Artificial Intelligence: Or-Tree-based Search

CPSC 433: Artificial Intelligence
Fall 2022

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

Thursday, October 13, 2022

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!

Definitions

Formal Definitions: Search Model

Or-tree-based Search Model $A_{\mathrm{V}} = (S_{\mathrm{V}}, T_{\mathrm{V}})$

$Prob$ set of problem descriptions

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

$S_{\mathrm{V}} \subseteq Otree$ set of possible states, is subset tree structures

where $Otree$ is recursively defined by

$(pr, sol) \in Otree$ for $pr \in Prob, sol \in \{yes, ?, \boldsymbol{no}\}$

$(pr, sol, b_1, \ldots, bn) \in Otree$ for $pr \in Prob, sol \in \{yes, ?, \boldsymbol{no}\}, b_i \in Otree$

$T_{\mathrm{V}} \subseteq S_{\mathrm{V}} \times S_{\mathrm{V}}$ transitions between states, but more specifically

$T_{\mathrm{V}} = \{(s_1, s_2) \mid s_1, s_2 \in S_{\mathrm{V}}$ and $Erw_{\mathrm{V}}(s_1, s_2)$ ~~or $Erw_{\mathrm{V}}^{*}(s_1, s_2)$~~$\}$

UNIVERSITY OF CALGARY

Less formally: Search Model

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

Formal Definitions: Erw

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

- $Erw_\vee((pr,?),(pr,yes))$ if pr is solved
- $Erw_\vee((pr,?),(pr,no))$ 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 $i{\neq}j$

Formal Definitions: Search Process

Or-tree-based Search Process $P_{\vee} = (A_{\vee}, Env, K_{\vee})$

Not more specific than general definition

What is selected is the leaf to expand.

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

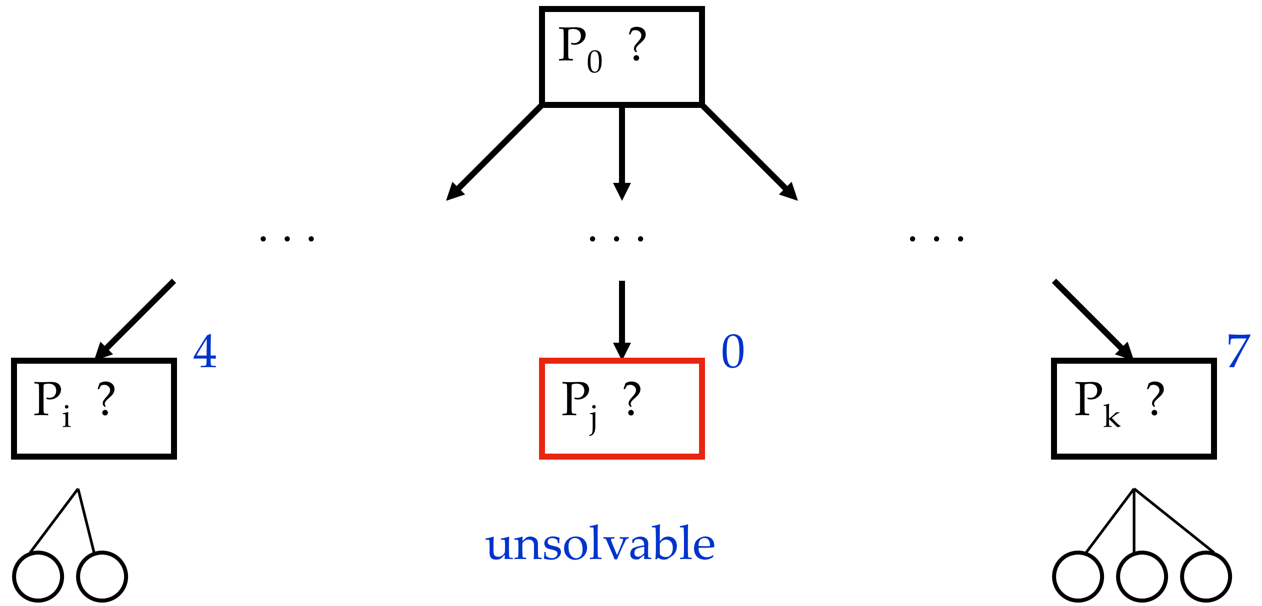
- $s_0 = (pr, ?)$
- $G_{\vee}(s) = yes$, if and only if
  - $s = (pr', yes)$ or
  - $s = (pr', ?, b_1, \ldots, b_n), G_{\vee}(b_i) = yes$ for an i or
  - All leafs of s have either the sol-entry no or cannot be processed using $Altern$

Less formally

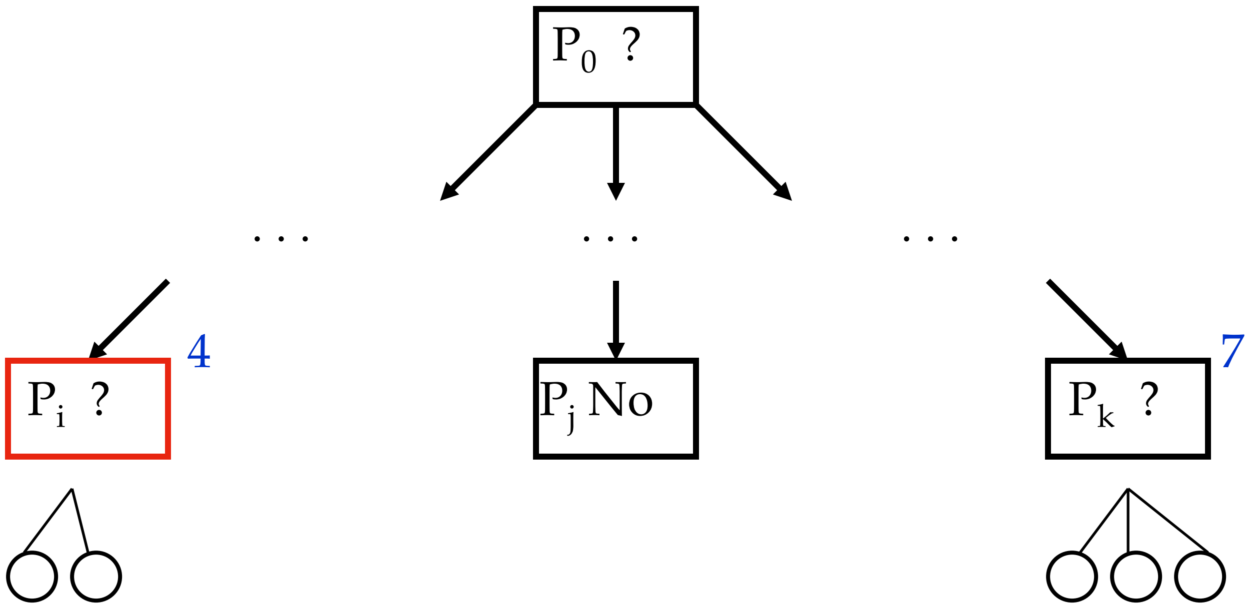
- If all alternative decisions to a leaf are guaranteed to lead to a solution, we often do not want the alternatives showing up in the search state (☞ no temptation to change choices and do therefore redundant work).
  Then we combine this first decision with the next decision and have several transitions 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

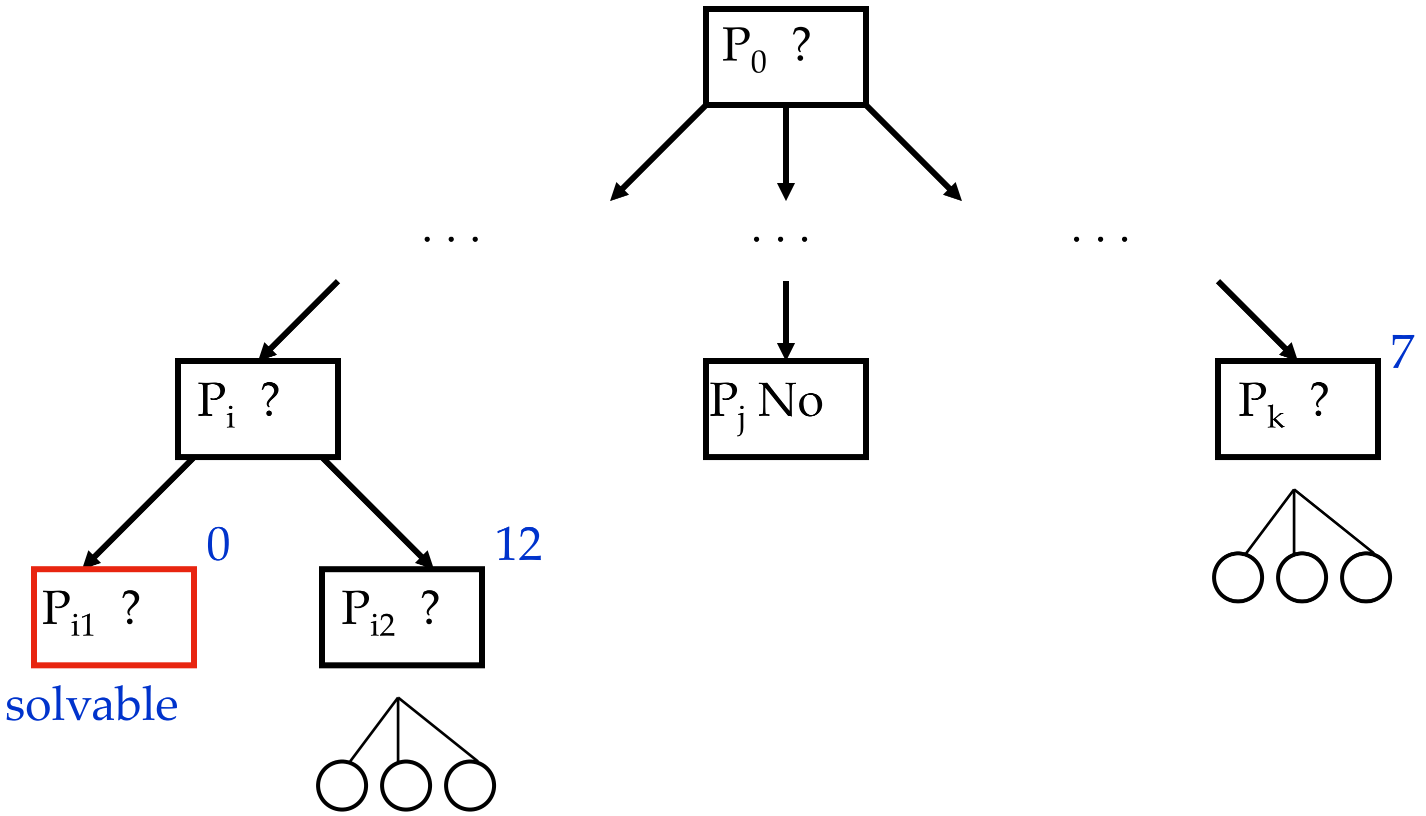
Conceptual Example (III): Or-tree-based Search



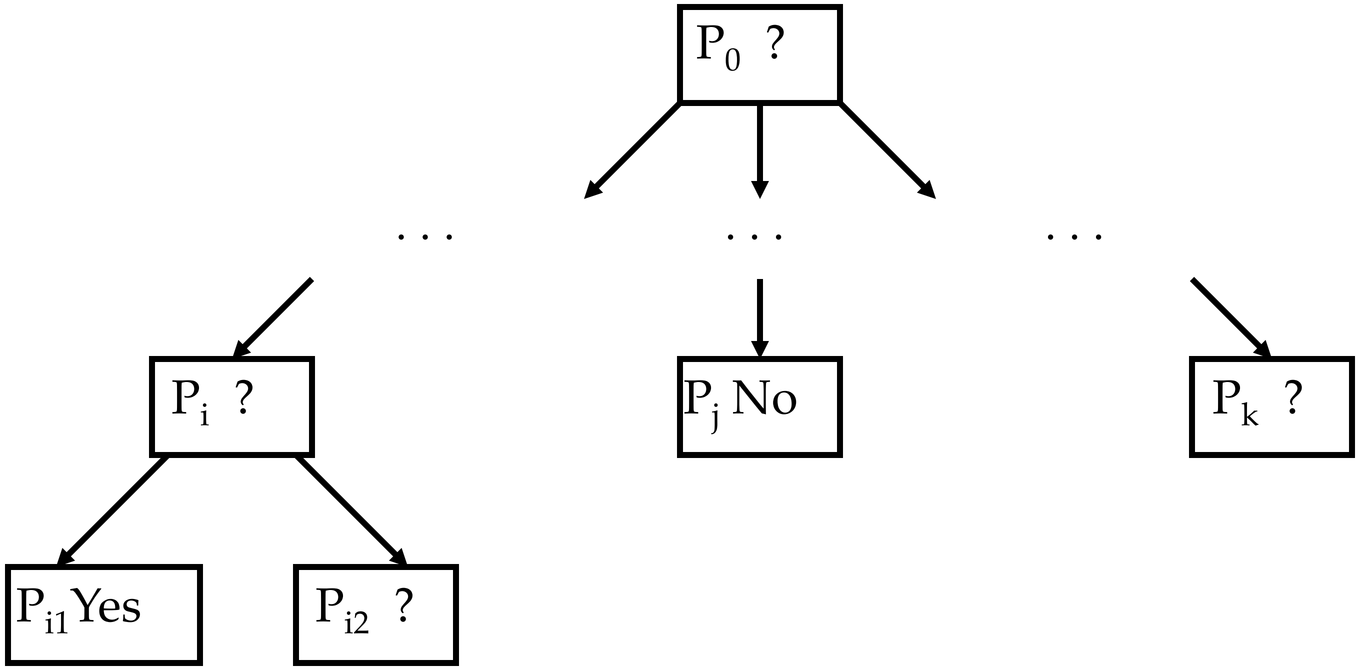
Conceptual Example (III): Or-tree-based Search



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)

Constraint Satisfaction

What is Search For?

- Assumptions about the world: a single agent, deterministic actions, fully observed state, discrete state space

- Planning: sequences of actions
  - The path to the goal is the important thing
  - Paths have various costs, depths
  - Heuristics give problem-specific guidance

- Identification: assignments to variables
  - The goal itself is important, not the path
  - All paths at the same depth (for some formulations)
  - CSPs are specialized for identification problems

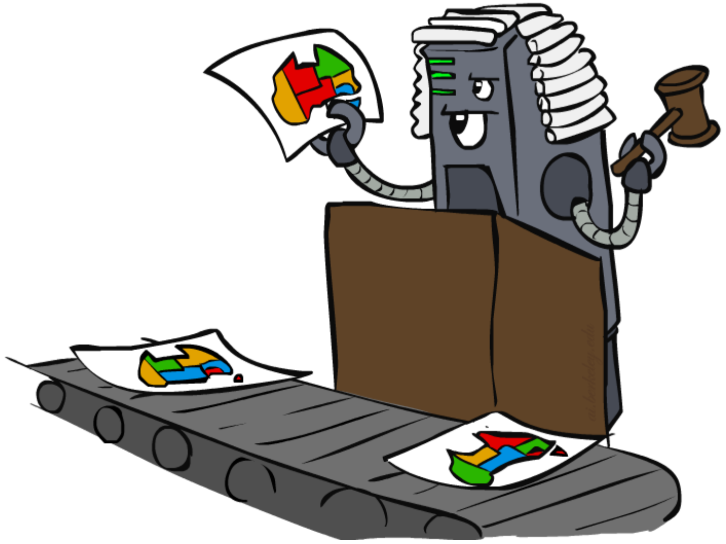
Constraint Satisfaction Problems

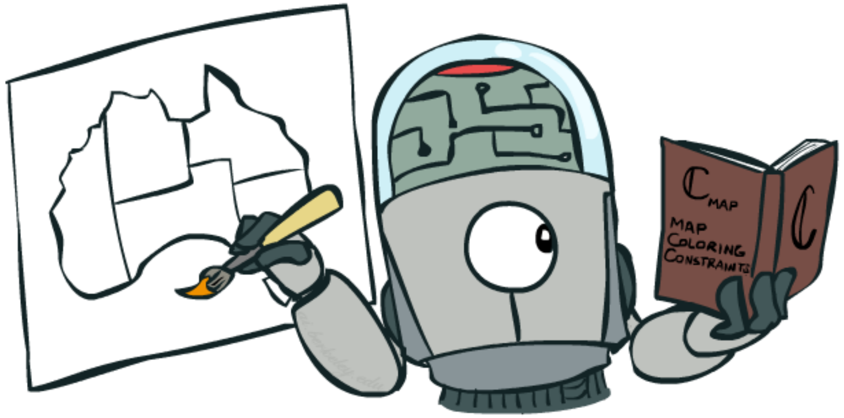
Constraint Satisfaction Problems

- Standard search problems:
  - State is a “black box”: arbitrary data structure
  - Goal test can be any function over states
  - Successor function can also be anything

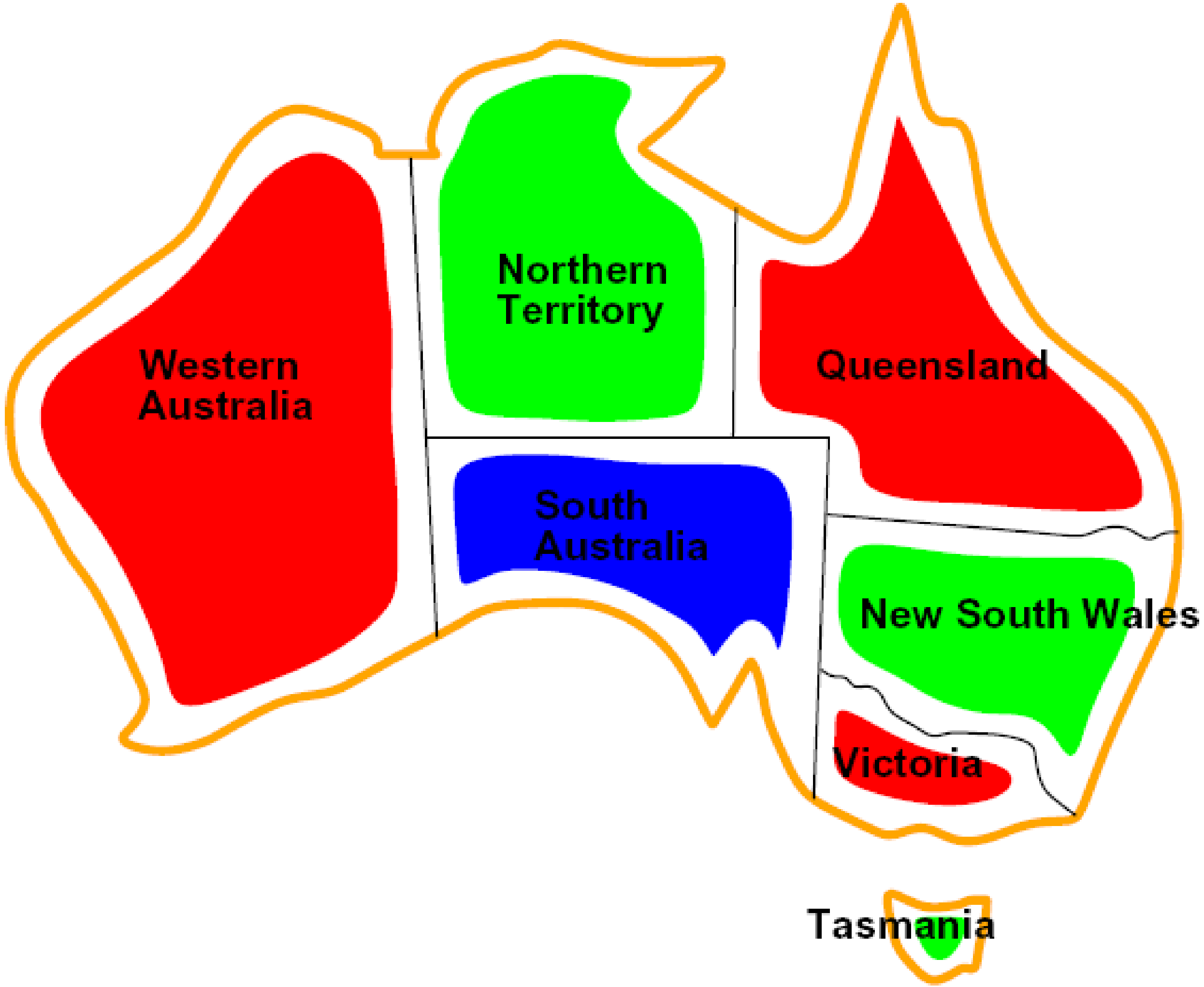
- Constraint satisfaction problems (CSPs):
  - A special subset of search problems
  - State is defined by variables X_i with values from a domain D (sometimes D depends on i)
  - Goal test is a set of constraints specifying allowable combinations of values for subsets of variables

- Allows useful general-purpose algorithms with more power than standard search algorithms





CSP Examples



Example: Map Coloring

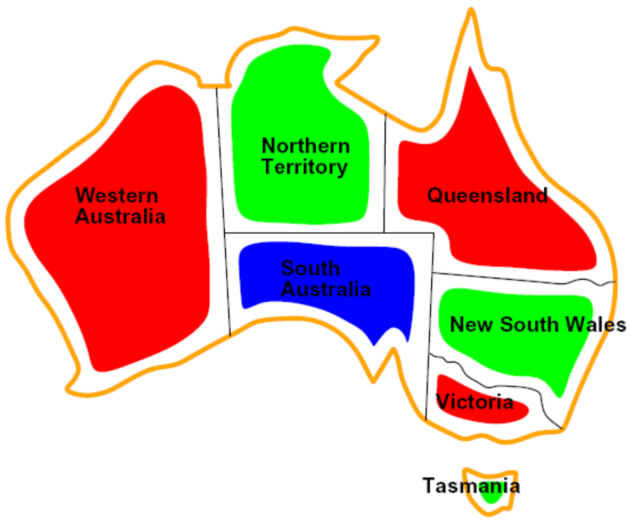
- Variables: $\mathrm{WA,\ NT,\ Q,\ NSW,\ V,\ SA,\ T}$
- Domains: $D = \{\text{red}, \text{green}, \text{blue}\}$
- Constraints: adjacent regions must have different colors

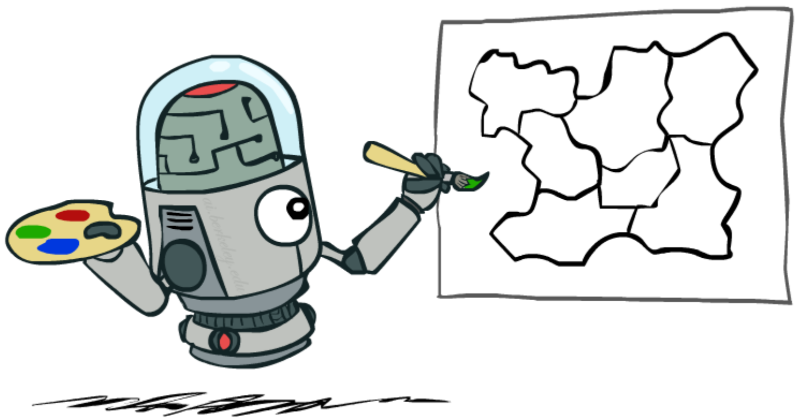
  Implicit: $\mathrm{WA} \neq \mathrm{NT}$

  Explicit: $(\mathrm{WA}, \mathrm{NT}) \in \{(\text{red}, \text{green}), (\text{red}, \text{blue}), \ldots\}$

- Solutions are assignments satisfying all constraints, e.g.:

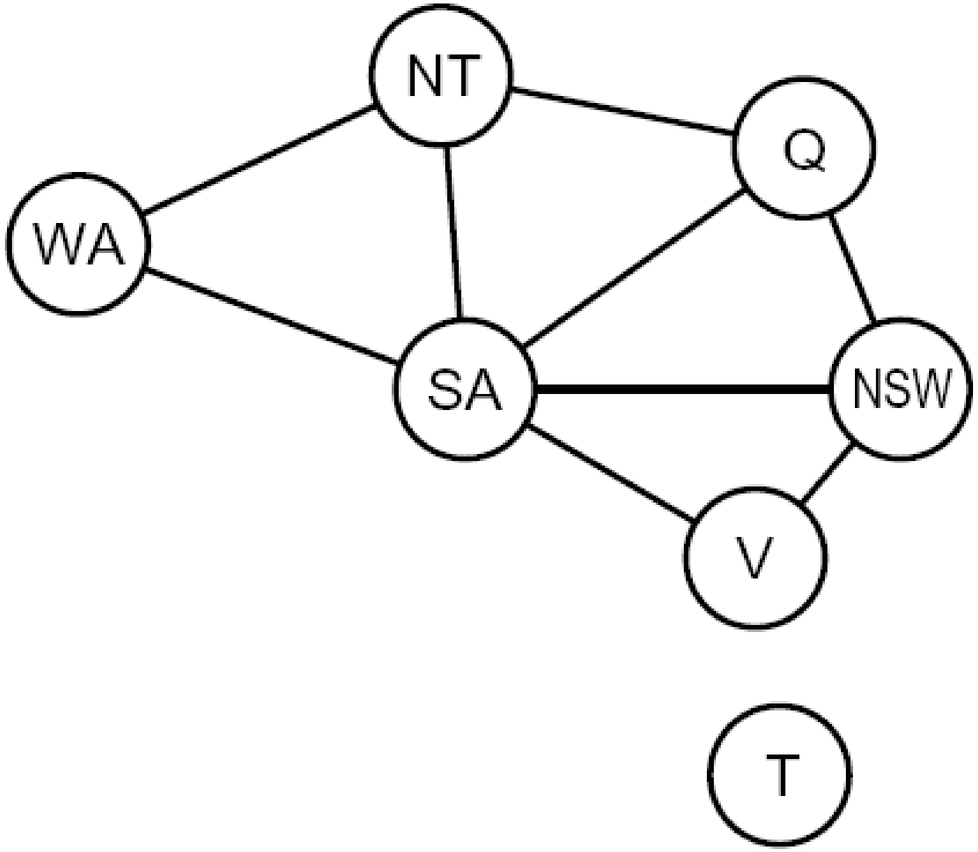
  {WA=red, NT=green, Q=red, NSW=green, V=red, SA=blue, T=green}





UNIVERSITY OF CALGARY

Constraint Graphs



Example: Cryptarithmetic

- Variables:

  $F \ T \ U \ W \ R \ O \ X_1 \ X_2 \ X_3$

- Domains:

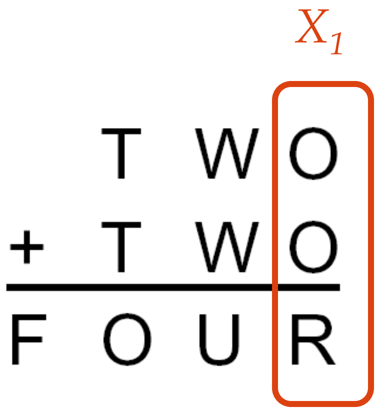
  $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

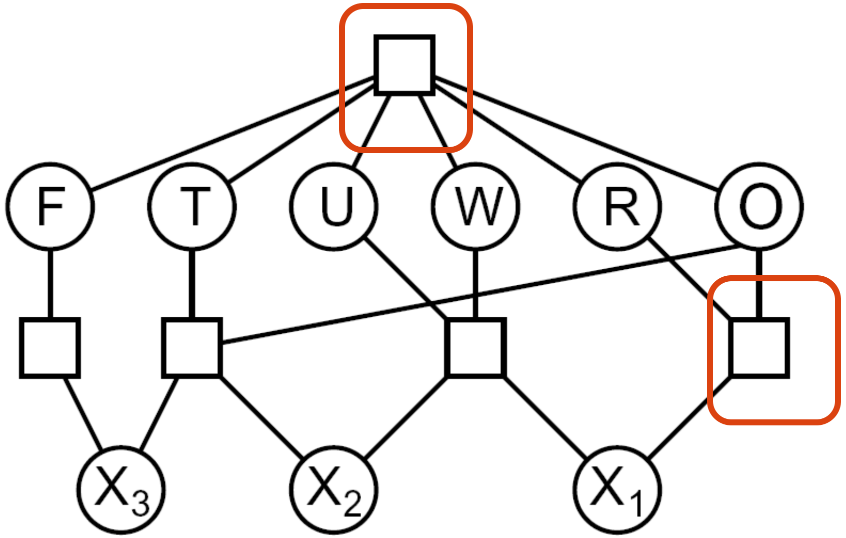
- Constraints:

  $\text{alldiff}(F, T, U, W, R, O)$

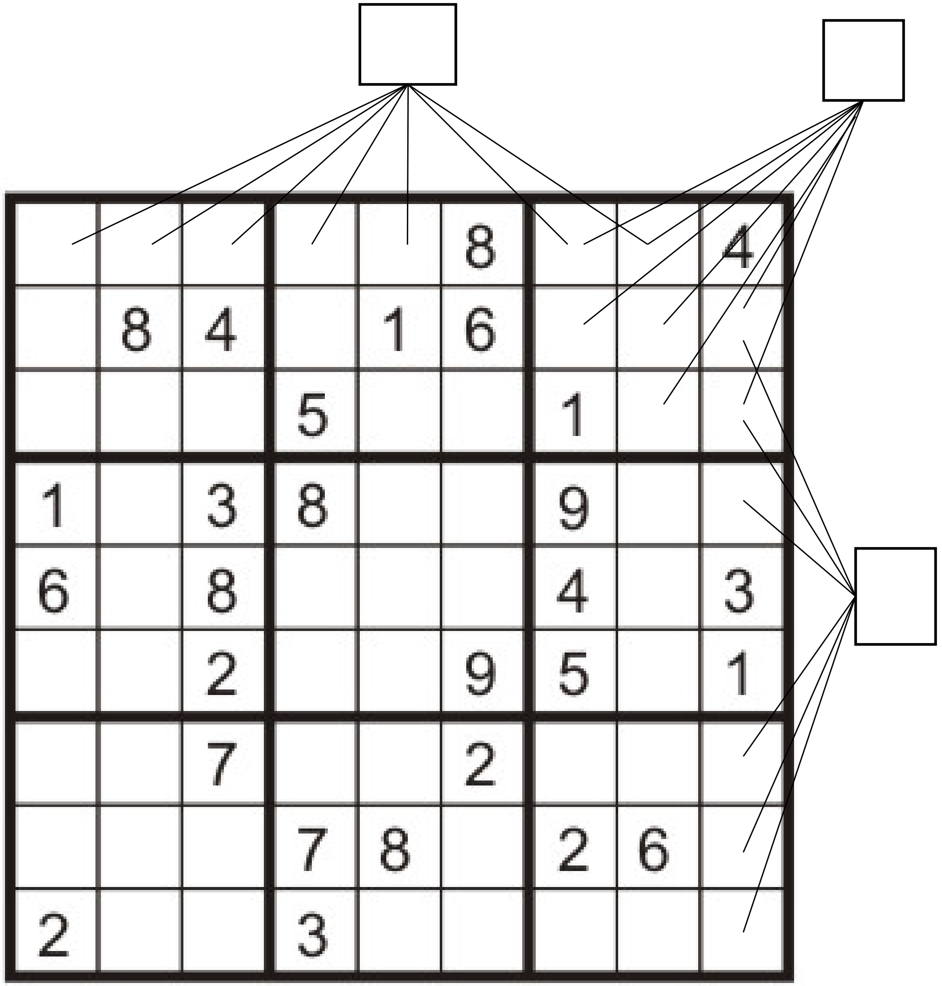
  $O + O = R + 10 \cdot X_1$

  $\cdots$





Example: Sudoku



- Variables:
  - Each (open) square
- Domains:
  - {1,2,…,9}
- Constraints:

9-way alldiff for each column

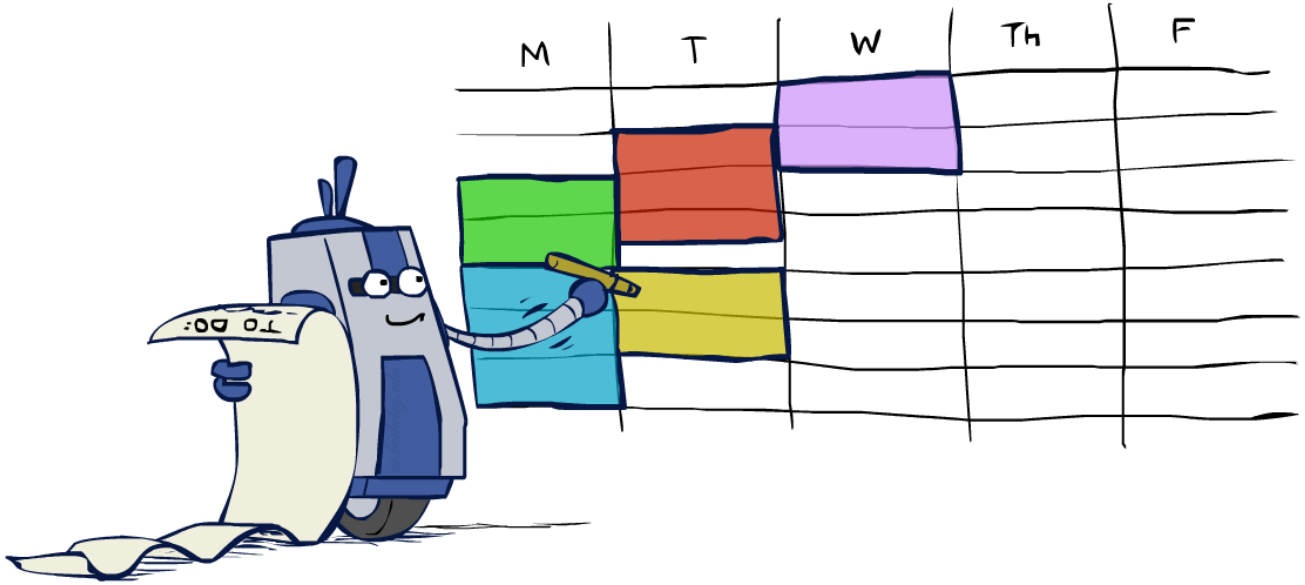
9-way alldiff for each row

9-way alldiff for each region

(or can have a bunch of pairwise inequality constraints)

Real-World CSPs

- Assignment problems: e.g., who teaches what class
- Timetabling problems: e.g., which class is offered when and where?
- Hardware configuration
- Transportation scheduling
- Factory scheduling
- Circuit layout
- Fault diagnosis
- … lots more!



- Many real-world problems involve real-valued variables…

Applied to Constraint Satisfaction

Solving CSPs

UNIVERSITY OF CALGARY

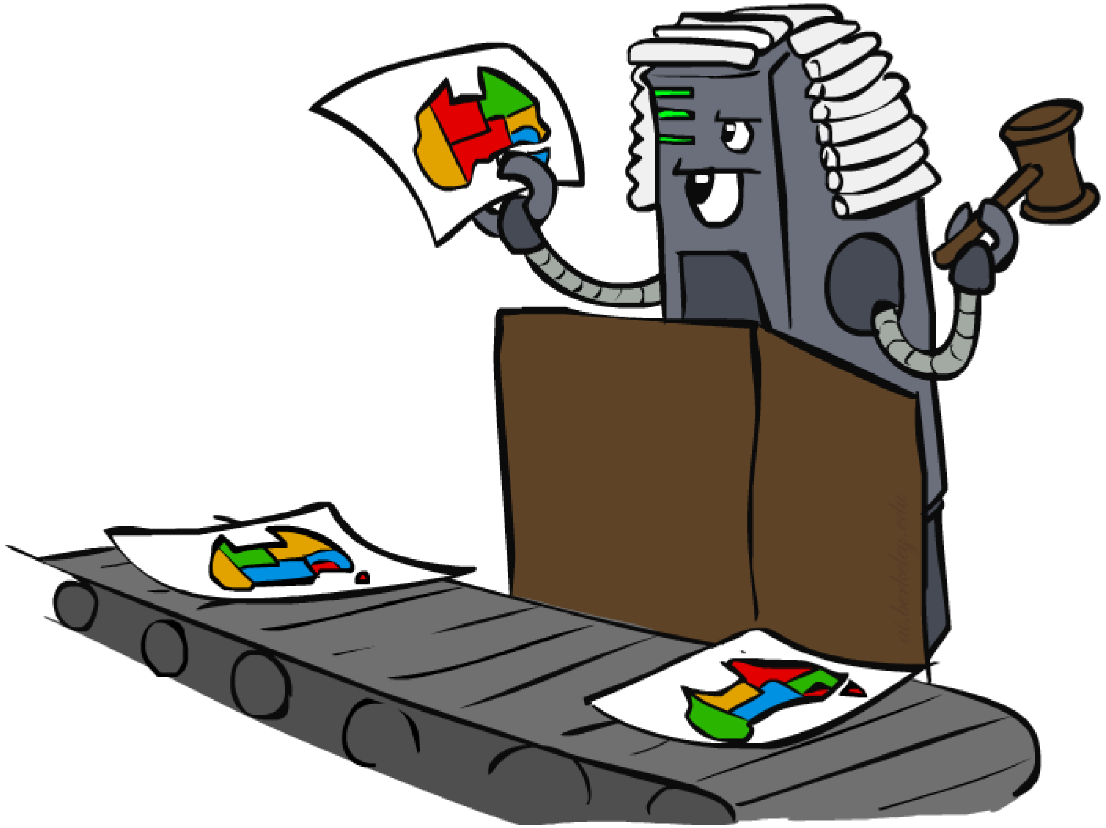
Concrete Example: Constraint Satisfaction (I)

- A constraint satisfaction problem (CSP) consists of
  - a set $X = \{X_1, \ldots, X_n\}$ of variables over some finite, discrete-valued domains $D = \{D_1, \ldots, D_n\}$ and
  - a set of constraints $C = \{C_1, \ldots, C_m\}$. Each constraint C_i is a relation over the domains of a subset of the variables, i.e.

$$C_i = R_i(X_{i,1}, \ldots, X_{i,k})$$

where the relation R_i describes every value-tuple in $D_{i,1} \times \cdots \times D_{i,k}$ that fulfills the constraint.

The problem is to find a value for each X_j (out of its D_j) that fulfills all C_i.



Constraint Satisfaction: Examples

Constraint Satisfaction (II): Examples

- $X = \{X_1, X_2\}$
$D_1 = \{1,2,3\}$
$D_2 = \{1,2,3,4\}$
$C = \{C_1, C_2, C_3\}$

$C_1: X_1 + X_2 \leq 4 \quad C_2: X_1 + X_2 \geq 3 \;\; C_3: X_1 \geq 2$

- $X = \{X_1, X_2, X_3\}$
$D_1 = D_2 = D_3 = \{true, false\}$
$C = \{C_1, C_2, C_3\}$

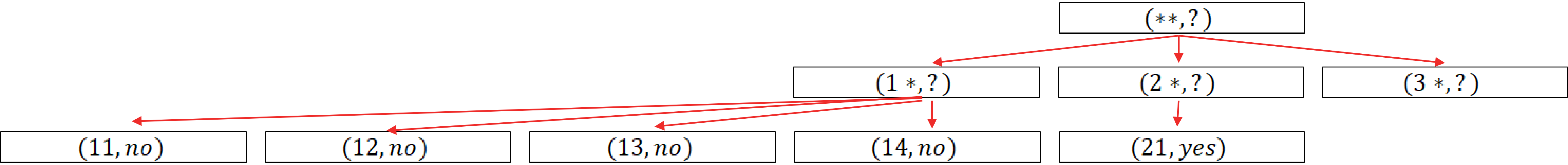
$C_1: X_1 \vee \neg X_2 \vee X_3 \quad C_2: \neg X_1 \vee X_3 \;\; C_3: \neg X_2 \vee \neg X_3$

Constraint Satisfaction (II): Examples

- $X = \{X_1, X_2\}$

$D_1 = \{1,2,3\}$
$D_2 = \{1,2,3,4\}$
$C = \{C_1, C_2, C_3\}$

$C_1: X_1 + X_2 \leq 4 \quad C_2: X_1 + X_2 \geq 3 \quad C_3: X_1 \geq 2$

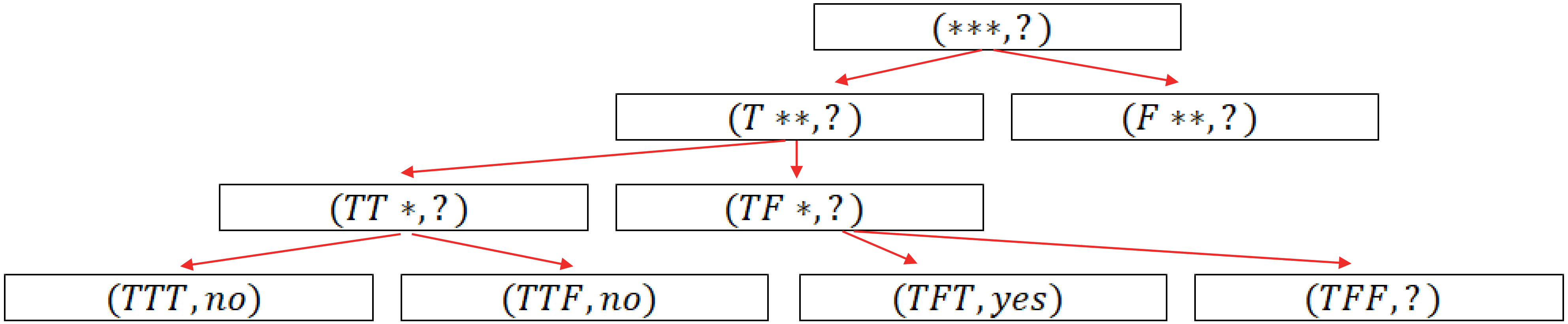


UNIVERSITY OF CALGARY

Constraint Satisfaction (II): Examples

- $X = \{X_1, X_2, X_3\}$
$D_1 = D_2 = D_3 = \{true, false\}$
$C = \{C_1, C_2, C_3\}$

$C_1: X_1 \vee \neg X_2 \vee X_3$ $C_2: \neg X_1 \vee X_3$ $C_3: \neg X_2 \vee \neg X_3$

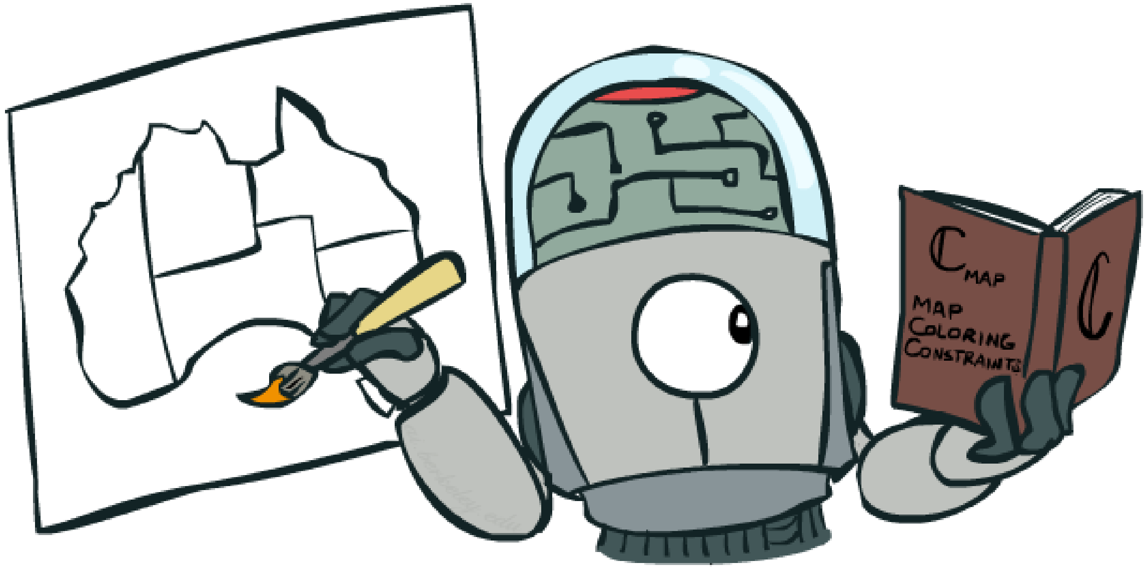


Constraint Satisfaction: Or-Tree-Based

Constraint Satisfaction (III)

Tasks:

- Describe CSPs as or-tree-based search model
- Describe formally a search control for your model based on the idea of identifying the variable occuring in the most constraints and selecting it and its domain for branching
(combined with a depth-criteria and a tiebreaker, if necessary)
- Solve the problem instances from the last slide



Search control for CSP example

Let $(pr_1,?),\ldots,(pr_o,?)$ be the open leafs in the current state and let

$const(X_j) = |\{C_i \mid C_i \in C, C_i = R_i(X_{i,1},\ldots,X_{i,k}), X_j \in \{X_{i,1},\ldots,X_{i,k}\}\}|$

For a problem $pr = (x_1,\ldots,x_n)$ let

$Csolved(pr) = |\{C_i | C_i \in C, x_1,\ldots,x_n \text{ fulfills } C_i\}|$

Then our search control K selects the leaf to work on and the transition to this leaf (there are several possible, i.e. special case on "Less formally (II)) as follows:

Search control for CSP example

If one of the pr_j is solved, perform the transition that changes its sol-entry. If there are several, select one of them randomly.

Else if one of the pr_j is unsolvable, perform the transition that changes its sol-entry. If there are several, again select one of them randomly.

Else

- select the leaf (pr_j,?) such that
  a) Csolved(pr_j) = max_{prl}({Csolved(pr_l)})
  b) if there are several, select the deepest leaf in the tree with this property.
  c) if there are still several, select the one the most left in the tree (tiebreaker without knowledge)

Search control for CSP example

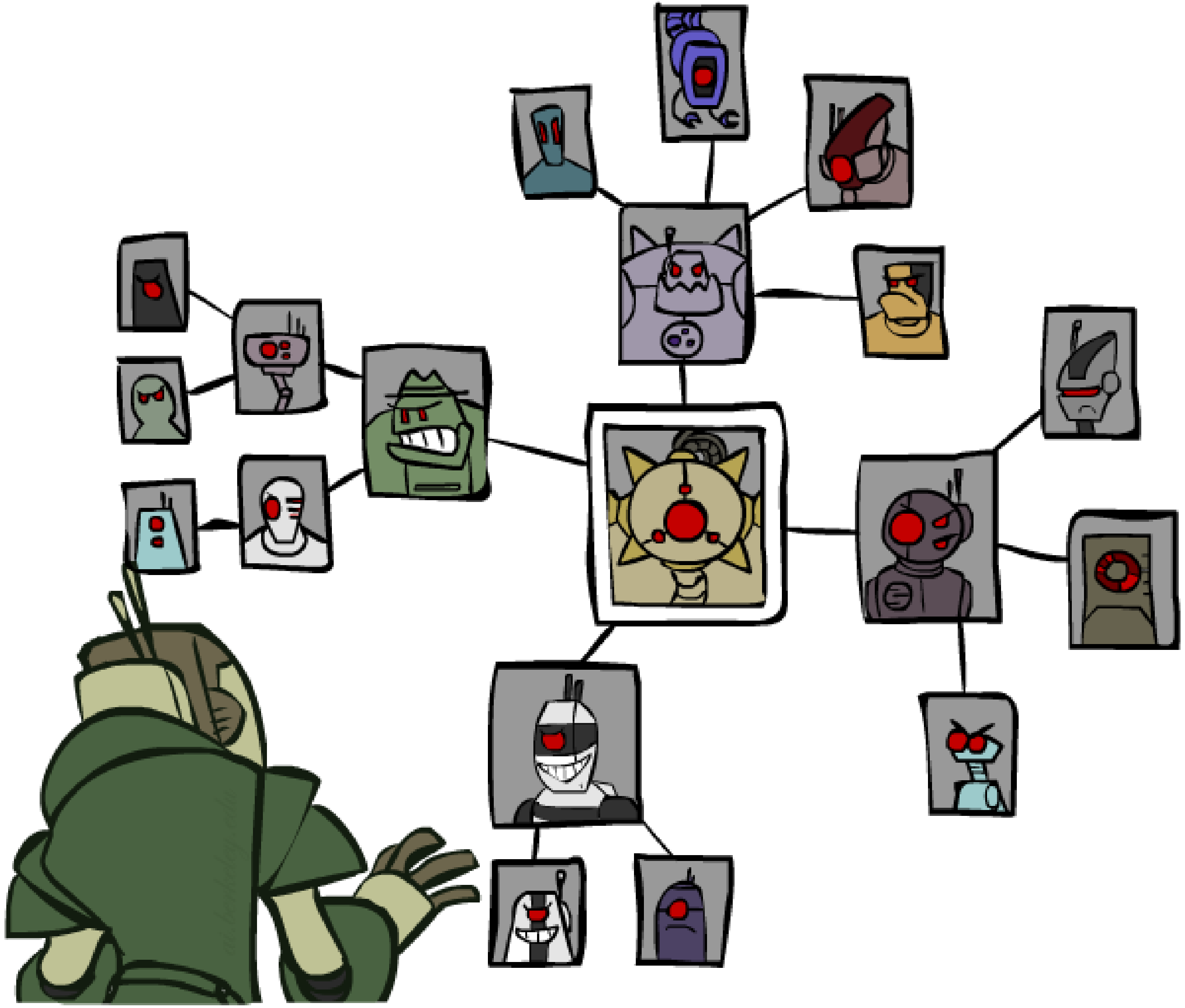
- for the transition select the one with *Altern*(pr_j,pr_{j1},...,pr_{jk}) such that the variable X_i we use to create the element in *Altern* is the one with maximal Const-value.
  If there are several of those, use the one with minimal index i (tiebreaker without knowledge)

Remarks

- And-tree-based and or-tree-based search have a lot in common. The difference from the search problem point of view can be best described as
  - or-tree: one solution
  - and-tree: all solutions
- Consequently, the criteria used by search controls differ, due to the different goals.
- A lot of problems have transformations into a CSP. Therefore there are a lot of papers on solving CSPs and good controls for it.

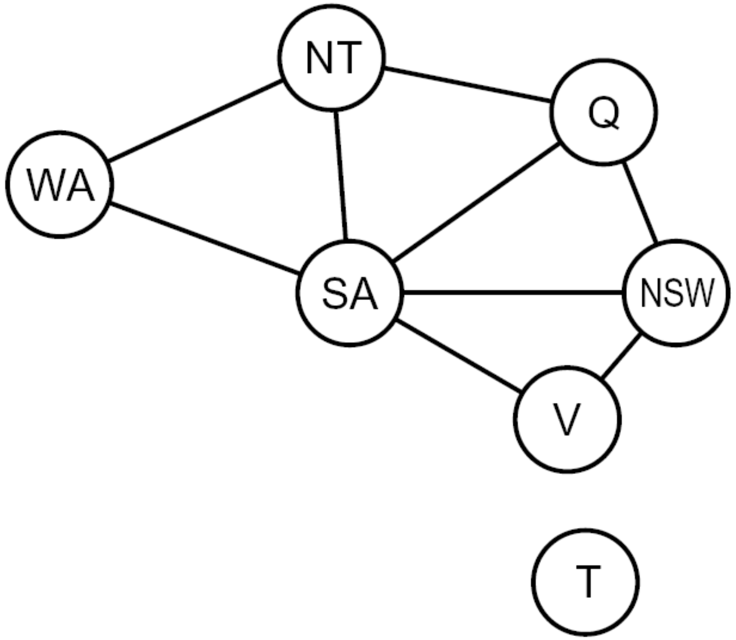
Structure?

Bonus (time permitting): Structure

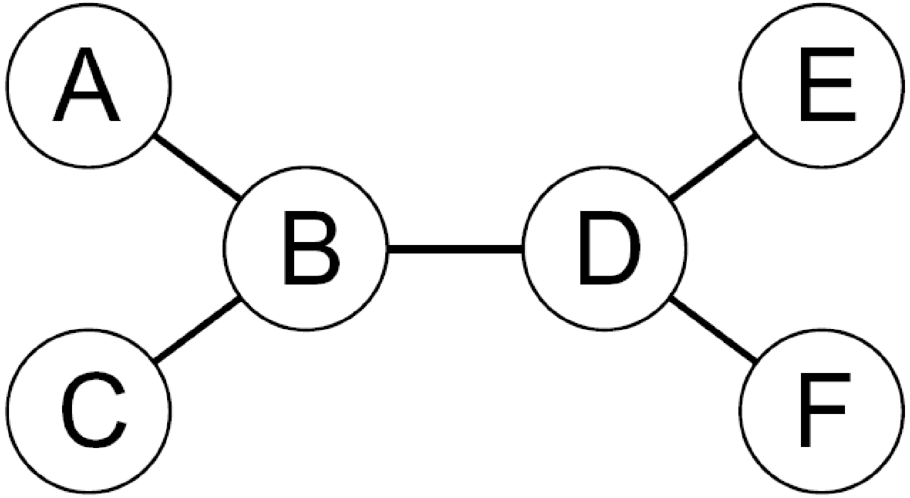


Problem Structure

- Extreme case: independent subproblems
  - Example: Tasmania and mainland do not interact
- Independent subproblems are identifiable as connected components of constraint graph
- Suppose a graph of n variables can be broken into subproblems of only c variables:
  - Worst-case solution cost is $O((n/c)(d^c))$, linear in n
  - E.g., n = 80, d = 2, c =20
  - 2^{80} = 4 billion years at 10 million nodes/sec
  - $(4)(2^{20})$ = 0.4 seconds at 10 million nodes/sec



Tree-Structured CSPs

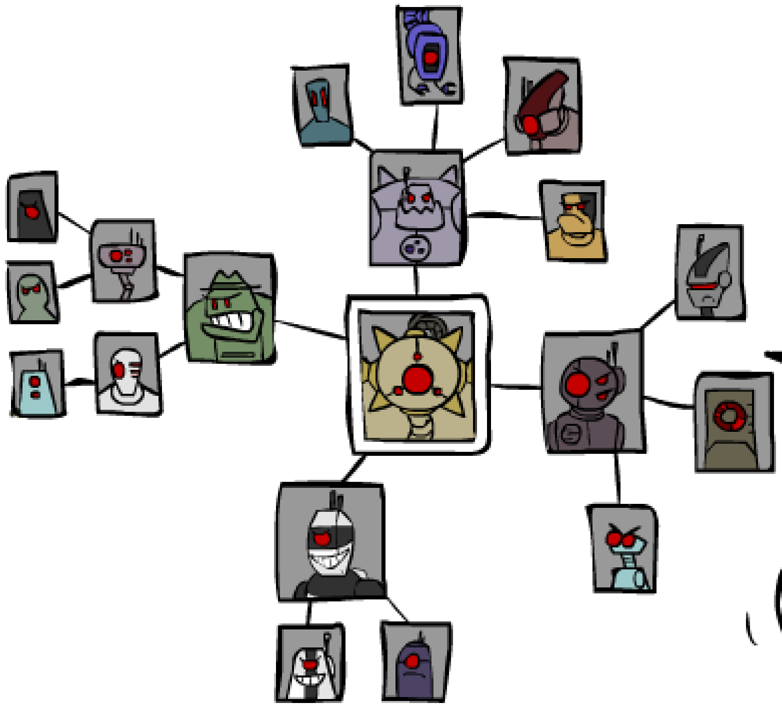


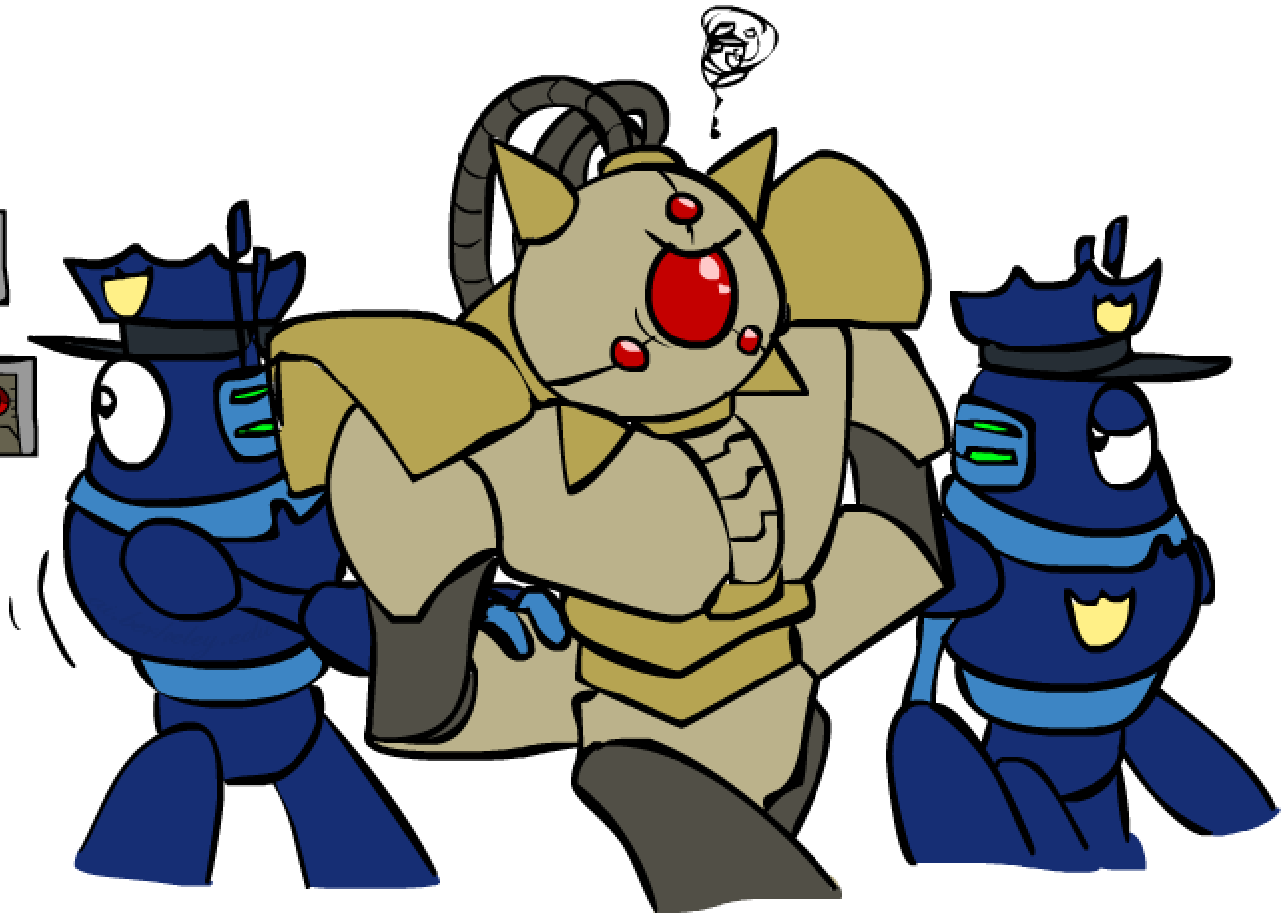
- Theorem: if the constraint graph has no loops, the CSP can be solved in $O(n\ d^2)$ time
  - Compare to general CSPs, where worst-case time is $O(d^n)$

- This property also applies to probabilistic reasoning (later): an example of the relation between syntactic restrictions and the complexity of reasoning

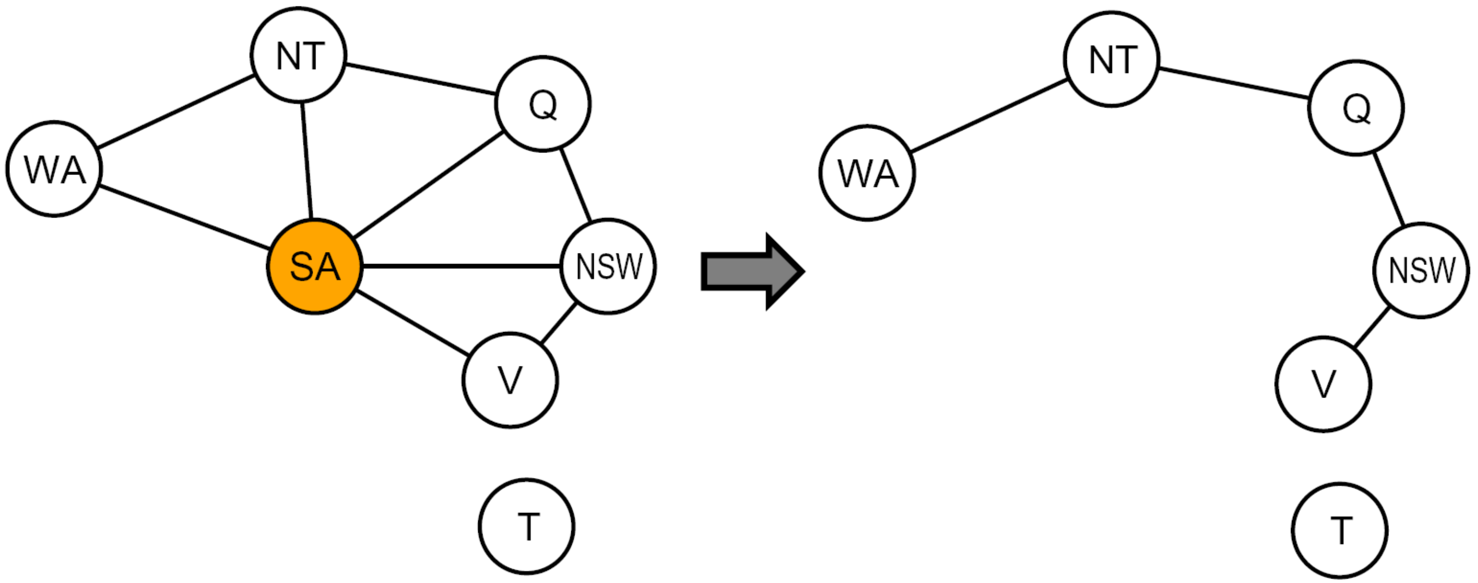
UNIVERSITY OF CALGARY

Improving Structure





Nearly Tree-Structured CSPs



- Conditioning: instantiate a variable, prune its neighbors' domains
- Cutset conditioning: instantiate (in all ways) a set of variables such that the remaining constraint graph is a tree
- Cutset size c gives runtime $O((d^c) (n-c) d^2)$, very fast for small c

UNIVERSITY OF CALGARY

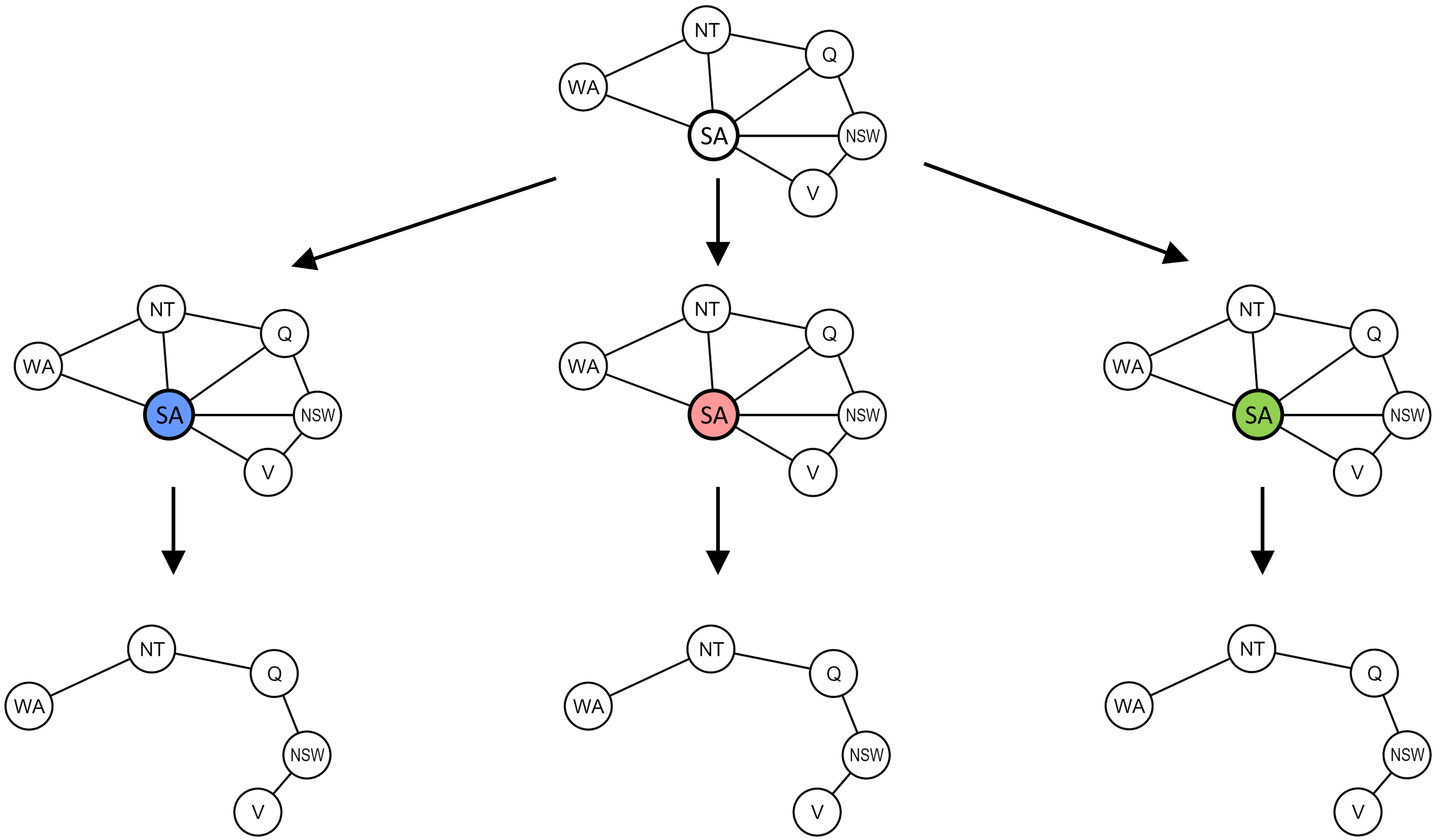
Cutset Conditioning

Choose a cutset

Instantiate the cutset (all possible ways)

Compute residual CSP for each assignment

Solve the residual CSPs (tree structured)



Onward to …
… other search models

Jonathan Hudson
jwhudson@ucalgary.ca
https://pages.cpsc.ucalgary.ca/~jwhudson/

UNIVERSITY OF CALGARY