Lecture 2: Search 1

Victor R. Lesser

CMPSCI 683

Fall 2010

Review of Lecture 1

- Dealing directly with Uncertainty in Computation
  - one of the aspects of AI that differentiates it from other sub-disciplines of CS
  - Focus of Course – Mechanisms for Dealing with different types of uncertainty
    - *Search*
    - Uncertainty Reasoning
    - Learning

Today's lecture

- Why is search the key problem-solving technique in AI?
- Formulating and solving search problems.
- Understanding and comparing several "blind" search techniques.

Overview of Material You Should Know!!

You should have read material in Chapters 1-3.7

Why Search?

Uncertainty in Agent Knowledge–Non-Determinism

- *"An agent with several immediate options of unknown value can decide what to do by first examining different possible sequences of actions that lead to states of known value and then choosing the best sequence of actions"*
- Search simulates actions in the world to some level of abstraction to understand how to proceed or to solve a problem

Real-world problems

- Signal interpretation (e.g. speech understanding)
- Theorem proving (e.g. resolution techniques)
- Combinatorial optimization (e.g. VLSI layout)
- Robot navigation (e.g. path planning)
- Factory scheduling (e.g. flexible manufacturing)
- Symbolic computation (e.g. symbolic integration)

Can we find closed form solutions to these problems?

Searching for Solutions

- The state space model provides a formal definition of a problem and what constitutes a solution to a problem.
  - States – All Possible Complete and maybe Partial (Intermediate) Solutions
  - Operators -- generate new states from existing states
- A solution is
  - a state (called a goal state) whose attributes have certain properties and maybe
  - a sequence of operators that will change the initial state to a goal state
- A solution is found by *searching* through the state space until a (*goal*) state with "specific" properties is found
  - Global and/or Local Properties (e.g., least costly operator path)
- Search involves *exploring* (*explicitly generating*) parts of the state space until a solution is found (or the entire space is explored).
  - The effective search involves finding a solution without exploring the entire state space (without generating the complete state space).

Searching through Partial Solutions

Finding a Path from the Initial state to the Goal state

- States and state spaces
- Operators - representing possible actions
- Successor function
  - state to set of {action, successor state} pairs
- Initial state and goal test
- Path cost function

Deterministic, fully observable ⇒ single-state problem --solution is a sequence

What is a Solution in this type of Search?

- A solution to a search problem is a sequence of operators that generate a path from the initial state to a goal state.
- An optimal solution is a minimal cost solution.
- Solution cost versus search cost -- which one to optimize?

Route Finding

State space representation:

- There is a state corresponding to each city;
- Initial state is the start city state;
- Goal state is the destination city state;
- Operators correspond to roads:
  - there is an operator "$city_a \rightarrow city_b$"
  - Iff there is a road from $city_a$ to $city_b$.

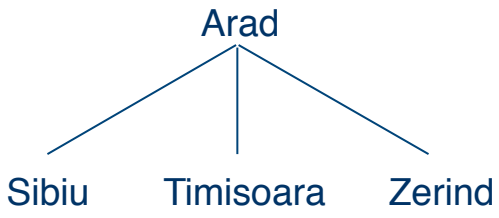
Example: Route Finding *(cont'd)*

- **Initial state is *Arad*; Goal state is *Bucharest*.**

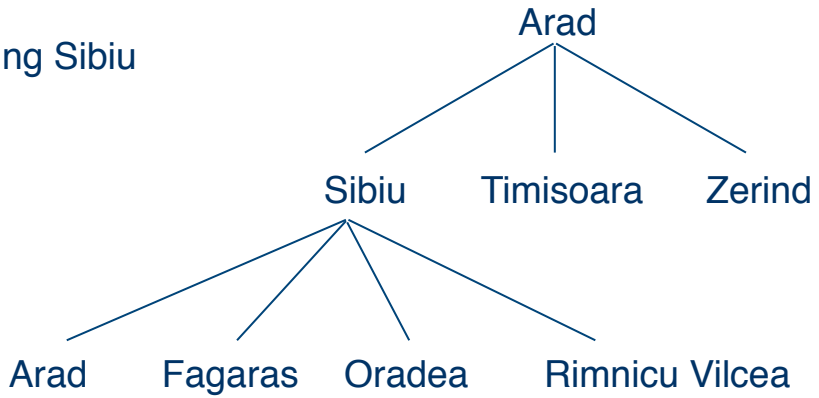
Partial search tree:

(a) The initial state

(b) After expanding Arad

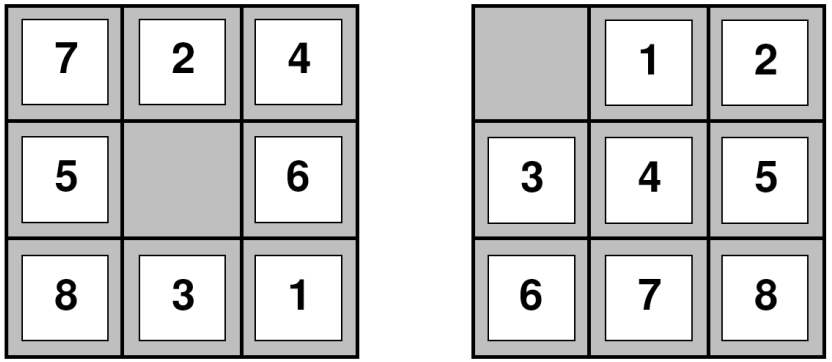


(c) After expanding Sibiu



The final search tree shows six partial solutions (open search nodes).

The 8-puzzle



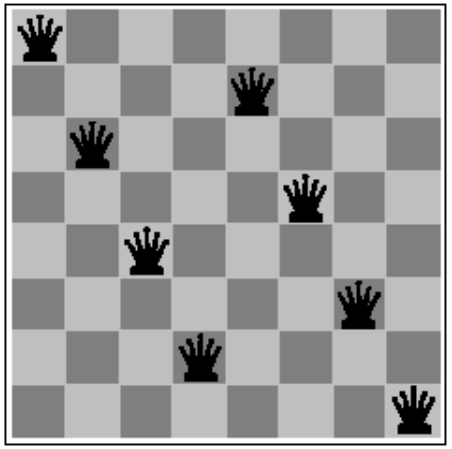
Start State Goal State

- States: integer locations of tiles
- Actions: move blank left, right, up, down
- Goal test: goal state (given)
- Path cost: 1 per move, total path cost = number of steps

Note: Solving *n*-puzzle problems optimally is NP-hard 10^{25} States for 5 x 5 board (24 puzzle)

8- Queens Problem

- **Initial State**:
  - Any arrangement of 0 to 8 queens on board.
- **Operators**:
  - Add or move a queen to any square.
- **Goal Test**:
  - 8 queens on board, none attacked.
- **Path cost**:
  - not applicable or Zero (because only the final state counts, search cost might be of interest).



Explicit solution for n ≥ 4

[Hoffman, Loessi, and Moore, 1969]

If n is even but not of the form 6k+2:

For j = 1, 2, ..., n/2 place queens on elements

(j, 2j), (n/2+j, 2j-1)

If n is even but not of the form 6k:

For j = 1, 2, ..., n/2 place queens on elements

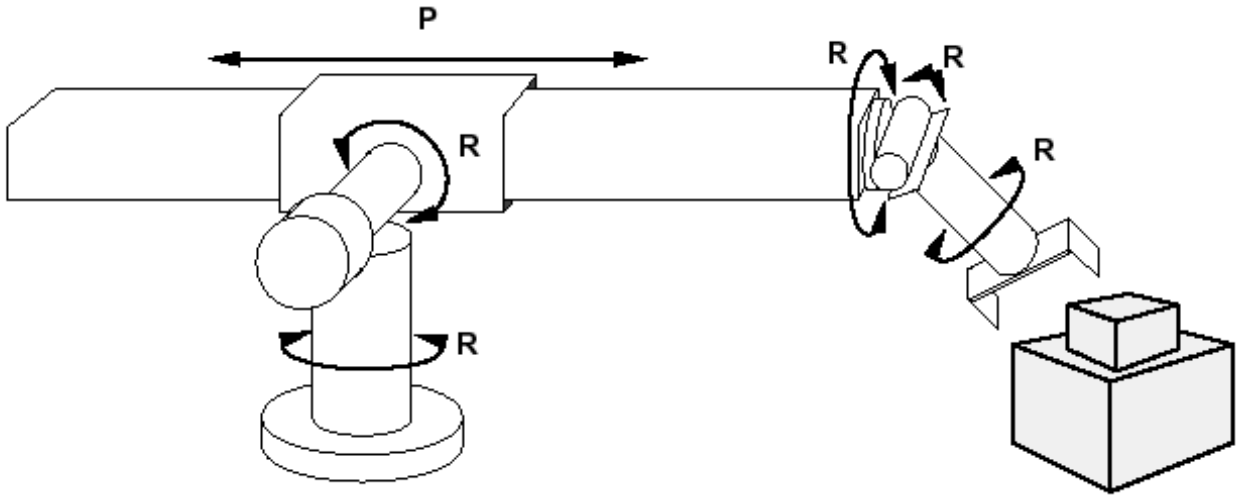
(j, 1+[(2(j-1) + n/2 - 1) mod n]), (n+1-j, n-[(2(j-1) + n/2 - 1) mod n])

If n is odd:

Use case A or B on n-1 and extend with a queen at (n,n)

Is this a good benchmark problem for testing search techniques?

Robot Assembly



- States: real-valued coordinates of robot joint angles; parts of the object to be assembled
- Actions: continuous motion of robot joints
- Goal test: complete assembly
- Path cost: time to execute

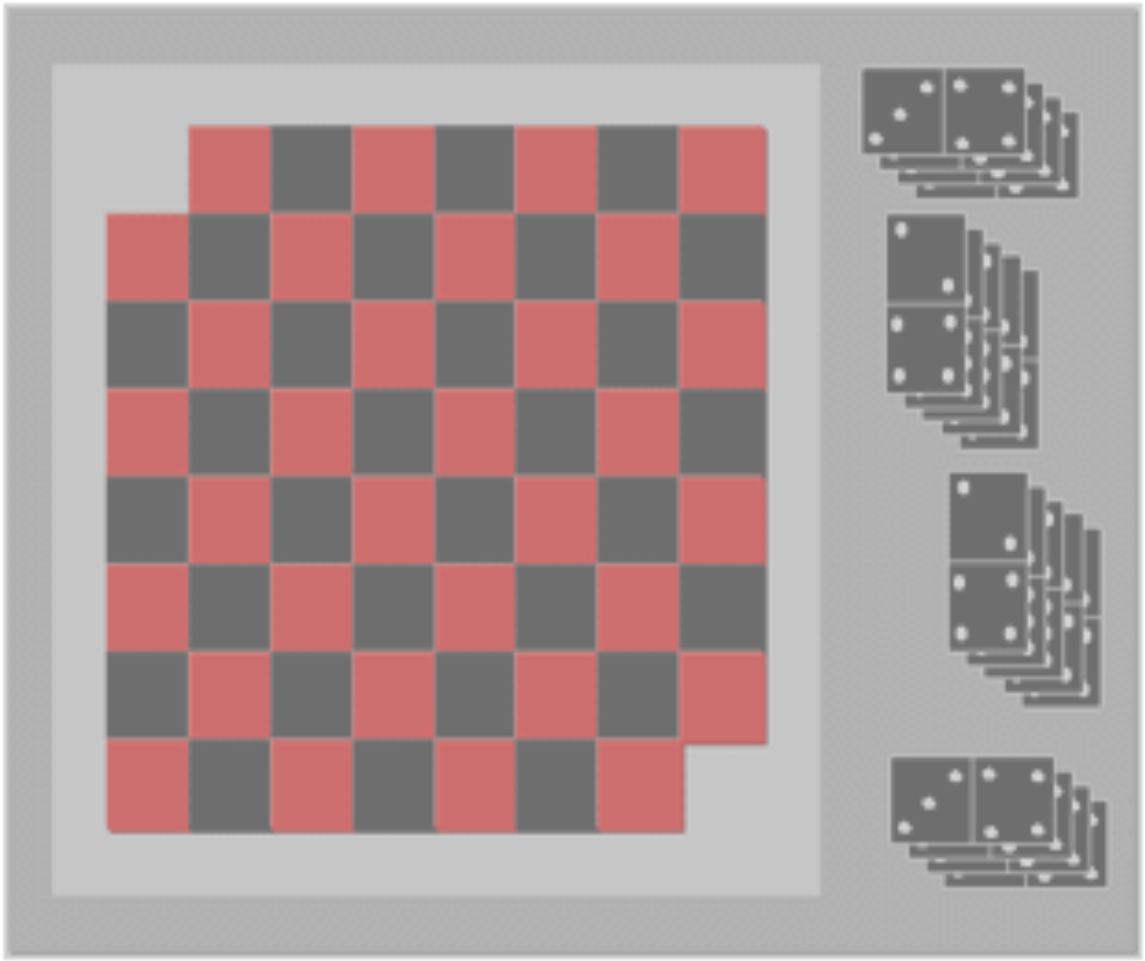
Cryptarithmetic

```
 FORTY          Solution: 29786      F=2, O=9, R=7, etc
+  TEN                     850
+  TEN                     850
 -----                   -----
 SIXTY                   31486
```

- **Initial State**:
  - a cryptarithmetic puzzle with some letters replaced by digits.
- **Operators**:
  - replace all occurrences of a letter with a non-repeating digit.
- **Goal Test**:
  - puzzle contains non-repeating digits for each different letter in the puzzle, and represents a correct sum.
- **Path cost**:
  - not applicable or 0 (because all solutions equally valid).

Mutilated Checkerboard Problem

Find a solution where the dominos exactly cover the board



Executing the mechanism:
Auctioneer's winner determination problem

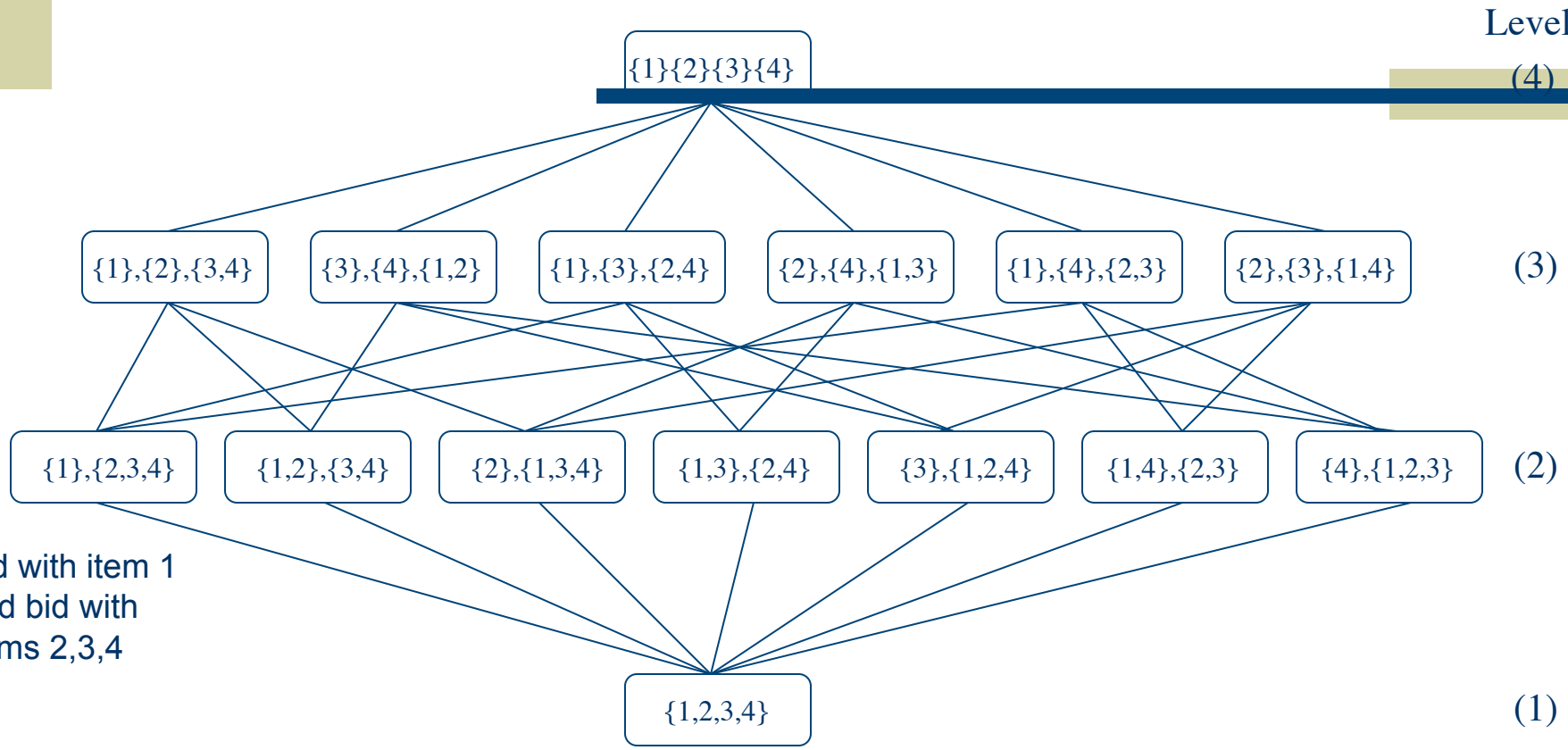
- **Set of items,** $M = \{1, 2, \ldots, \#items\}$
- **Set of bids,** $\mathcal{B} = \{B_1, B_2, \ldots, B_{\#bids}\}$
- $B_j = \langle S_j, p_j \rangle$**, where** $S_j \subseteq M$ **is a set of items and** p_j **is a price**
- $S_j \neq S_k$ **(if multiple bids concern the same set of items, all but the highest bid can be discarded by a preprocessor)**
- **Problem: Label the bids as winning** ($x_j = 1$) **or losing** ($x_j = 0$) **so as to maximize auctioneer's *revenue* such that each item is allocated to at most one bid:**

$$\max \sum_{j=1}^{\#bids} p_j x_j \quad \textbf{s.t.} \quad \sum_{j|i \in S_j} x_j \leq 1 \qquad i = 1, 2, \ldots, \#items$$

$$x_j \in \{0, 1\}$$

- **Without free disposal,** $\leq$ **becomes** $=$ **(also in generalizations)**

Space of allocations of Items in Bids



#allocations is O(#items$^{\#items}$)
check each node whether it exists as bids and if so calculate its cost – preprocess so that have only one bid for each group of items; take highest bid

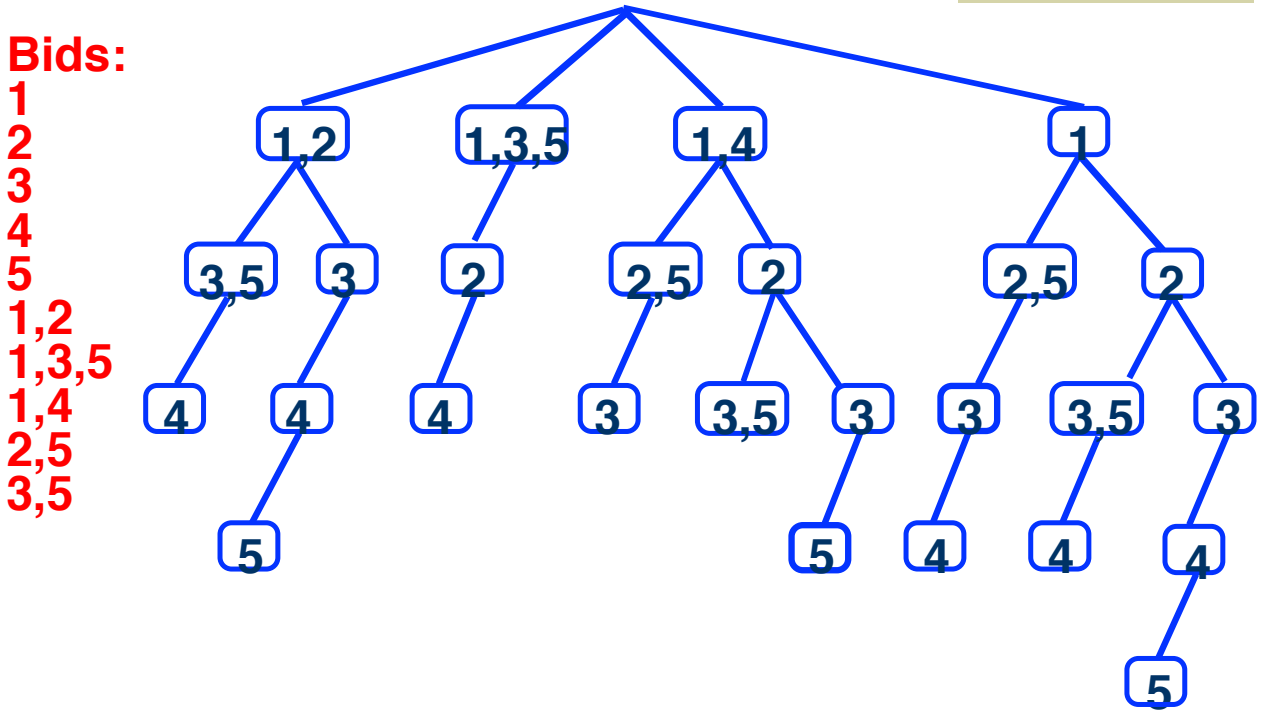
Search algorithm for Optimal / Anytime winner determination

- **Capitalizes on sparsely populated space of bids**
  - Main insight is that in practice actual bids sparsely populate the possible bids space.
- **Generates only populated parts of space of allocations**
  - Depth first search through disjoint bid-sets space until reaching either a covering bid-set or a dead end.
- **Highly optimized**
  - Keep track of best solution found to prune doomed search paths (anytime)
- **First generation algorithm scaled to hundreds of items & thousands of bids** [Sandholm IJCAI-99] -
- **Second generation algorithm** [Sandholm&Suri AAAI-00, Sandholm et al. IJCAI-01]

First generation search algorithm

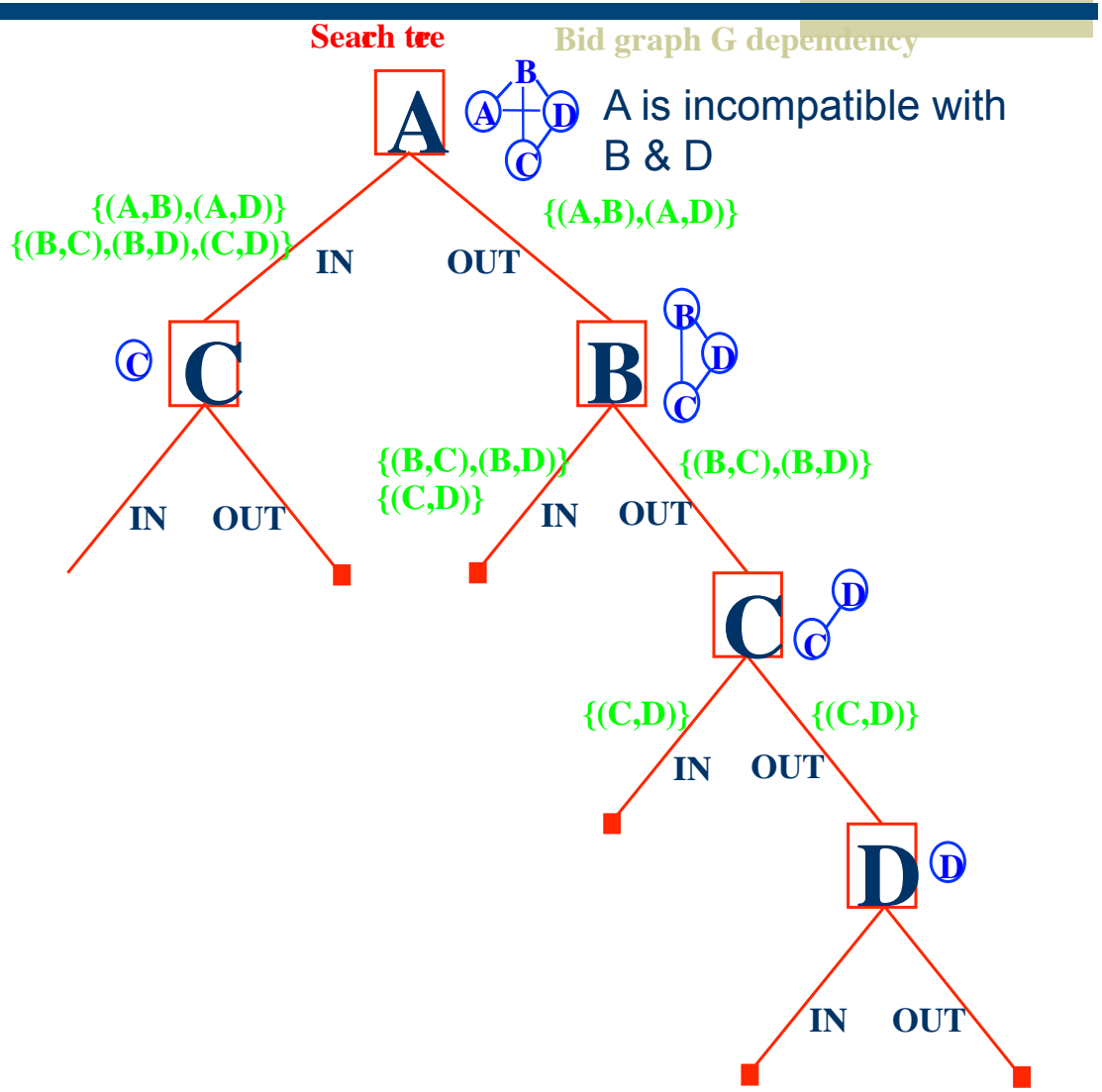
At each level of tree all the valid bids that contain the smallest numbered item not covered in bids higher in the tree

9 terminal nodes – legal bid combinations that can not be further expanded



2nd generation search algorithm: *Branching on bids*

E.g. bids
A={1,2}
B={2,3}
C={3}
D={1,3}



Review of State Space Terminology

- **State space**: a *graph* of the set of states reachable from the initial states via some operator sequence.

  (The state space is sometimes also called the *problem space* or the *search space*.)
- **Path**: a sequence of operators from one state to some other state.
- **Solution**: a *path* from an initial state to a goal state.
- **Partial solution**: a *path* from an initial state to a (non-dead-end) intermediate state.
  - Encompasses family of possible solutions
- **Goal test:** predicate that tests if a state is a goal state (goal states may be explicitly listed or specified by a property).
- **Path cost**: function that assigns a cost to a path (often denoted g).
  - *sum* of costs of the operators/actions of the path.

Searching for Solutions (*con'd.)*

- **Search tree**: tree (or graph) of states (really *nodes*) explored by the search process.
  - search tree (or search graph) is a *subgraph* of the state space.
- Search involves maintaining and incrementally extending a *set* of *partial solutions*.
  - We refer to these partial solutions as search **nodes** (nodes in the search tree).
- The process of extending a partial solution is called **expanding** a node.
  - Basically, expanding a node involves using all/? operators applicable to the latest state of the node to identify reachable states and so generate new partial solutions (nodes).
  - It is common to refer to nodes by their latest states, but a node really represents a partial solution (operator sequence).

Problem Formalization Issues

- Key issues in defining states:
  - which objects/relations to represent;
  - which configurations of objects/relations need to be mapped into separate states.
- Key issues in defining operators:
  - may have to make explicit, unstated assumptions in the problem description;
  - how state-specific/general should operators be;
  - how much domain-specific knowledge should be “compiled” into the operators.
- Developing an effective state space representation of a problem is choosing an appropriate abstraction.
  - Without abstraction, agents would be swamped by the details of the real-world.

Next Lecture

- Finish off Introductory Material on Search?
- Brief Review of Blind Search
- How to use heuristics (domain knowledge) in order to accelerate search?
- Reading: Sections 4.1-4.2.