Artificial Intelligence

Blai Bonet

Universidad Simón Bolívar, Caracas, Venezuela

AND/OR search

Motivation: 12 coins

Consider the following problem:

There are 12 coins one of which is **counterfeit** *with a* **weight** *that is different from the others. You need to determine which coin is counterfeit and whether it is heavier or lighter*

You are given a **balance scale** to find the counterfeit coin and determine its relative weight in a minimum number of weights

How do you solve it?

How many weights are needed?

[Image from `http://exchange.smarttech.com`]

Decomposition in 12-coin problem

Previous problem is example of a **decomposition task** in which the problem needs to be decomposed into **subproblems**

Represent **knowledge about coins** by tuple (s, ls, hs, u) where:

– $s + ls + hs + u = 12$

– s is number of coins **known** to be of standard weight

– ls is number of coins known to be lighter or of standard weight

– hs is number of coins known to be heavier or standard weight

– u is number of coins known to be of completely unknown weight

Each weigh on the balance then produces one or **more outcomes**

The problem contains non-deterministic actions

Decomposition in 12-coin problem

States for 12-coin of the form (s, ls, hs, u)

Initial state $(0, 0, 0, 12)$ reflects **complete ignorance** on the coins

Action that puts 4 unknown coins on each plate may produce:

- $(8, 0, 0, 4)$ if the plates perfectly level on the balance
- $(4, 4, 4, 0)$ if the plates don't level on the balance

The solution is a **strategy** that tells how to weigh the coins for each possible outcome of the actions

The 12 coin problems can be solved with 3 weighs!

Solution form

Solutions for AND/OR models are **strategies** rather than **linear sequences of actions**

Strategies can be compared on different grounds (optimality criteria is not unique)

Model for 12 coins is **acyclic** but there are AND/OR problems with **cyclic** state spaces

Different **solution concepts** define the set of valid solutions

Intuition for AND/OR graphs

Depending on the task, nodes in AND/OR graphs may represent:

- Subproblems to be solved
- Current state of the model
- Knowledge about current state

AND/OR graphs are used to represent problems in which tasks can be decomposed into different substasks on problems in which actions may have **non-deterministic effects**

General AND/OR model

Formally, an AND/OR graph is a **directed hypegraph**

Each edge has a source vertex and $k \geq 1$ destination vertices; edges are called k-connectors

If all edges are 1-connectors, the AND/OR graph is a **regular graph**

Each k-connector $C = (n_0, \{n_1, \ldots, n_k\})$ has cost $cost(C)$. We say:

- n_0 is a parent of each n_i
- each n_i is a child of n_0
- C leaves n_0 and enters each n_i

General AND/OR model

Vertices without children are **terminal vertices** and without parents **root vertices**

If every vertex has at most one parent and there is just one root, the graph is an **AND/OR tree**

If there is no sequence of vertices $(n_0, n_1, \ldots, n_k)$ such that n_i is parent of n_{i+1}, $0 \leq i < k$, and $n_0 = n_k$, the graph is **acyclic**

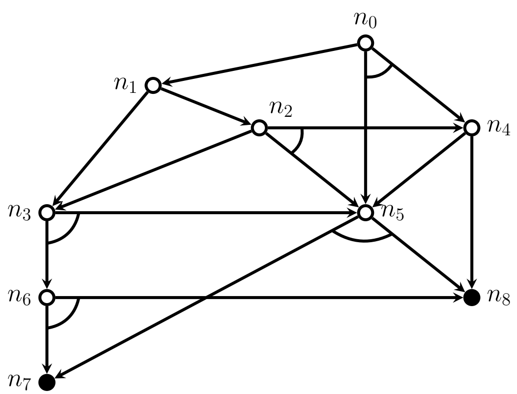
General AND/OR model

Formally, and AND/OR graph is tuple $(V, E, T, n_0, cost)$ where:

- V is a set of vertices
- E is a set of connectors
- $T \subseteq V$ is a set of terminal vertices
- $n_0 \in V$ is an initial vertex
- $cost : T \cup E \to \mathbb{R}$ is the cost function

Example of AND/OR model

- Vertices $V = \{n_0, n_1, \ldots, n_8\}$
- Terminals $T = \{n_7, n_8\}$
- Edges: $E = \{(n_0, \{n_1\}), (n_0, \{n_4, n_5\}), (n_1, \{n_2\}), (n_1, \{n_3\}), (n_2, \{n_3\}), (n_2, \{n_4, n_5\}), (n_3, \{n_5, n_6\}), (n_4, \{n_5\}), (n_4, \{n_8\}), (n_5, \{n_7, n_8\}), (n_6, \{n_7, n_8\})\}$



Solutions

Let $G = (V, E, T, n_0, cost)$ be AND/OR model

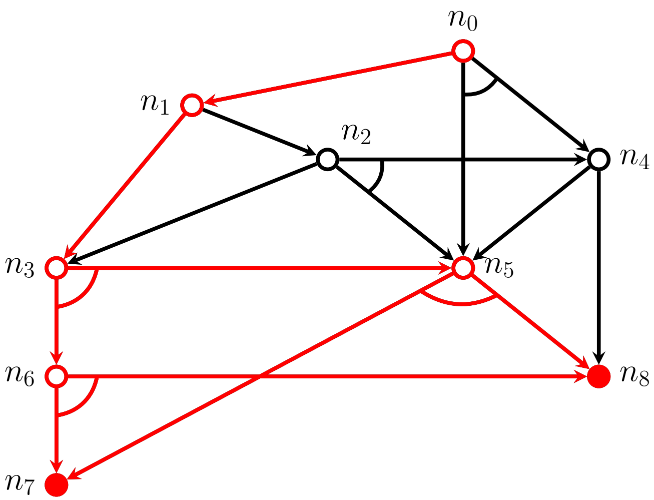
A **solution for vertex n** is subgraph $S = (V', E', T', n, cost')$:

- $V' \subseteq V$, $E' \subseteq E$, and $cost'$ is $cost$ restricted to $T' \cup E'$
- each terminal vertex in S belongs to T (i.e. $T' \subseteq T$)
- for each n in $V' \setminus T$, there is **exactly one connector** in E' that leaves n

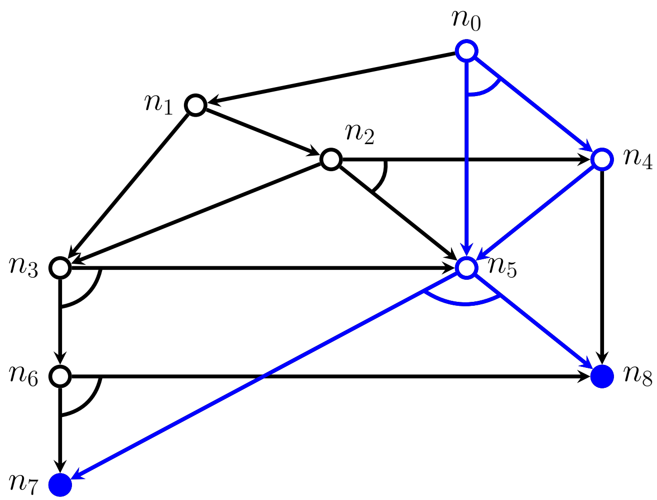
A **solution for G** is a solution for vertex n_0

Remark: if all connectors are 1-connectors, a solution S is a path in G from vertex n to some vertex in T

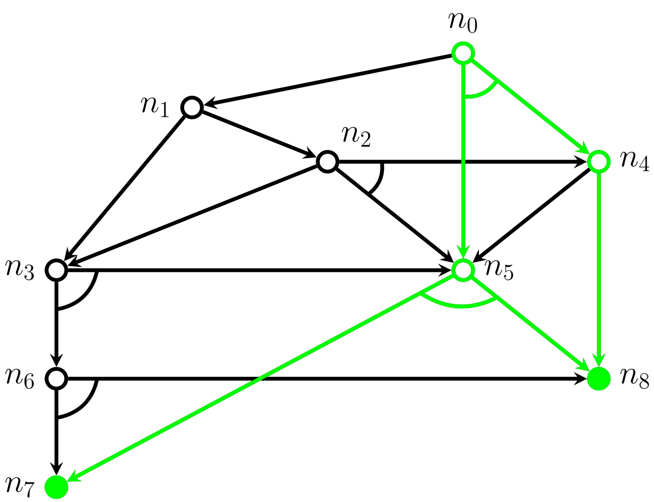
Examples of solutions



Examples of solutions



Examples of solutions



Costs for acyclic solutions

Let $G = (V, E, T, n_0, cost)$ be AND/OR model

Let $S = (V', E', T', n, cost')$ be **acyclic solution** for vertex n

We define $cost(n', S)$ for $n' \in V'$ **inductively**:

– for **terminal** vertices $n' \in T'$: $cost(n', S) \ = \ cost'(n')$

– for **non-terminal** vertices $n' \in V' \setminus T'$:

$$cost(n', S) \ = \ cost'(C) + \sum_{i=1}^{k} cost(n_i, S)$$

where $C = (n', \{n_1, \ldots, n_k\})$ is **unique** connector in E' leaving n'

Finally, $cost(S)$ is defined as $cost(n, S)$

AO* algorithm

AO* is a best-first algorithm for finding **optimal** solutions in **implicit** and **acyclic** AND/OR graphs

AO* maintains the best **partial solution** seen so far until it becomes a complete solution

Like A*, AO* constructs an explicit graph as the implicit graph is explored; the explicit graph is called the "explicated graph"

AO* uses **heuristic** h that is assumed to be admissible and consistent:

- for every terminal vertex $n \in T$, $h(n) = cost(n)$
- for every non-terminal vertex $n \in V \setminus T$, and every connector $C = (n, \{n_1, n_2, \ldots, n_k\})$ that leaves n:

$$h(n) \ \leq \ cost(C) + \textstyle\sum_{i=1}^{k} h(n_i)$$

AO*: pseudocode

1. Make explicit graph GE with only n_0; associate cost $q(n_0) = h(n_0)$
2. While n_0 is not marked as SOLVED do:
   - 2.1 Traverse best partial solution S in GE by following **marked connectors** at each vertex. (Connectors get marked below)
   - 2.2 **Select vertex** n in S that is leaf (tip) and isn't SOLVED
   - 2.3 **Expand** n**.** Add all successors n' to GE. For each child n', associate cost $q(n') = h(n')$ and marked as SOLVED if n' is terminal
   - 2.4 Make set $R = \{n\}$ of vertices to revise
   - 2.5 While $R \neq \emptyset$ do:
     - 2.5.1 Select (and remove) vertex $m \in R$ that has no descendant in R. (It can be done since graph is acyclic)
     - 2.5.2 **Revise cost** $q(m)$ associated with m (see next slide)
     - 2.5.3 If m is marked as SOLVED or its cost $q(m)$ changes, add to R all parents of m through **marked connectors**

Revise cost in AO*

Consider vertex m in R such that m has **no descendant** in R

To revise cost of vertex m:

- If m is terminal, marked it as SOLVED and terminate
- For each connector $C = (m, \{n_1, n_2, \ldots, n_k\})$ that leaves m, compute $q(C) = cost(C) + \sum_{i=1}^{k} q(n_i)$. (The values $q(n_i)$ were computed in this interation (of outer loop) or previous iteration of this loop)
- Select connector C^* with minimum q-value. Assign $q(m) = q(C^*)$. Mark connector C^* and erase marks on any other connector leaving m
- If all vertices "entered" by C^* are SOLVED, mark m as SOLVED
- If no connector leaves m, assign $q(m)$ a very high cost denoting that no solution exists below m

Example of AO*

Consider previous example and let cost of k-connector be k

Use heuristic h given by:

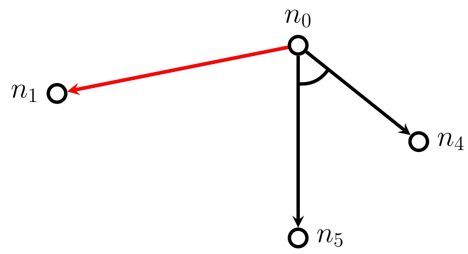
- $h(n_0) = 0$
- $h(n_1) = 2$
- $h(n_2) = h(n_3) = 4$
- $h(n_4) = h(n_5) = 1$
- $h(n_6) = 2$
- $h(n_7) = h(n_8) = 0$

Terminal costs equal to 0

Example of AO*

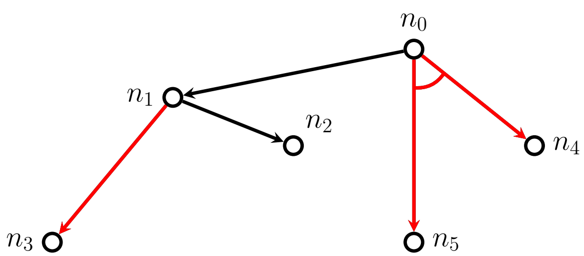
n	$q(n)$
n_0	0

Example of AO*



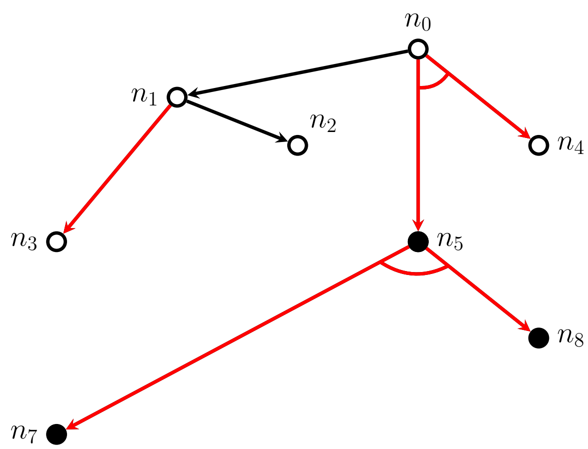
n	$q(n)$
n_0	3
n_1	2
n_4	1
n_5	1

Example of AO*



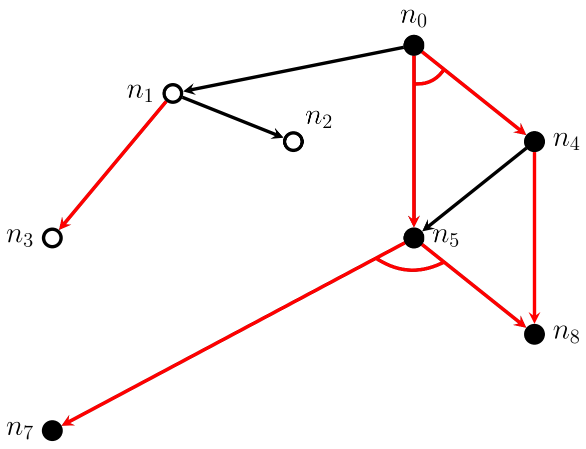
n	$q(n)$
n_0	4
n_1	5
n_2	4
n_3	4
n_4	1
n_5	1

Example of AO*



n	$q(n)$
n_0	5
n_1	5
n_2	4
n_3	4
n_4	1
n_5	2
n_7	0
n_8	0

Example of AO*



n	$q(n)$
n_0	5
n_1	5
n_2	4
n_3	4
n_4	1
n_5	2
n_7	0
n_8	0

Summary

- 12-coin problem
- General AND/OR model and solutions
- AO* algorithm