Lecture 3: How To Search Huge Graphs

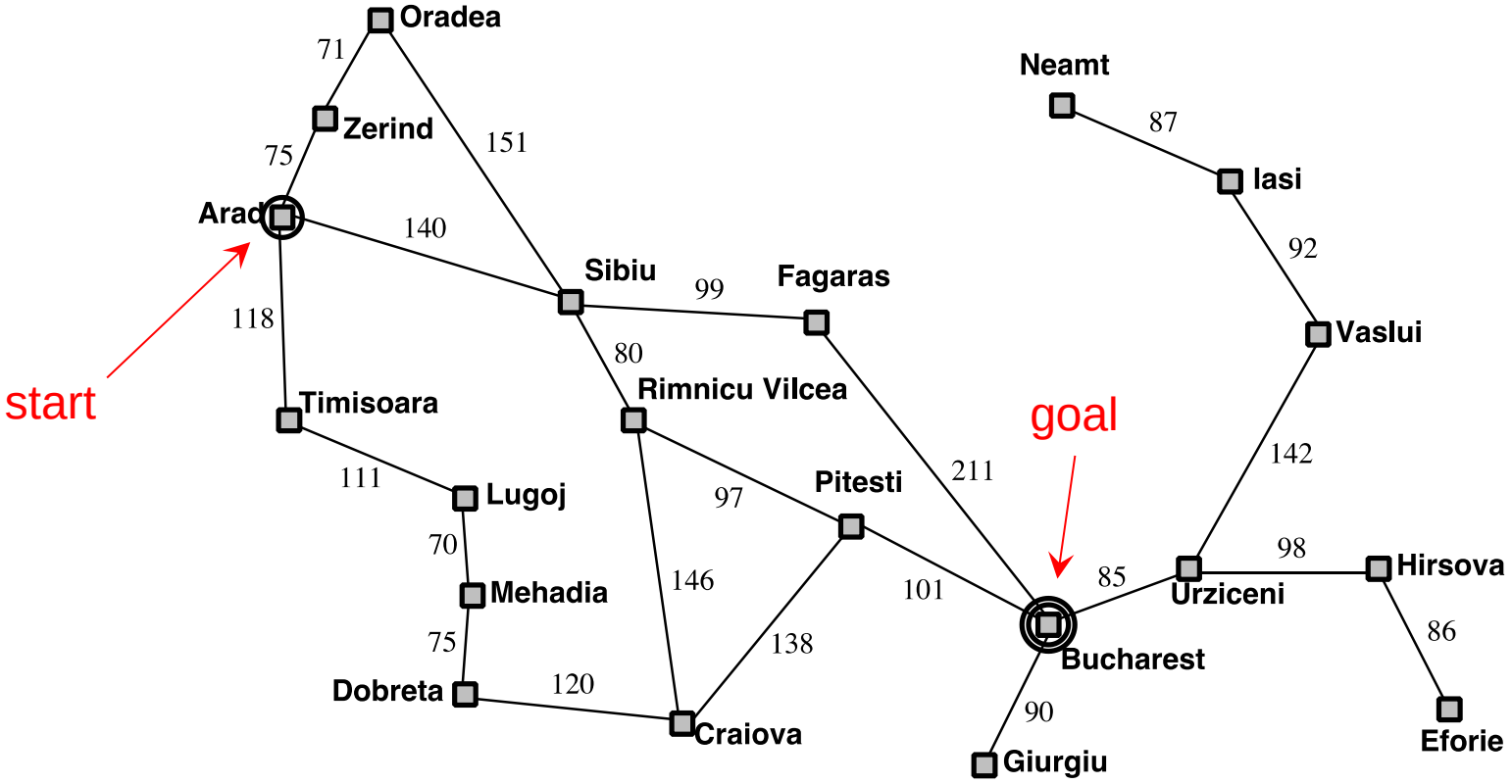
Dr John Levine

CS310 Foundations of Artificial Intelligence
January 26th 2016

University of Strathclyde Science

Explicit Graphs

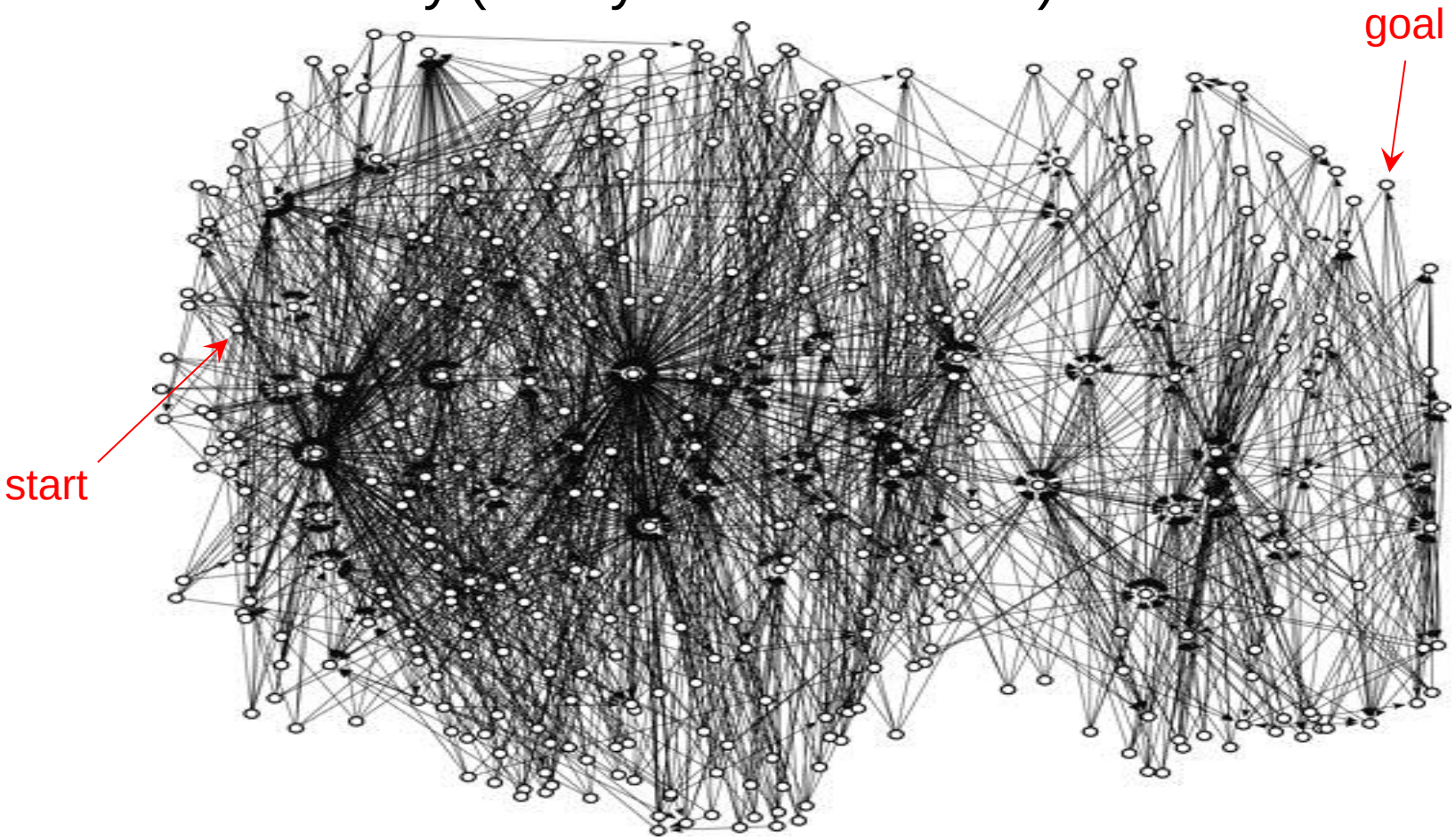
Explicit graphs are ones where you can hold the whole of the graph in memory: solve using Dijkstra's algorithm



University of Strathclyde Science

Implicit Graphs

Implicit graphs are ones where the graph is too large to hold in memory (it may even be infinite):



University of Strathclyde Science

How To Search Huge Graphs

- Don't try to hold the whole graph in memory!
- Instead, we have the start state (s_0), the goal state (g) and a function called *next-states(s)*
- *next-states(s)* computes those states which we can get to from state *s* in a single action
- To do the search, apply *next-states* to s_0 to give a list of states, then apply *next-states* to those states
- Keep doing this until state *g* appears in the list

Problem Solving using Search

Many diverse problems in AI involve search in very big implicitly-defined graphs:

- Route finding in robotics
- Blocks World planning
- Rubik's cube
- Logistics planning
- Task scheduling
- Data mining
- Machine learning

University of Strathclyde Science

What are Problems?

Each of these problems can be characterised by:

- Problem states, including the start state and the goal state
- Legal moves, or actions which transform problem states into other states
- Example: Rubik's cube
- The start state is the muddled up cube, the goal is to have the state in which all sides are the same colour and the moves are the rotations of sides of the cube

University of Strathclyde Science

Solutions

- Solutions are sequences of moves which transform the start state into the goal state
- The quality of the solution required will affect the amount of work we need to do
  - any solution will do
  - fixed amount of time, return best solution
  - near optimal solution needed
  - optimal solution needed

University of Strathclyde Science

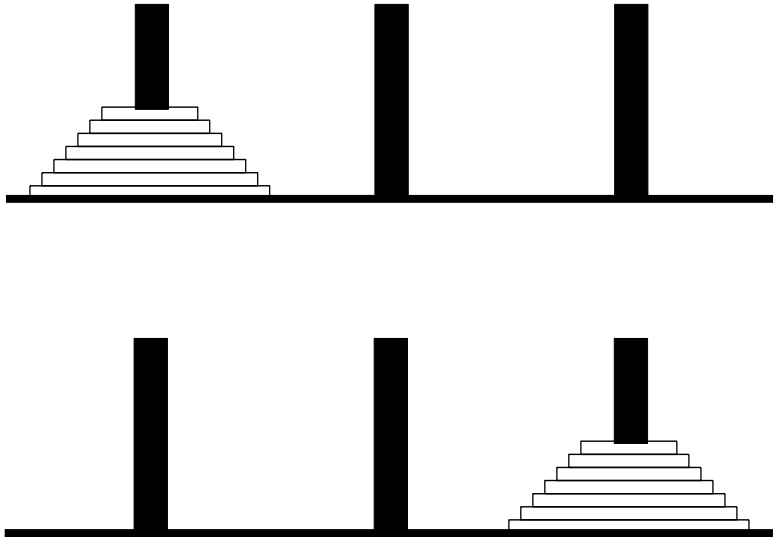
Formulating Problems

- A good formulation saves work
  - less search for the answer
- Three requirements for a search algorithm:
  - formal structures to describe the states
  - rules for manipulating them
  - identifying what constitutes a solution
- This gives us a state space representation

University of Strathclyde Science

State Space Representation

- A state space comprises
  - states: snapshots of the problem
  - operators: how to move from one state to another



Example problem: Towers of Hanoi

Only move one disc at a time

Never put a larger disc on top of a smaller one

University of Strathclyde Science

State Space Search

Problem solving using state space search consists of the following four steps:

1. Design a representation for states (including the initial state and the goal state)
2. Characterise the operators
3. Build a goal state recogniser
4. Search through the state space somehow by considering (in some or other order) the states reachable from the initial and goal states

University of Strathclyde Science

Example: Blocks World

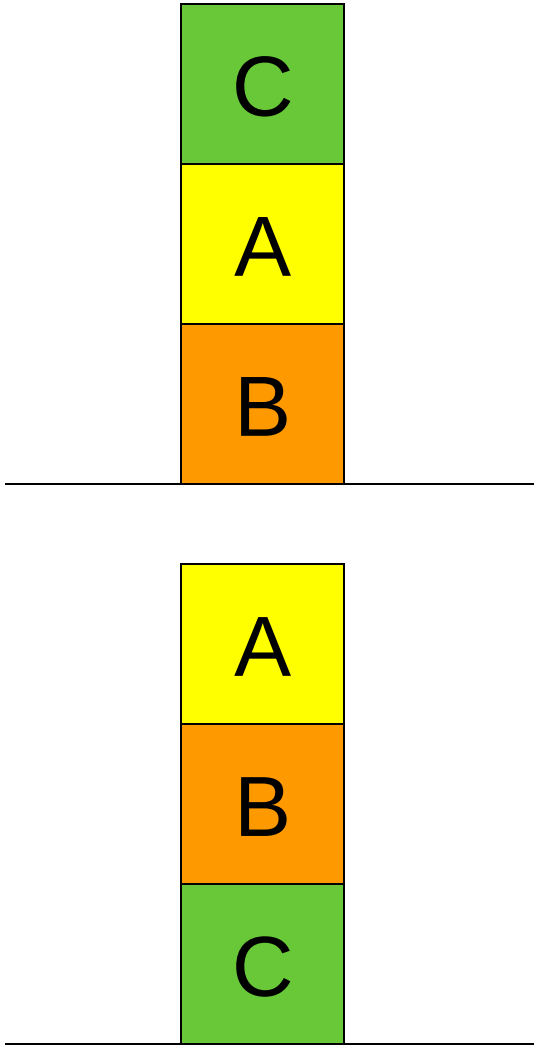
A “classic” problem in AI planning

The aim is to rearrange the blocks using the single robot arm so that the configuration in the goal state is achieved

An optimal solution performs the transformation using as few steps as possible

Any solution: linear complexity

Optimal solution: exponential complexity (NP hard)



University of Strathclyde Science

Blocks World Representation

The blocks world problem can be represented as:

- States: stacks are lists, states are sets of stacks e.g. initial state = { [a,b],[c] }
- Transitions between states can be done using a single move operator: move(x,y) picks up object x and puts it on y (which may be the table)

  { [a,b,c] } → { [b,c],[a] }
  by applying move(a,table)

  { [a],[b,c] } → { [a,b,c] }
  by applying move(a,b)

University of Strathclyde Science

Blocks World Representation

- NextStates(State) → list of legal states resulting from a single transition
  e.g. NextStates({ [a,b],[c] }) →
  { [a],[b],[c] } by applying move(a,table)
  { [b],[a,c] } by applying move(a,c)
  { [c,a,b] } by applying move(c,a)
- Goal(State) returns true if State is identical with the goal state
- Search the space: start with the start state, explore reachable states, continue until the goal state is found

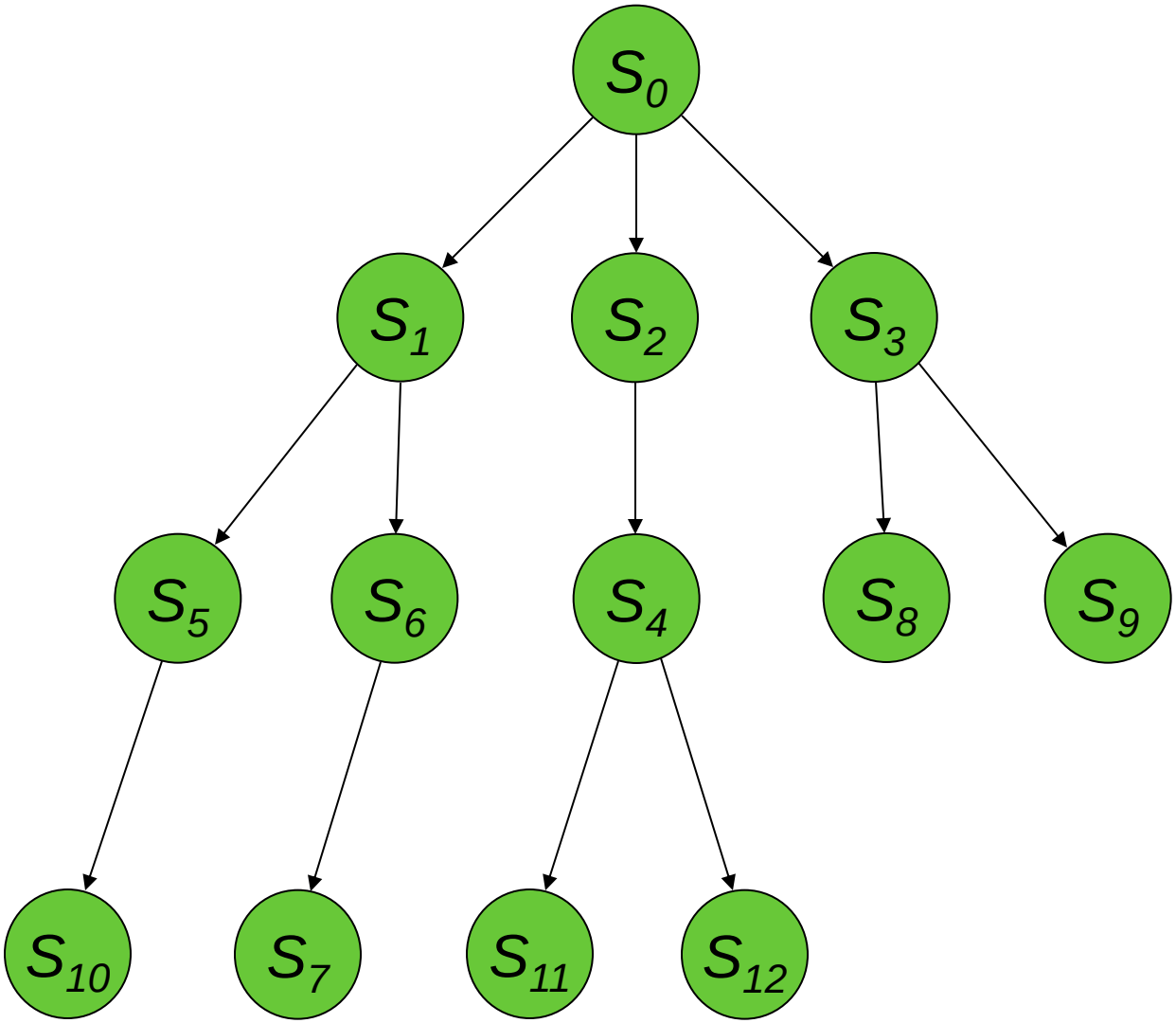
University of Strathclyde Science

Search Spaces

- The search space of a problem is implicit in its formulation
  - You search the space of your representations
- We generate the space dynamically during search (including loops, dead ends, branches)
- Operators are move generators
- We can represent the search space with trees
- Each node in the tree is a state
- When we call NextStates(S_0) → [S_1,S_2,S_3], then we say we have expanded S_0

University of Strathclyde Science

Expanding Nodes in the Search Space



University of Strathclyde Science

Gödel, Escher, Bach, Chapter 1

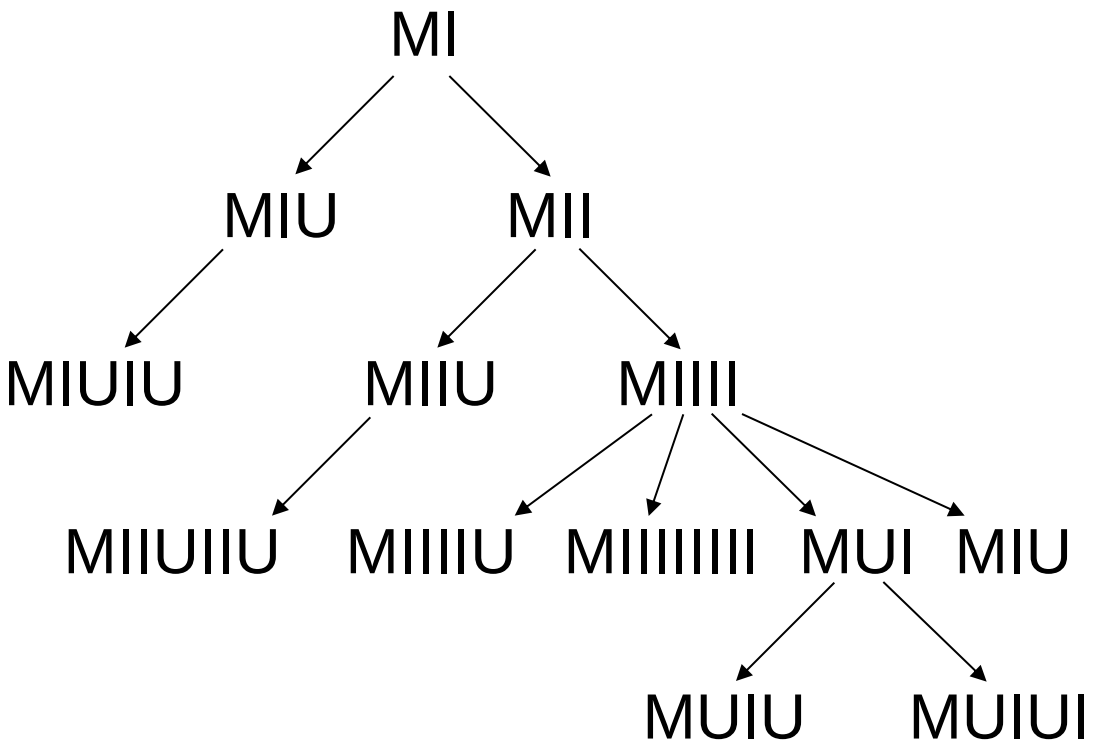
- Read pages 33-35, up to the words “Have fun.”
- Now try to make the theorem MU, starting only with the axiom MI, using these inference rules:

I. $x\mathrm{I} \rightarrow x\mathrm{IU}$
II. $\mathrm{M}x \rightarrow \mathrm{M}xx$
III. $x\mathrm{III}y \rightarrow x\mathrm{U}y$
IV. $x\mathrm{UU}y \rightarrow xy$

- Is it possible to make MU?
- Then read to the end of the chapter (page 41).

University of Strathclyde Science

A Program that Searches?



MU???

University of Strathclyde Science

Practical 1, Mon/Tue 1st/2nd February

- In Java, I want you write the *next-states(s)* function for the MIU problem
- The state is represented as a string e.g. “MIUIU”
- next-states(“MI”) returns [“MIU”, “MII”]
- I (or Damien) will check your function works as it should – this counts 3% towards your mark
- We’ll then use this function to construct a solver for the MIU problem next week…