And-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

And-tree-based Search

Basic Idea:

1. Divide a problem into subproblems, whose solutions can be put together into a solution for the initial problem.

Examples of subproblem division:

- Construction of something: different parts of it
- Optimization problems: different instantiations of free variables; putting solution together by comparing all possibilities

And-tree-based Search

Good for optimization problems

- In simplest form they are exhaustive earch
  - search for all options and then return the optimal option
- Tree can be bounded (pruned)
  - branch-and-bound algorithms

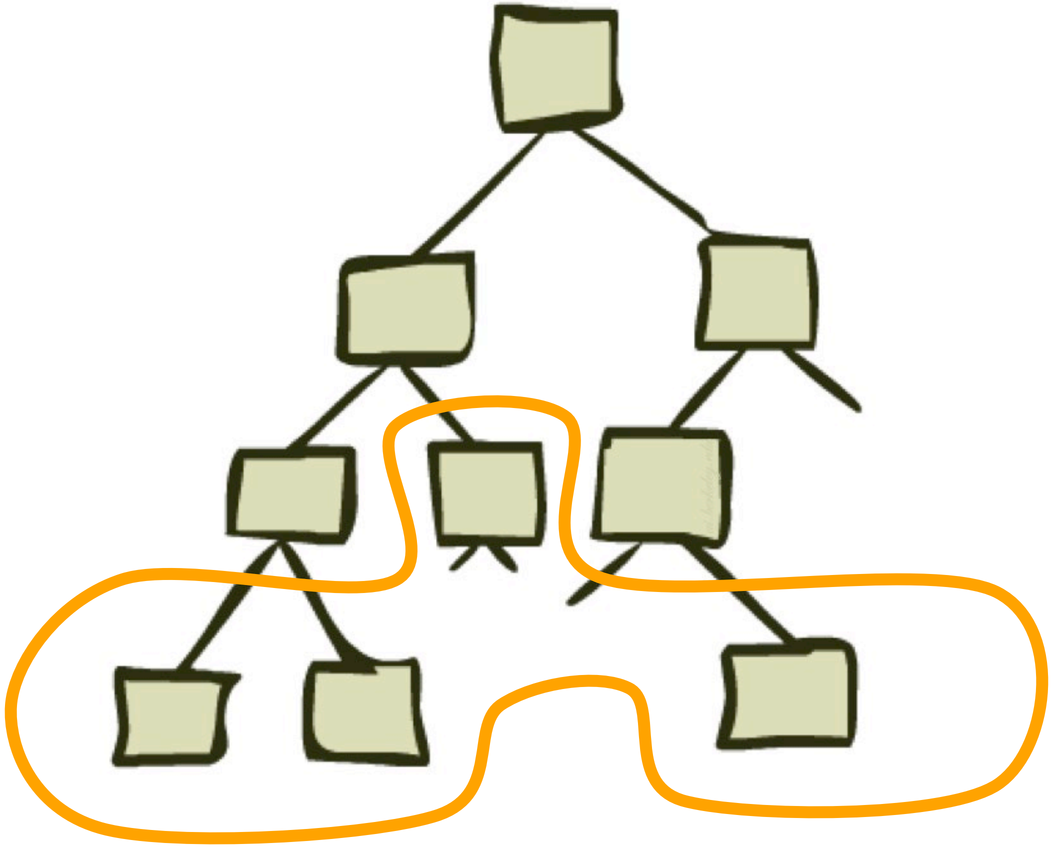
Good for problems where you need to solve all sub-problems and combine them

- Divide-and-conquer algorithms
- Recursive algorithms

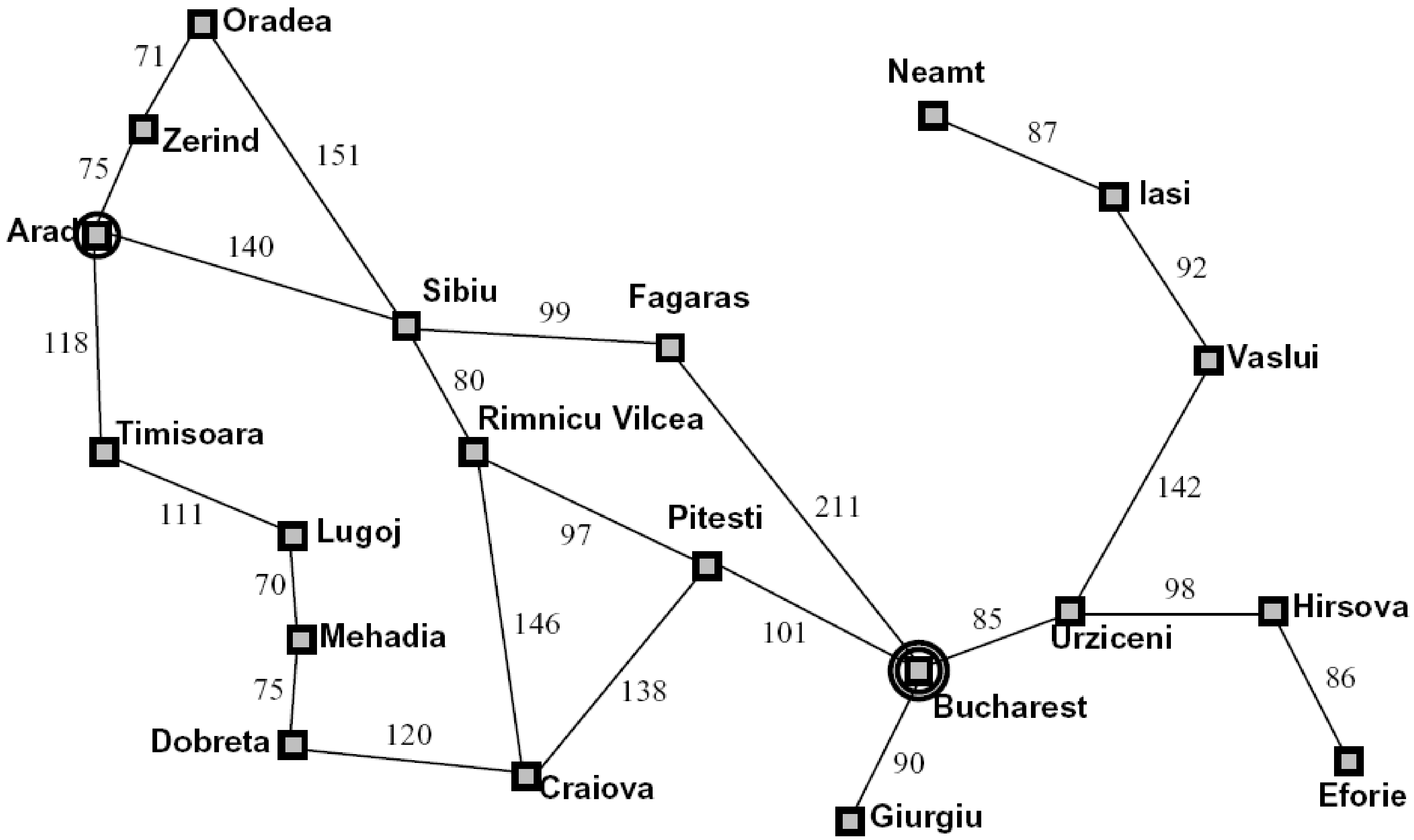
Take a lot of space and computation (but that’s how we get optimal results)

Tree Search

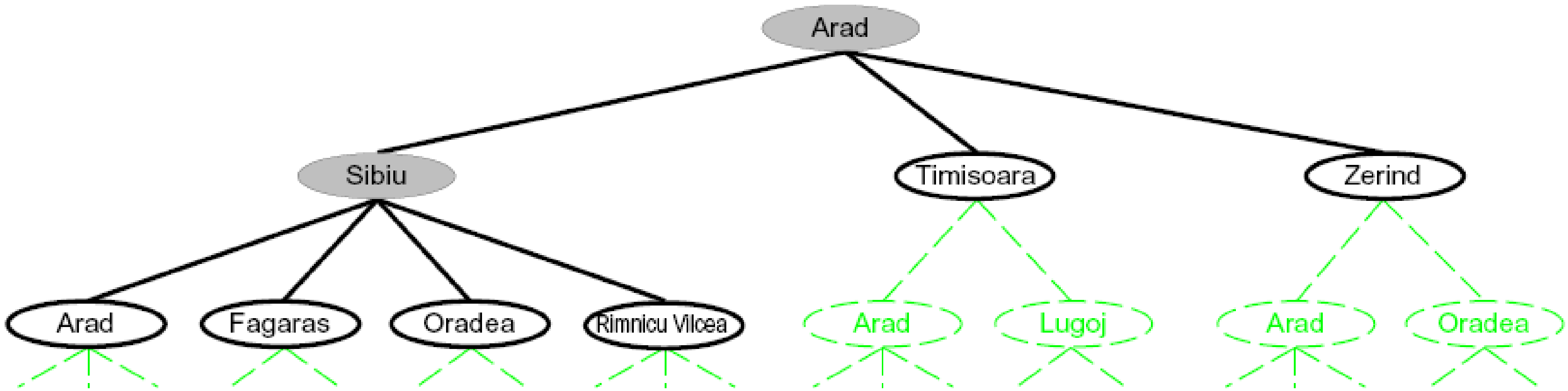
Tree Search



Search Example: Romania



Searching with a Search Tree



- Search:
  - Expand out potential plans (tree nodes)
  - Maintain a fringe of partial plans under consideration
  - Try to expand as few tree nodes as possible

General Tree Search

```
function TREE-SEARCH( problem, strategy) returns a solution, or failure
    initialize the search tree using the initial state of problem
    loop do
        if there are no candidates for expansion then return failure
        choose a leaf node for expansion according to strategy
        if the node contains a goal state then return the corresponding solution
        else expand the node and add the resulting nodes to the search tree
    end
```

- Important ideas:
  - Fringe
  - Expansion
  - Exploration strategy

- Main question: which fringe nodes to explore?

Model

Formal Definitions: Model

And-tree-based Search Model

$A_\wedge = (S_\wedge, T_\wedge)$

$\boldsymbol{Prob}$ set of problem descriptions

$\boldsymbol{Div} \subseteq \boldsymbol{Prob}^+$ division relation ($\boldsymbol{Prob}^+ \rightarrow$ things that can be generated by dividing problems in $\boldsymbol{Prob}$)

$S_\wedge \subseteq \boldsymbol{Atree}$ set of possible states, is subset tree structures

where $Atree$ is recursively defined by

$(pr, sol) \in \boldsymbol{Atree}$ for $pr \in \boldsymbol{Prob}$, $sol \in \{\boldsymbol{yes}, ?\}$

$(pr, sol, b_1, \dots, b_n) \in \boldsymbol{Atree}$ for $pr \in \boldsymbol{Prob}$, $sol \in \{\boldsymbol{yes}, ?\}$, $b_i \in \boldsymbol{Atree}$

$T_\wedge \subseteq S_\wedge \times S_\wedge$ transitions between states, but more specifically

$T_\wedge = \{(s_1, s_2) \mid s_1, s_2 \in S_\wedge \text{ and } Erw_\wedge(s_1, s_2) \text{ or } Erw_\wedge^*(s_1, s_2)\}$

Formal Definitions: Model

And-tree-based Search Model

$A_\wedge = (S_\wedge, T_\wedge)$

$\boldsymbol{Prob}$ set of problem descriptions

$\boldsymbol{Div} \subseteq \boldsymbol{Prob}^+$ division relation ($\boldsymbol{Prob}^+ \rightarrow$ things that can be generated by dividing problems in $\boldsymbol{Prob}$)

$S_\wedge \subseteq \boldsymbol{Atree}$ set of possible states, is subset tree structures

where $Atree$ is recursively defined by

$(pr, sol) \in \boldsymbol{Atree}$ **Leaf nodes**

$(pr, sol, b_1, \dots, b_n) \in \boldsymbol{Atree}$ **Internal nodes**

$T_\wedge \subseteq S_\wedge \times S_\wedge$ transitions between states, but more specifically

$T_\wedge = \{(s_1, s_2) \mid s_1, s_2 \in S_\wedge \text{ and } \boldsymbol{tree\ expansion} \text{ or } \boldsymbol{tree\ contraction}\}$

Formal Definitions: Model

And-tree-based Search Model

$A_\wedge = (S_\wedge, T_\wedge)$

You need to make

$\boldsymbol{Prob}$

$\boldsymbol{Div} \subseteq \boldsymbol{Prob}^+$

Comes for free by model definition

$S_\wedge \subseteq \boldsymbol{Atree}$ set of possible states, is subset tree structures

$T_\wedge \subseteq S_\wedge \times S_\wedge$ transitions between states, but more specifically

Less formally: Model

- $\boldsymbol{Prob}$ usually is described using an additional data structure: a set of formulas describing the world, a matrix describing distances to remaining cities, and so on.
- $\boldsymbol{Prob}$ can also just remember all decisions made so far
- Obviously, different problems produce different sets $\boldsymbol{Prob}$

- $\boldsymbol{Div}$ formally describes what divisions of problems into subproblems are possible; also absolutely dependent on the problem we want to solve.

Less formally: Model (II)

- A node containing a problem and a sol-entry is an **and-tree** (***Atree***).
- If we have several (i.e. n) **and-trees**, then putting them as successors to a node representing a problem and a sol-entry also produces an **and-tree**.

Note: this does not say anything about the connection between the problems in such a tree; in fact, most elements of $\boldsymbol{Atree}$ will never be used as search states, because they do not make sense for the application.

(There are a lot of expansions defined, but many are useless to be chosen by a useful search control)

Extension function (tree expansion and contraction)

Formal Definitions: Erw (Extension function)

$Erw_\wedge$ and $Erw_\wedge^*$ are relations on $Atree$ defined by

- $Erw_\wedge\big((pr,?),(pr,\mathbf{yes})\big)$ if pr is solved
- $Erw_\wedge\Big((pr,?),\big(pr,?,(pr_1,?),\ldots,(pr_n,?)\big)\Big)$ if $\mathrm{D}iv(pr,pr_1,\ldots,pr_n)$ holds
- $Erw_\wedge\big((pr,?,b_1,\ldots,b_n),(pr,?,b_1',\ldots,b_n')\big)$
  if for an i: $Erw_\wedge(b_i,b_i')$ and $b_j = b_j{}'$ for $i{\neq}j$
- $Erw_\wedge \subseteq Erw_\wedge^*$
- $Erw_\wedge^*\big((pr,?,b_1,\ldots,b_n),(pr,?,b_1{}',\ldots,b_n{}')\big)$
  if for all i either $Erw_\wedge^*(b_i,bi')$ or $b_i = b_i{}'$ holds

UNIVERSITY OF CALGARY

Formal Definitions: Erw (Extension function)

$Erw_\wedge$ and $Erw_\wedge^*$ are relations on $Atree$ defined by

- $Erw_\wedge\big((pr, ?), (pr, \mathbf{yes})\big)$ **leaf node is done**
- $Erw_\wedge\Big((pr, ?), \big(pr, ?, (pr_1, ?), \ldots, (pr_n, ?)\big)\Big)$ **leaf expansion**
- $Erw_\wedge\big((pr, ?, b_1, \ldots, b_n), (pr, ?, b_1', \ldots, b_n')\big)$ **allow above leaf rules to apply to more than root of tree**
- $Erw_\wedge \subseteq Erw_\wedge^*$ **backtracking exists (can reverse expansion)**
- $Erw_\wedge^*\big((pr, ?, b_1, \ldots, b_n), (pr, ?, b_1', \ldots, b_n')\big)$ **backtracking can undo multiple**

Less formally: Erw (Extension function)

- $Erw_{\wedge}$ connects and-trees that reflect the idea of dividing problems into subproblems
  - if we know the **solution** to a problem in a **leaf** node (i.e. it is solved for us), we mark it (sol-entry yes)
  - else, if we know the division of a problem in a (**leaf**) node into subproblems, then we generate successors to this node for each subproblem
  - else, if we know the division of a problem in a (**internal**) node into subproblems, then we generate successors to this node for each subproblem
  - else, see remarks about back-tracking ($Erw_{\wedge}^{*}$)

Back-tracking (tree contraction)

Less formally: Erw* (Extension function)

- $Erw^*_\wedge$ is for intelligent backtracking (note the sequence of arguments in the definition of $T_\wedge$). It allows us to take away the results of several applications of $Erw_\wedge$ as one transition (therefore "intelligent").

- Backtracking is necessary, if you reach a tree with a leaf that neither represents a solved problem nor has a problem that can be divided into subproblems (or we already have unsuccessfully tried out all of its divisions defined by $\boldsymbol{Div}$).
  - *If we have (pr,?) and no remaining Div option at a leaf*
    - *Remember we need all leafs to reach (pr, yes) -> then we back-track*
    - *We back-track (by collapsing the tree upwards to internal nodes)*
    - *Until we reach an internal node where we have another Div option we can select instead of the prior Div option we had selected, then we chose this next Div option instead*
- Controls usually employ backtracking only in very clearly defined (special) cases.

Process

Formal Definitions: Search Process

And-tree-based Search Process

$P_\wedge = (A_\wedge, Env, K_\wedge)$

Not more specific than general definition given previously

But: often control uses two functions

- one function $\boldsymbol{f_{leaf}}$ that compares all leaves of the tree representing the state and selecting one
- one function $\boldsymbol{f_{trans}}$ that selects one of the transitions that deal with the selected leaf

Less formally: Search Process

- Due to the possibility of having several divisions of the same problem in $\boldsymbol{Div}$, first determining a leaf to "expand" and then selecting the division is often sensible.
- But sometimes the availability of certain divisions determines what leaf to select next, so that $\boldsymbol{f_{leaf}}$ and $\boldsymbol{f_{trans}}$ are not always used.
- An and-tree-based search starts with putting the problem instance to solve into the root of an and-tree.
- If we have found a solution to every subproblem represented by a leaf, then it is still possible that the solutions are not compatible. Then other solutions have to be found (☞ backtracking).

Instance

Formal Definitions: Search Instance (IV)

And-tree-based Search Instance

$Ins_\wedge = (s_0, G_\wedge)$

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

- $s_0 = (pr, ?)$
- $G_\wedge(s) = yes$, if and only if
  1. $s = (pr', \boldsymbol{yes})$ or
  2. $s = (pr', ?, b_1, \ldots, b_n), G_\wedge(b_1) = \cdots = G_\wedge(b_n) = \boldsymbol{yes}$ and the solutions to $b_1, \ldots, bn$ are compatible with each other or
  3. there is no transition that has not been tried out already

UNIVERSITY OF CALGARY

Formal Definitions: Search Instance (IV)

And-tree-based Search Instance

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

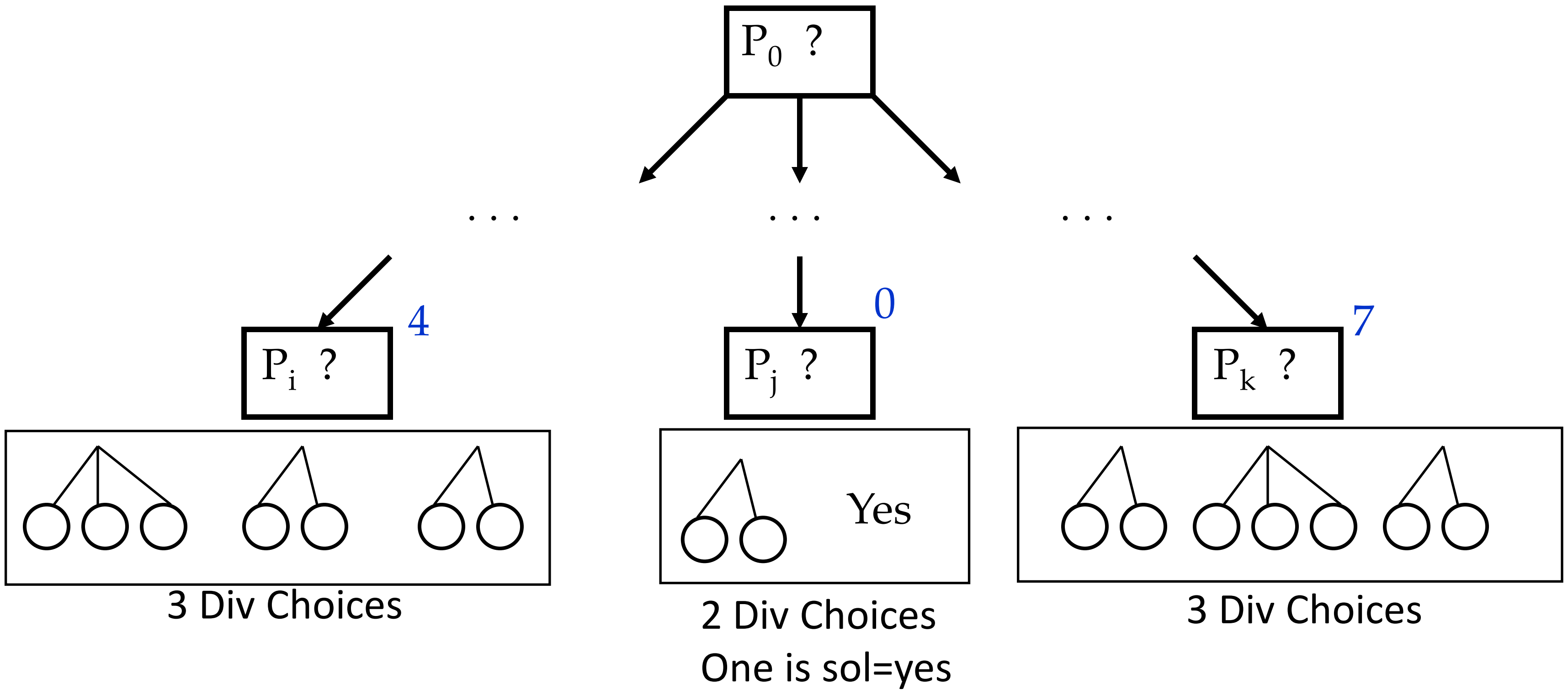
If the given problem to solve is pr, then we have

- $s_0 = (pr, ?)$
- $G_{\wedge}(s) = yes$, if and only if
  1. Root is yes
  2. All branches are yes and compatible
  3. We've tried everything, all remaining leaves are ?, and we've tried all back-tracking and alternate Div expansions

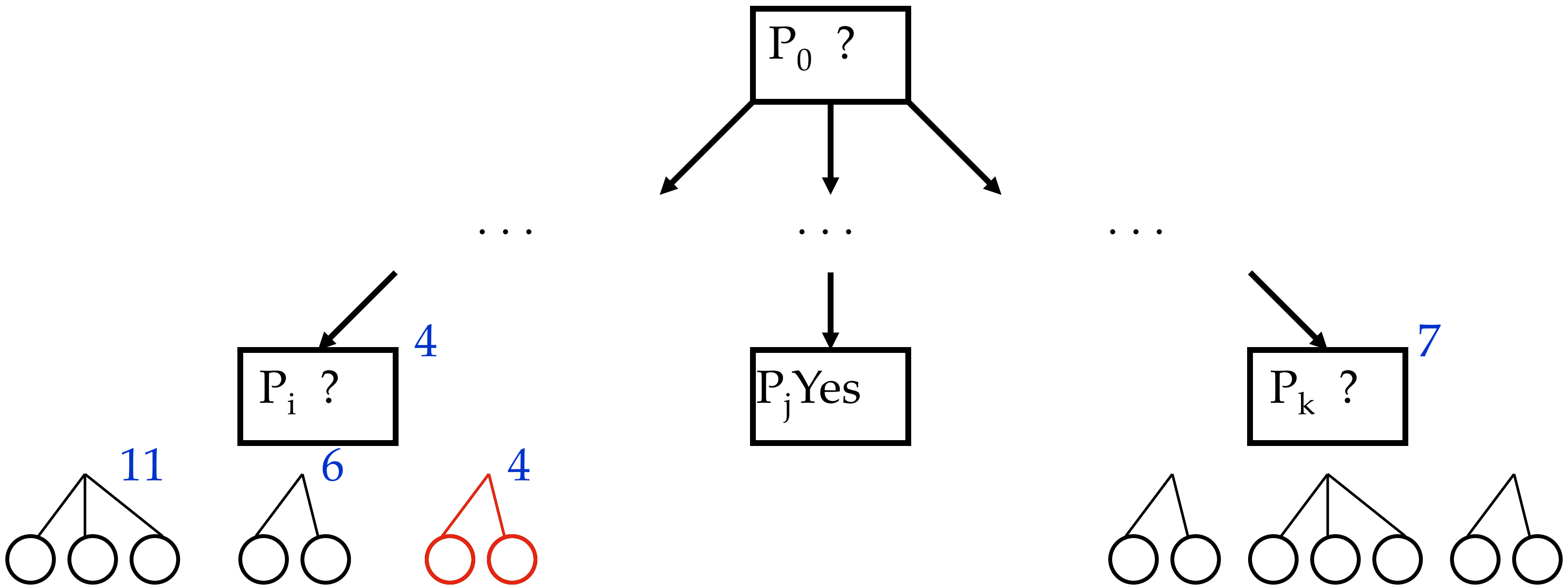
Common for optimization as we either need to find the best of all valid solutions, or find no valid solution

Visualize

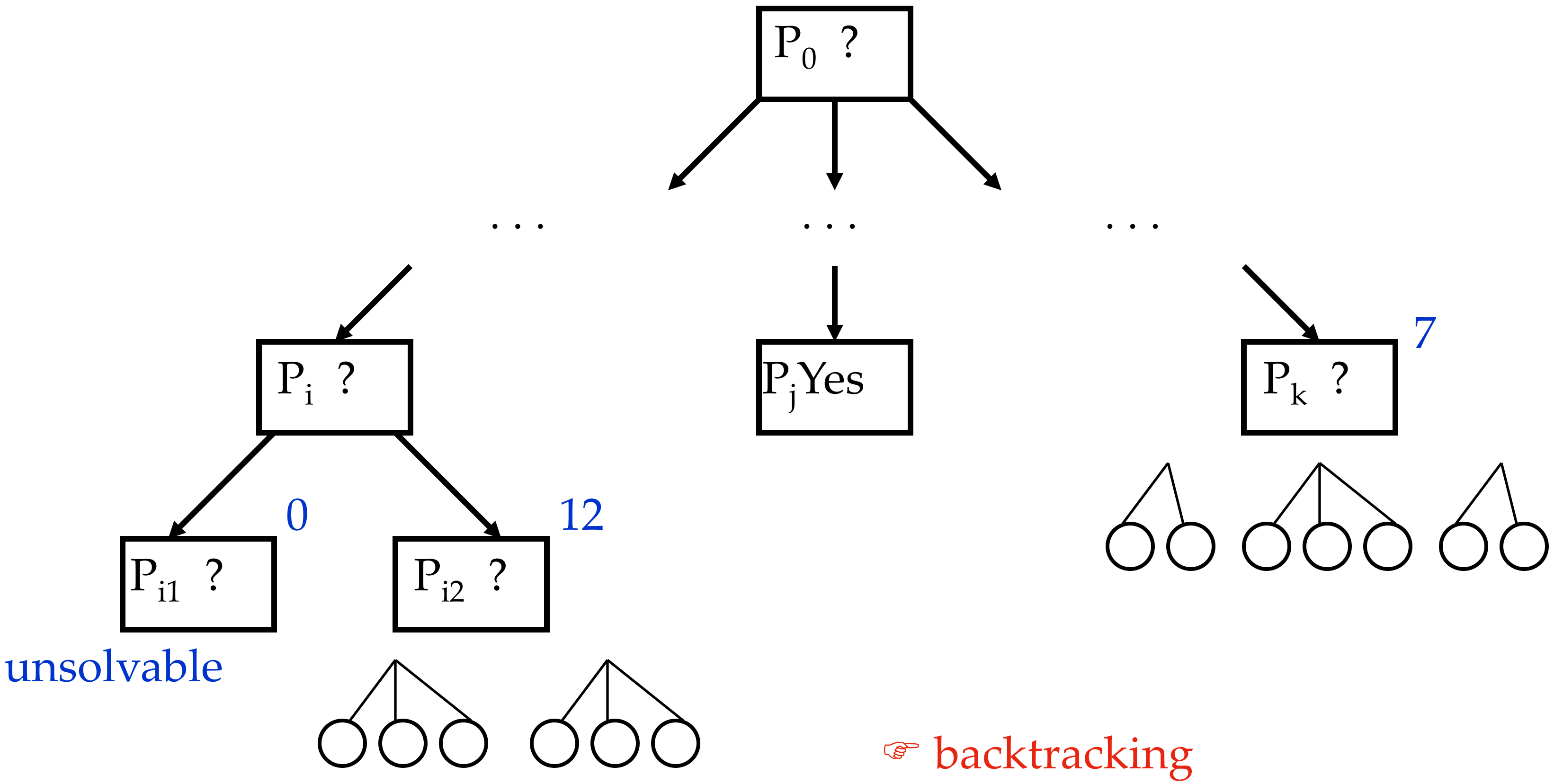
Conceptual Example (II): And-tree-based Search



Conceptual Example (II): And-tree-based Search



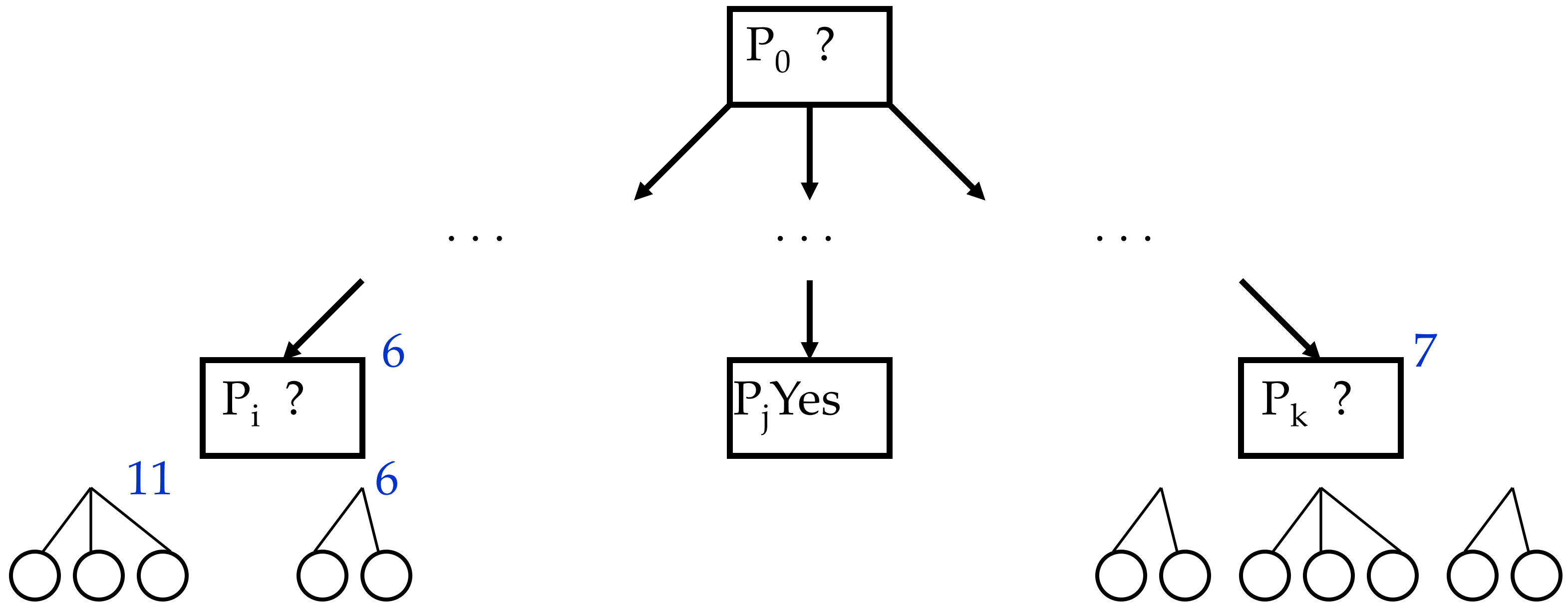
Conceptual Example (II): And-tree-based Search



☞ backtracking

UNIVERSITY OF CALGARY

Conceptual Example (II): And-tree-based Search



Design

Designing and-tree-based search models

1. Identify how you can describe a problem (resp. what is needed to describe sub-problems) ☞ $\boldsymbol{Prob}$
2. Define how to identify if a problem is solved
3. Identify the basic ideas how to divide a problem into subproblems ☞ $\boldsymbol{Div}$
4. Determine if it is possible that you run into deadends (i.e. can there be leafs that neither are solved nor appear in $\boldsymbol{Div}$ as first argument). If yes, we need backtracking, if no, we do not need backtracking.

Designing and-tree-based search processes

1. Identify how you can measure a problem in a leaf
    1. Priority to problems that are solved
    2. See other slides for criteria
2. Use 1. to come up with a $\boldsymbol{f_{leaf}}$-function comparing the leaves in an and-tree.
3. For the $\boldsymbol{f_{trans}}$-function that determines the transition you are doing:
    1. If there is an unsolvable problem in a leaf then backtrack
    2. If the selected leaf can be solved, do it
    3. Determine the different divisions of the leaf problem and measure them

Onward to … model-elimination via and-tree-based search

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