Or-Tree-based Search

CPSC 433: Artificial Intelligence
Fall 2024

Jonathan Hudson, Ph.D.
Assistant Professor (Teaching)
Department of Computer Science
University of Calgary

August 8, 2024

UNIVERSITY OF CALGARY

Or-tree-based Search

Basic Idea:

1. If every solution is okay, represent the different possibilities that might lead to a solution in the search state (as successors of a node)

Examples for solution possibilities:

- The different actions a robot can do
- The different instantiations for a variable

- Backtracking is messy so throw it away!
- Not used for optimization!
- Good for finding one valid solution (like hard constraint satisfaction), but unlike set based search designed so we can keep a history so that we don't repeat steps when one path of search fails, **less space than and-tree generally**

Model

Formal Definitions: Search Model

Or-tree-based Search Model

$A_V = (S_V, T_V)$

$Prob$ set of problem descriptions

$\boldsymbol{Altern} \subseteq Prob^+$ alternatives relation (**1 option per unique (pr,?) unlike Div**)

$S_V \subseteq \boldsymbol{Otree}$ set of possible states, is subset tree structures

where $\boldsymbol{Otree}$ is recursively defined by

$(pr, sol) \in \boldsymbol{Otree}$ for

$pr \in Prob, sol \in \{yes, ?, \boldsymbol{no}\}$

$(pr, sol, b_1, \ldots, b_n) \in \boldsymbol{Otree}$ for

$pr \in Prob, sol \in \{yes, ?, \boldsymbol{no}\}, b_i \in \boldsymbol{Otree}$

Formal Definitions: Search Model

Or-tree-based Search Model

$A_V = (S_V, T_V)$

$Prob$ set of problem descriptions

$\boldsymbol{Altern} \subseteq Prob^+$ alternatives relation

$T_V \subseteq S_V \times S_V$ transitions between states, but more specifically

$T_V = \{(s_1, s_2) \mid s_1, s_2 \in S_V$ and $Erw_V(s_1, s_2)$ ~~or $Erw_V^*(s_1, s_2)$~~$\}$

UNIVERSITY OF CALGARY

Less formally: Search Model

- The search model looks very similar to and-trees. Only differences:
  1. we can model that an alternative (subproblem) is unsolvable (sol-entry no)
  2. relation *Altern* instead of *Div*
  3. no backtracking

- The search control only has to compare the leafs of the tree and the (theoretically) one transition that has the problem of the leaf as the problem to work on

Extension function (tree expansion ~~and contraction~~)

Formal Definitions: Erw (Extension function)

$Erw_\vee$ is a relation on ***Otree*** defined by

- $Erw_\vee((pr,?),(pr,yes))$ if pr is solved
- $\boldsymbol{Erw_\vee((pr,?),(pr,no))}$ if pr is unsolvable
- $Erw_\vee((pr,?),(pr,?,(pr_1,?),\ldots,(pr_n,?)))$
  if $Altern(pr,pr_1,\ldots,pr_n)$ holds
- $Erw_\vee((pr,?,b_1,\ldots,b_n),(pr,?,b_1',\ldots,b_n'))$
  if for an i: $Erw_\vee(b_i,b_i')$ and $b_j = b_j'$ for $i \neq j$

UNIVERSITY OF CALGARY

Formal Definitions: Erw (Extension function)

$Erw_\vee$ is a relation on ***Otree*** defined by

- $Erw_\vee((pr, ?), (pr, yes))$ **leaf node is answer**
- $\boldsymbol{Erw_\vee((pr, ?), (pr, no))}$ **leaf node is not answer**
- $Erw_\vee((pr, ?), (pr, ?, (pr_1, ?), \ldots, (pr_n, ?)))$ **leaf expansion**

- $Erw_\vee((pr, ?, b_1, \ldots, b_n), (pr, ?, b_1', \ldots, b_n'))$ **allow above leaf rules to apply to more than root of tree**

Process

Formal Definitions: Search Process

Or-tree-based Search Process

$P_{\vee} = (A_{\vee}, Env, K_{\vee})$

Nothing new from And Tree

What is selected is the leaf to expand.

Instance

Formal Definitions: Search Instance

Or-tree-based Search Instance

$Ins_{\vee} = (s_0, G_{\vee})$

If the given problem to solve is pr, then we have

- $s_0 = (pr, ?)$
- $G_{\vee}(s) = \boldsymbol{yes}$, if and only if
  - $s = (pr', \boldsymbol{yes})$ or
  - $s = (pr', ?, b_1, \dots, b_n), G_{\vee}(b_i) = \boldsymbol{yes}$ for an i or
  - All leafs of s have either the sol-entry no or cannot be processed using $Altern$

UNIVERSITY OF CALGARY

Formal Definitions: Search Instance

Or-tree-based Search Instance

$$Ins_{\vee} = (s_0, G_{\vee})$$

If the given problem to solve is pr, then we have

- $s_0 = (pr, ?)$
- $G_{\vee}(s) = \boldsymbol{yes}$, if and only if
  - The root is **yes**
  - A leaf has a **yes** in it
  - We have tried all possibilities and they are all **no** or we have no options left to try

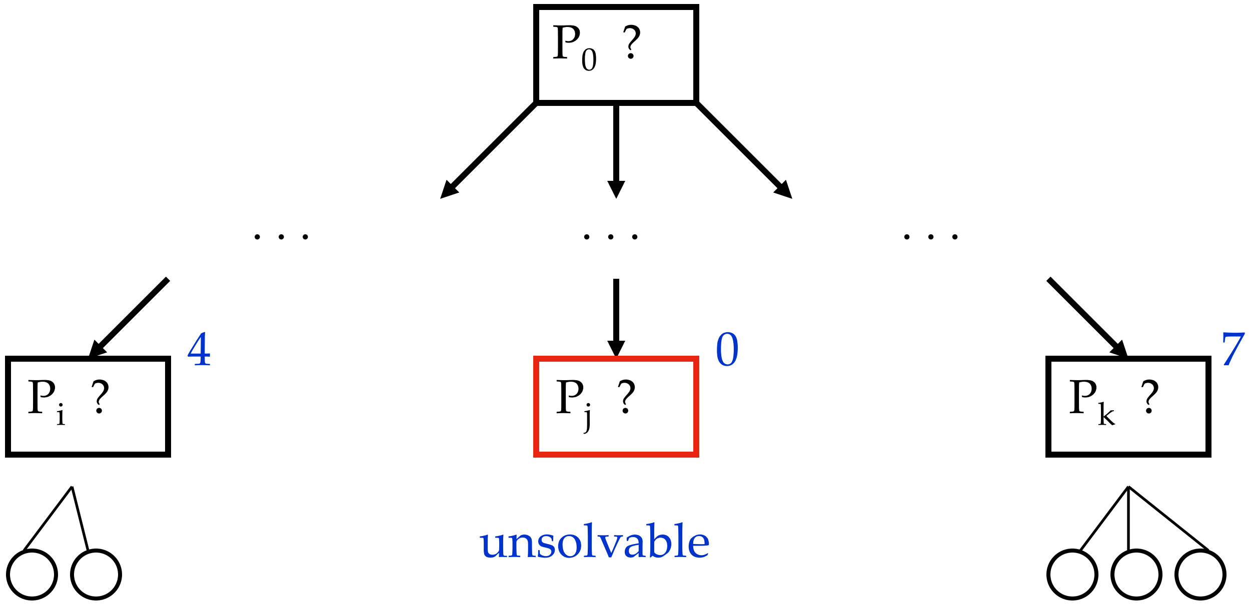
Not used for optimization, but useful if all you need is one valid solution

Less formally

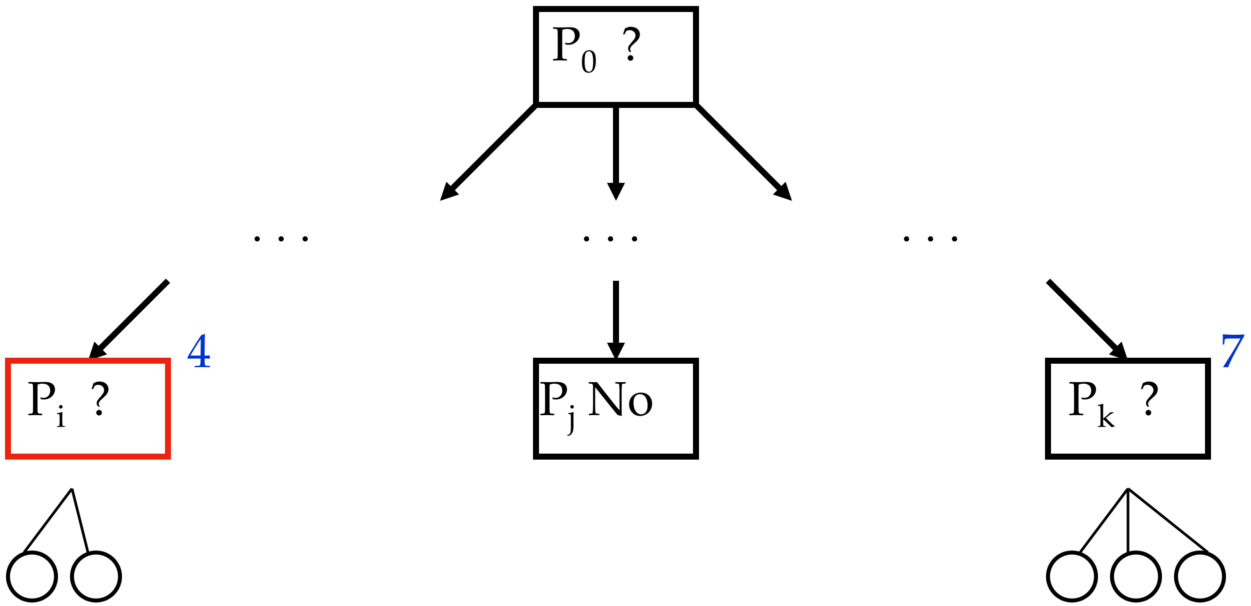
- If all alternative decisions to a leaf are guaranteed to lead to a solution, we often do not want the alternatives showing up in the search state (☞ no temptation to change choices and do therefore redundant work).
- Then we combine this first decision with the next decision and have several transitions to a leaf (see example).
- The search is finished, if the problem in one leaf has sol-entry yes (or all alternatives have proven to fail).

Visualize

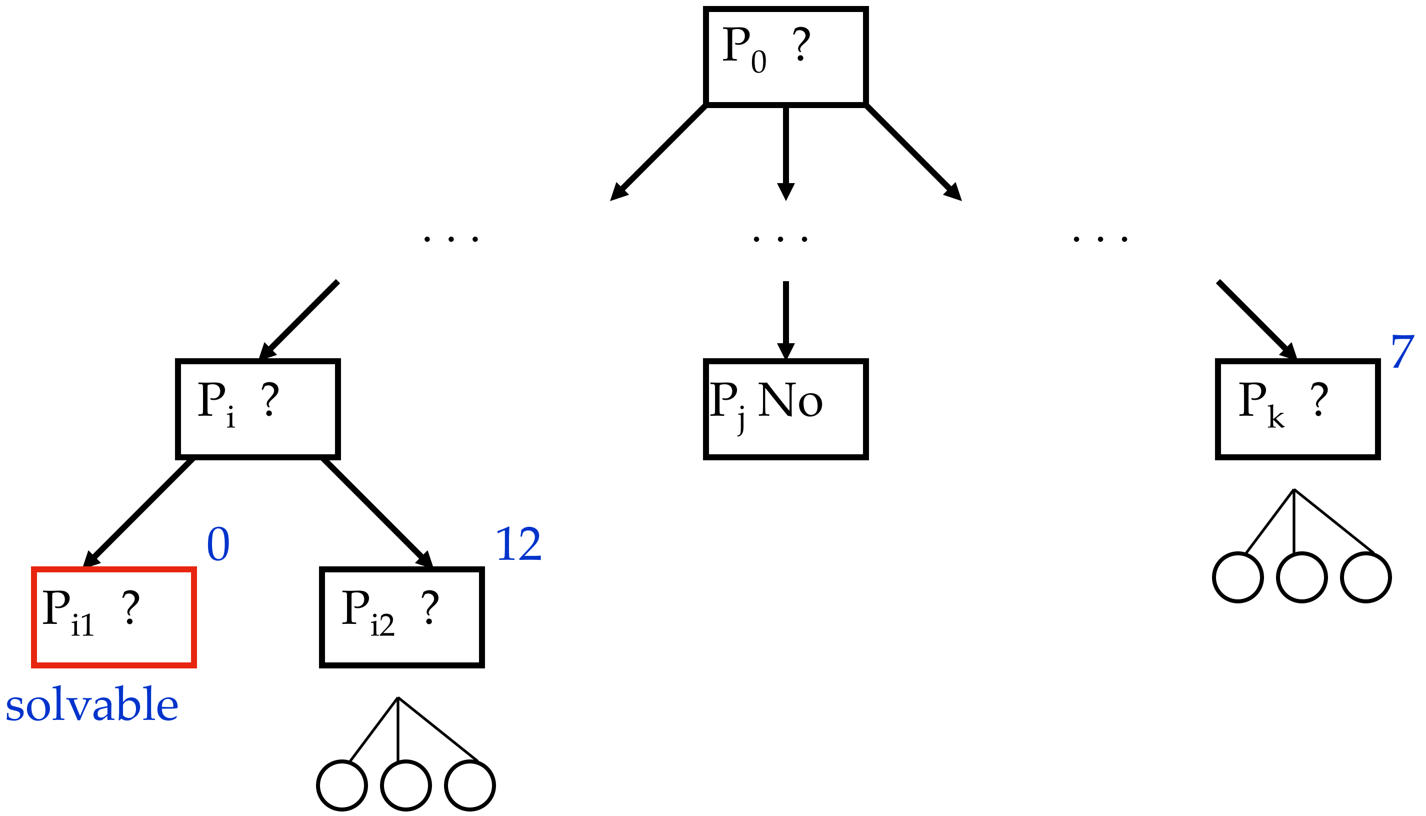
Conceptual Example (III): Or-tree-based Search



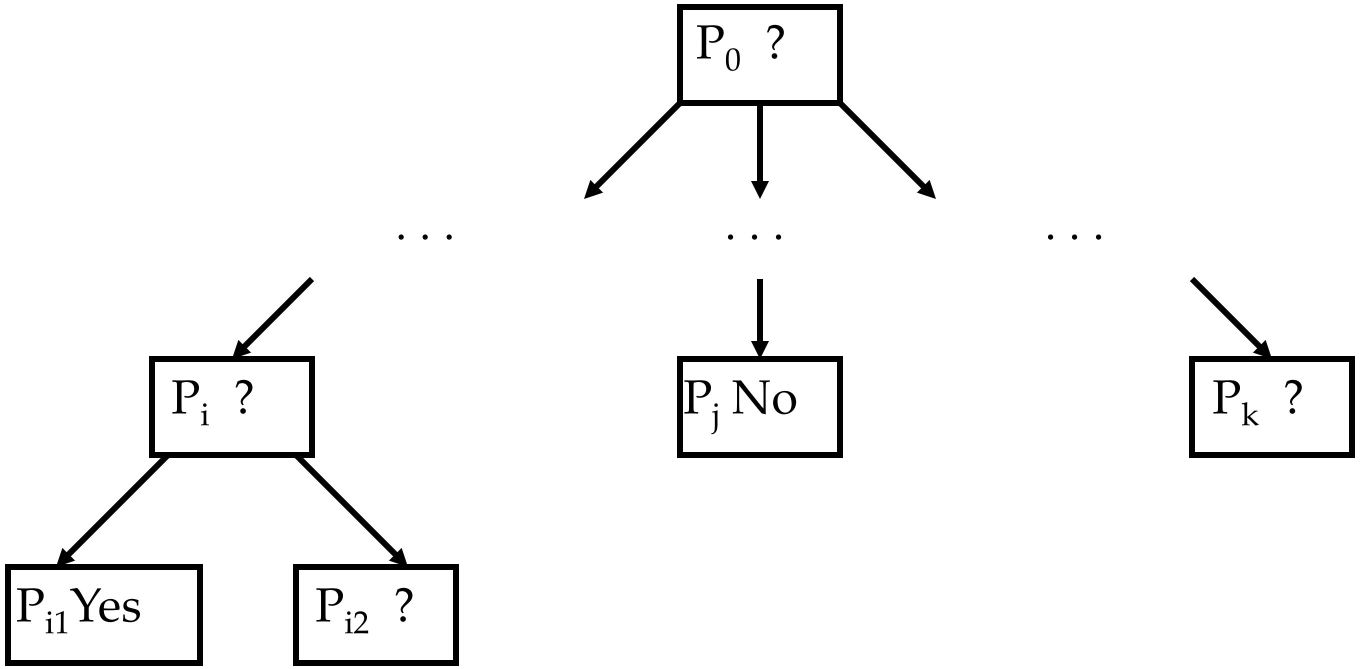
Conceptual Example (III): Or-tree-based Search



Conceptual Example (III): Or-tree-based Search



Conceptual Example (III): Or-tree-based Search



☞ finished

Design

Designing or-tree-based search models

1. Identify how you can describe a problem (resp. what is needed to describe steps towards a solution)
   ☞ $Prob$
2. Define how to identify if a problem is solved
3. Define how to identify if a problem is unsolvable
4. Identify the basic methods how a problem can be brought nearer to a solution; collect all these ideas for each problem ☞ $Altern$
5. Check if you really need all methods or if finding a solution can be already guaranteed without a particular one ☞ you might get rid of it

Designing or-tree-based search processes

1. Identify how you can measure the problem in a leaf regarding how far away from a solution it is
   - ☞ Priority to problems that are solved or unsolvable
2. Use 1. to select the leaf nearest a solution (if necessary, define tiebreakers)
3. If you have alternative collections of alternatives (i.e. several transitions with the same first problem in $Altern$), select one of them either using 1. for all successor problems or some other criteria (see and-trees for ideas)

Onward to … constraint satisfaction via and-tree-based search

Jonathan Hudson, Ph.D.
jwhudson@ucalgary.ca
https://cspages.ucalgary.ca/~jwhudson/

UNIVERSITY OF CALGARY