- length of the path: $|\pi| = n$
- cost of the path: $cost(\pi) = \sum_{i=1}^{n} cost(\pi_i)$

Note:

- paths with length 0 are allowed
- sometimes the state sequence $\langle s^{(0)}, \dots, s^{(n)} \rangle$ or the sequence $\langle s^{(0)}, \pi_1, s^{(1)}, \dots, s^{(n-1)}, \pi_n, s^{(n)} \rangle$ are also called path

State Spaces: Terminology

Additional terminology:

Definition (solution, optimal, solvable, reachable, dead end)

Let $\mathcal{S} = \langle S, A, cost, T, s_0, S_\star \rangle$ be a state space.

- A path from a state $s \in S$ to a state $s_\star \in S_\star$ is a solution for/of s.
- A solution for s_0 is a solution for/of $\mathcal{S}$.
- Optimal solutions (for s) have minimal cost among all solutions (for s).
- State space $\mathcal{S}$ is solvable if a solution for $\mathcal{S}$ exists.
- State s is reachable if there is a path from s_0 to s.
- State s is a dead end if no solution for s exists.

Representation of State Spaces

Representation of State Spaces

How to get the state space into the computer?

1. As an explicit graph:
   Nodes (states) and edges (transitions) represented explicitly,
   e. g. as adjacency lists or as adjacency matrix
   - impossible for large problems (needs too much space)
   - Dijkstra for small problems: $O(|S| \log |S| + |T|)$
2. As a declarative description:
   - compact description as input
     $\rightsquigarrow$ state space exponentially larger than input
   - algorithms work directly on compact description
     (e. g. reformulation, simplification of problem)

Representation of State Spaces

How to get the state space into the computer?

3. As a black box: abstract interface for state spaces (used here)

abstract interface for state spaces

State space $\mathcal{S} = \langle S, A, cost, T, s_0, S_\star \rangle$ as black box:

- init(): creates initial state
  Returns: the state s_0
- is-goal(s): tests if state s is goal state
  Returns: **true** if $s \in S_\star$; **false** otherwise
- succ(s): lists all applicable actions and successors of s
  Returns: List of tuples $\langle a, s' \rangle$ with $s \xrightarrow{a} s'$
- cost(a): determines action cost of action a
  Returns: the non-negative number $cost(a)$

Examples

Examples for Search Problems

- "Toy problems": combinatorial puzzles (Rubik's Cube, 15-puzzle, Towers of Hanoi, . . .)
- Scheduling, e. g. in factories
- Query optimization in databases
- NPCs in computer games
- Code optimization in compilers
- Verification of soft- and hardware
- Sequence alignment in bio-informatics
- Route planning (e. g. Google Maps)
- . . .

Thousands of practical examples!

Example 1: Blocks world

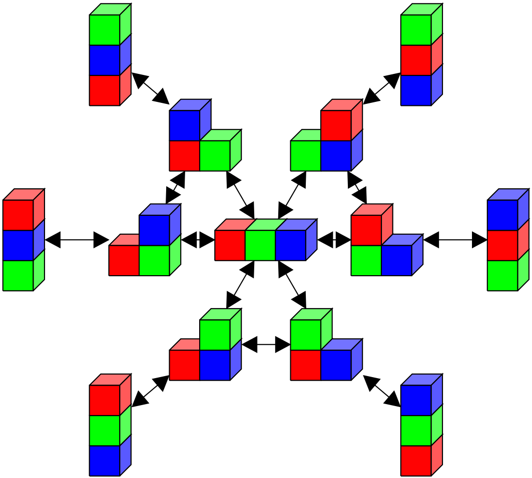
- The Blocks world is a traditional example problem in AI.

Task: blocks world

- Some colored blocks are on a table.
- They can be stacked to towers but only one block may be moved at a time.
- Our task is to reach a given goal configuration.

Blocks World with Three Blocks

(action names not shown;
initial state and goal states can be chosen for each problem)



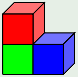
Blocks World: Formal Definition

State space $\langle S, A, cost, T, s_0, S_\star \rangle$ blocks world with n Blocks

State space: blocks world

States S:

Partitioning of $\{1, 2, \ldots, n\}$ into non-empty (ordered) sequences

Examples: $\{\langle 1, 2 \rangle, \langle 3 \rangle\} \sim$ , $\{\langle 1, 2, 3 \rangle\} \sim$ 

Initial state s_0 and goal state $S_\star$:

different choices possible, e. g.:

- $s_0 = \{\langle 1, 3 \rangle, \langle 2 \rangle\}$
- $S_\star = \{\{\langle 3, 2, 1 \rangle\}\}$

Blocks World: Formal Definition

State space $\langle S, A, cost, T, s_0, S_\star \rangle$ blocks world with n Blocks

State space: blocks world

Actions A:

- $\{move_{b,b'} \mid b, b' \in \{1, \ldots, n\}$ with $b \neq b'\}$
  - Move block b on top of block b'.
  - Both have to be topmost block of a tower.
- $\{totable_b \mid b \in \{1, \ldots, n\}\}$
  - Move block b on the table ($\rightsquigarrow$ creates new tower).
  - Has to be topmost block of a tower.

Action costs $cost$:

$cost(a) = 1$ for all actions a

Blocks World: Formal Definition

State space $\langle S, A, cost, T, s_0, S_\star \rangle$ blocks world with n Blocks

State space: blocks world

Transitions:

Example for action $a = move_{2,4}$:
Transition $s \xrightarrow{a} s'$ exists if and only if

- $s = \{\langle b_1, \dots, b_k, 2\rangle, \langle c_1, \dots, c_m, 4\rangle\} \cup X$ and
- in case $k > 0$: $s' = \{\langle b_1, \dots, b_k\rangle, \langle c_1, \dots, c_m, 4, 2\rangle\} \cup X$
- in case $k = 0$: $s' = \{\langle c_1, \dots, c_m, 4, 2\rangle\} \cup X$

Blocks World: Properties

Blocks	States
1	1
2	3
3	13
4	73
5	501
6	4051
7	37633
8	394353
9	4596553

Blocks	States
10	58941091
11	824073141
12	12470162233
13	202976401213
14	3535017524403
15	65573803186921
16	1290434218669921
17	26846616451246353
18	588633468315403843

- For every given initial state and goal state with n blocks simple algorithms can find solutions in $O(n)$ time. (How?)
- Finding optimal solutions is NP-complete (for a compact problem representation).

Example 2: Logistics

Task: logistics

- Given: Cities with locations, objects to be delivered
- Goal: Transport objects to destination locations

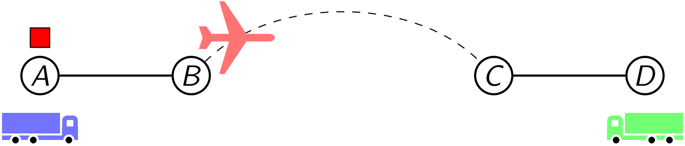
Actions: logistics

- Objects can be loaded and unloaded to trucks and airplanes.
- Trucks can drive between locations in a city.
- Airplanes can fly between airports.

Complexity of Logistics

- Finding suboptimal solutions is polynomial.
- Finding optimal solutions is NP-hard.

Logistics: Example



Goal: Transport red package from location A to location D.

1. load package in blue truck, drive to B, unload package
2. load package in airplane, fly to C, unload package
3. drive green truck to C, load package, drive to D, unload package

Example 3: Scanalyzer

- Business application (LemnaTec)
- Logistics for smart greenhouses
  - automated greenhouses with integrated imaging facilities
  - plants on conveyor belts

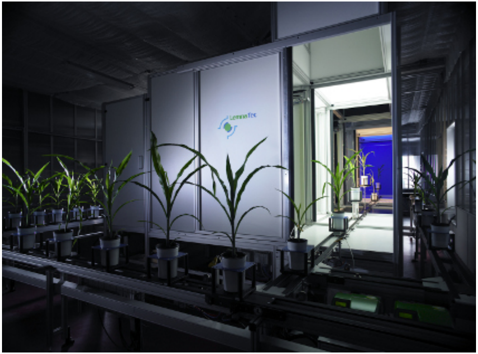


Image credit: LemnaTec

Scanalyzer

Difficulty

- Confined space
- Conveyor belts packed to capacity
- Conveyor belts only move in one direction
- Moving one plant moves others as well

Task: Scanalyzer

- Given a layout of conveyor belts
- Transport all plants through the imaging chamber
- Return every plant to its original position

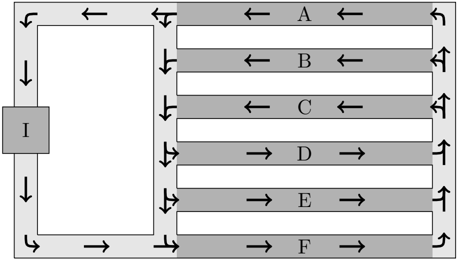
Scanalyzer: Actions

Actions: Scanalyzer

- Depend on the layout
- Rotate plant batches on two conveyor belts
- Rotate while routing through the imaging chamber

Complexity of Scanalyzer

- Depends on the layout
- Polynomial for simple, symmetric layouts



Example 4: Sokoban

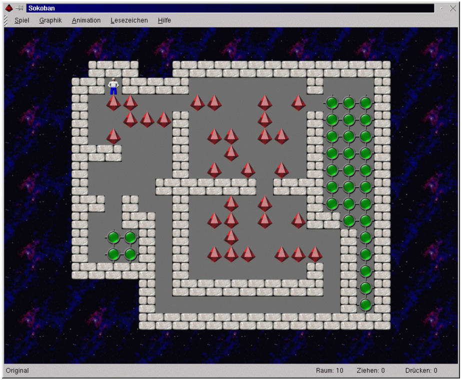


Image credit: KDE (KSokoban)

- Single player game
- Agent can push objects
- Goal: All objects are at destination locations

Sokoban

More Detailed Problem Description

- Given: Grid of locations, some locations contain objects
- Agent can push objects to free and adjacent locations
  - For example, to push an object to the right, the agent has to be located left to the object.
- Objects cannot be pulled

Complexity of Sokoban

- PSPACE-complete
- Particularly: Many dead-end states (e. g., objects in corners)

Example 5: Rovers

- Route planning and task distribution
- Multiple rovers with different capabilities
- Collect samples and take pictures of landmarks
- Transmit pictures and analysis results to lander

Image credit: NASA

Rovers

Rover capabilities

- Movement
  - different road map for each rover
- Rock/soil analysis tools
  - optional
  - limited storage capacity
- Cameras
  - optional
  - different modes (high res, color, ...)
  - have to be calibrated first
  - line of sight needed for calibration and taking pictures
- Transmission
  - only possible if lander is visible

Rovers

Task: Rovers

- Given a set of rovers with their equipment and road maps
- Collect all designated samples and pictures
- Transmit results back to lander

Complexity of Rovers

- Finding suboptimal solutions is polynomial.
- Finding optimal solutions is NP-hard.

Other Examples

- Depot
- Driverlog
- Freecell
- Woodworking
- Satellite
- Elevators
- . . .

Project

Topic

- 2-person team per topic
- Possible topics
  - Suggest your own search problem (ask us!)
  - Fallback: one of the examples
- Discuss your choice with Silvan and Jendrik next week (October 3)

Roadmap

- Phase 1: Uninformed search
  Task: Implement some uninformed solver for your domain
  Submission deadline: 21 November 2013
- Phase 2: Informed search
  Task: Implement some heuristic solver for your domain
  Submission deadline: 19 December 2013
- Phase 3: Improvements
  Task: Improve your solver (depending on results after phase 2)
  Submission deadline: 30 January 2014
- Optional presentation of results

Proceeding

Analysis

- Familiarize yourself with your domain
- How can you characterize it (e. g., size of state space, branching factor, complexity, ...)?
- What methods appear promising?
- Solve problems optimally or allow suboptimal plans?
- Consult your advisor

Proceeding

Implementation

- C++
- First strive for clean, readable code, then optimize it for efficiency
- Get feedback from you advisor frequently and already at an early stage (e. g., discuss your architecture before implementing it)

Proceeding

Evaluation

- Evaluate in each phase
- Plan your experiments: What do you want to find out? How can you accomplish this?
- As always, you are welcome to consult your advisor

Submission after each phase: Code and summary of evaluation results

Next steps

Next steps:

- Assignment of projects and advisors
- Create mercurial repository and grant your advisor write access
- Schedule kickoff meeting with your advisor