Artificial Intelligence (CS303)

Lecture 4: Constraint Satisfaction Problems

Hints for this lecture

- **The more we know about the problem characteristic/structure, the better we can solve it.**

Outline of this lecture

- **Make Search Algorithms Less General**
- **What is Constraint Satisfaction Problems**
- **Search Algorithms for CSPs**

I. Make Search Algorithms Less General

Make Search Algorithms Less General

- **The search methods talked previously are to much extent general methods, i.e., applicable to any problem.**
- **Generality is nice and worthy of pursuit, while it usually conflicts with the other desired features of algorithms, e.g., efficiency.**
- **Question: What makes a general search algorithm “general” ?**

Make Search Algorithms Less General

- **The search methods talked previously are to much extent general methods, i.e., applicable to any problem.**
- **Generality is nice and worthy of pursuit, while it usually conflicts with the other desired features of algorithms, e.g., efficiency.**
- **Question: What makes a general search algorithm "general"?**

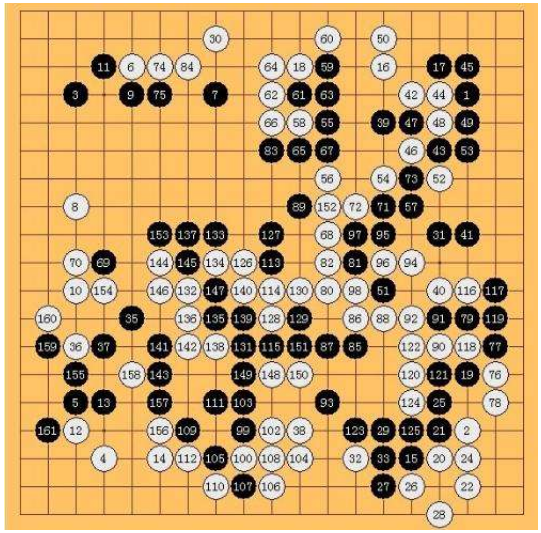
General Representation

Make Search Algorithms Less General

- A representation (e.g. tree) is general because no assumption is made (or nothing is known) about the characteristic/structure of the state space.
- To be realistic, we are usually interested in solving one specific problem (class) rather than all possible problems.
- When designing an algorithm for a problem (class), taking the structure of the state space into account usually requires search only part of the state space, making the search more efficient.

Make Search Algorithms Less General

- **While in many cases, we do know something about problem structure (i.e., not an assumption, but a fact).**
- **For example, a "state" is usually more than an atom.**



For each step, we have the information of all previous steps

Make Search Algorithms Less General

- **General search algorithms are nice. But as long as a problem (class) is of sufficient significance, it is worthy of designing problem-specific algorithm for it.**
- **Constraint Satisfaction Problem is such a problem class.**

II. Constraint Satisfaction Problems

Definition

Standard search problem:

state is a "black box"—any old data structure
that supports goal test, eval, successor

CSP:

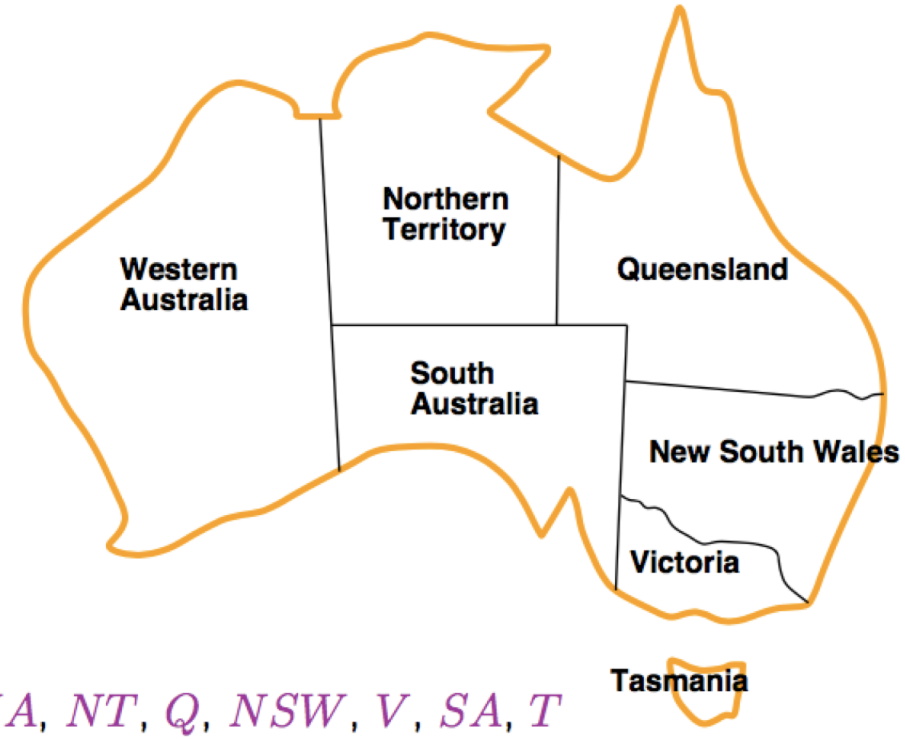
state is defined by variables X_i with values from domain D_i

goal test is a set of constraints specifying
allowable combinations of values for subsets of variables

Simple example of a **formal representation language**

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

Example: Map Coloring



Variables WA, NT, Q, NSW, V, SA, T

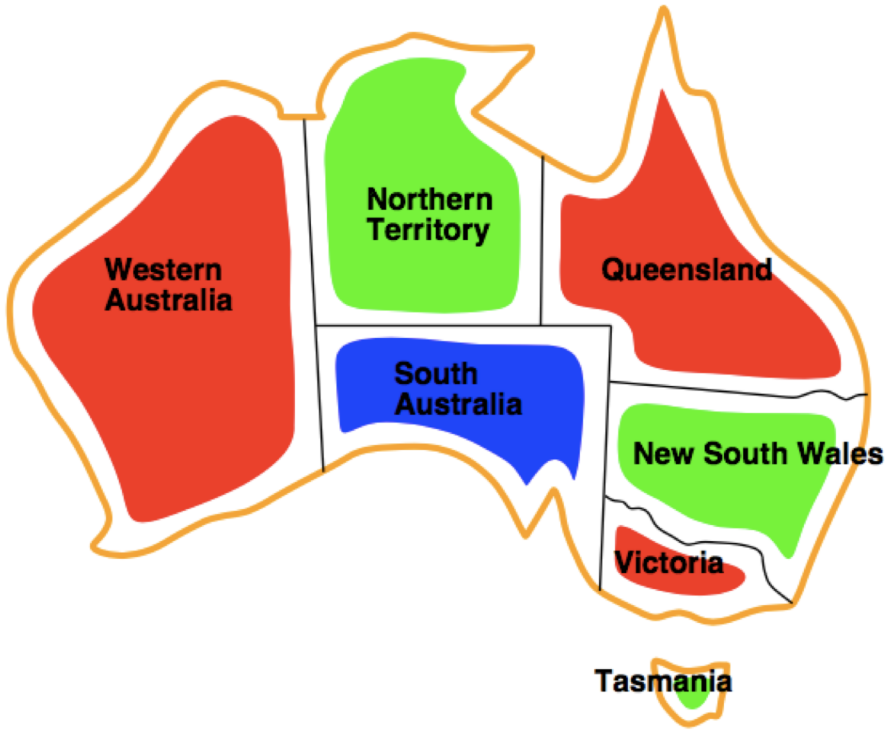
Domains $D_i = \{red, green, blue\}$

Constraints: adjacent regions must have different colors

e.g., $WA \neq NT$ (if the language allows this), or

$(WA, NT) \in \{(red, green), (red, blue), (green, red), (green, blue), \ldots\}$

Example: Map Coloring



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

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

Variants of CSPs

- Unary constraints involve a single variable.
- Binary constraints involve pairs of variables.
- Higher-order constraints involve 3 or more variables.
- Preferences (Soft constraints), e.g., red is better than green, is often represented by a cost for each variable assignment (i.e., the target is to minimize the cost).

Real-world CSPs

- Assignment problems
- Timetabling problems
- Hardware configuration
- Floorplanning
- Factory scheduling
- …

What is the search tree of a CSP?

III. Search Algorithms for CSPs

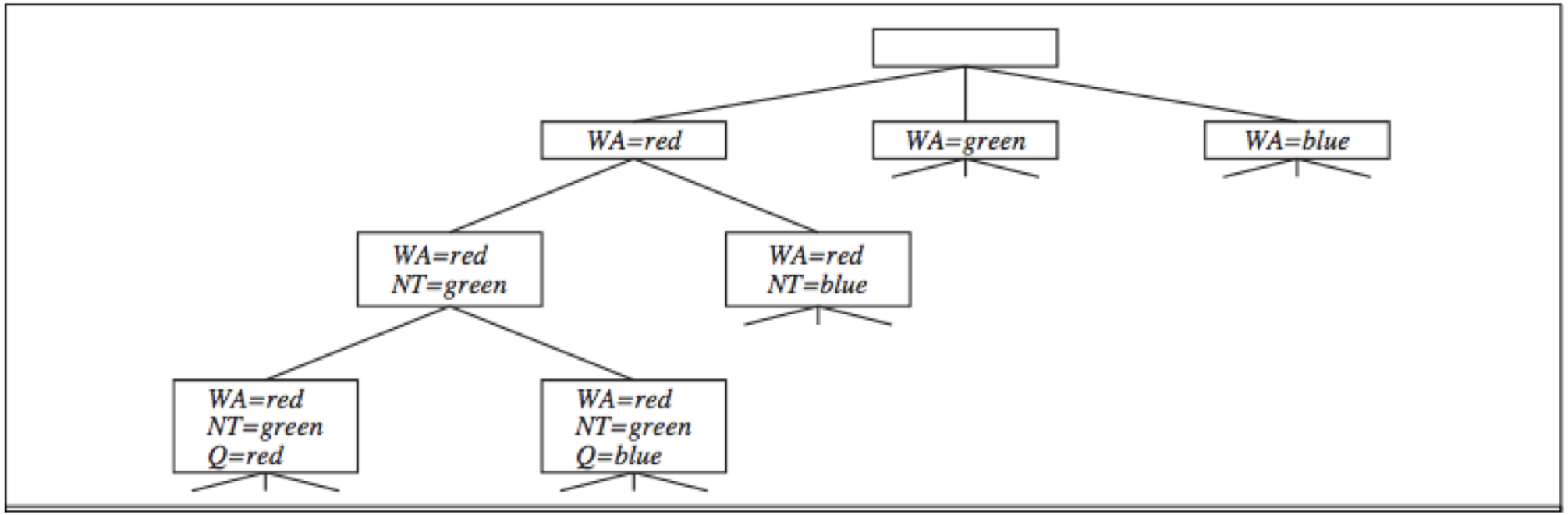
Special characteristics of CSPs

- **Commutativity: the order of assigning values to variables does not affect the final outcome.**
- **The constrains provide additional information that could be represented by a constraint graph.**

Note: We are talking about the characteristics of problems rather than algorithms.

Commutativity

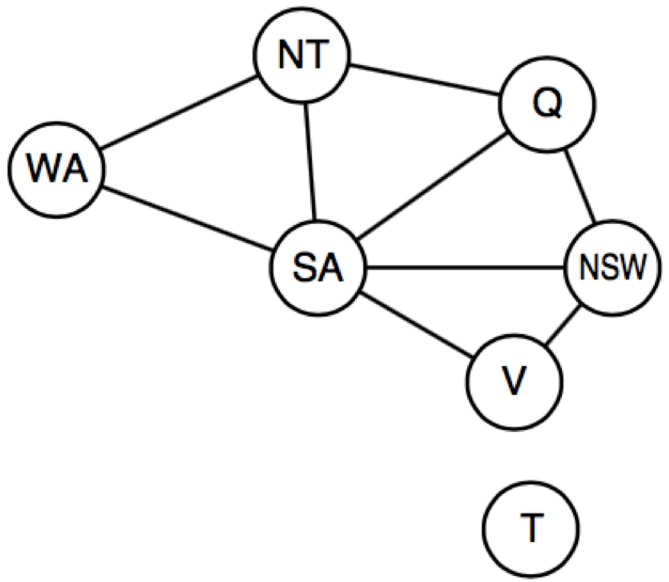
- **Commutativity indicates that only 1 variable needs to be considered at each node in the search tree.**



Constraint Graph

Binary CSP: each constraint relates at most two variables

Constraint graph: nodes are variables, arcs show constraints



General-purpose CSP algorithms use the graph structure
to speed up search. E.g., Tasmania is an independent subproblem!

Inference

- A constraint graph allows the agent to do inference in addition to search.
- Inference basically means checking local consistency (or detecting inconsistency)
  - Node consistency
  - Arc Consistency
  - Path Consistency
  - *K*-consistency
  - Global consistency
- Inference helps prune the search tree, either before or during the search.

Backtracking Search for CSP

- Depth-first search, assign a value to unassigned variables recursively.
- If inconsistency occurred, move 1 step back to try another value.

```
function BACKTRACKING-SEARCH(csp) returns solution/failure
    return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment given CONSTRAINTS[csp] then
            add {var = value} to assignment
            result ← RECURSIVE-BACKTRACKING(assignment, csp)
            if result ≠ failure then return result
            remove {var = value} from assignment
    return failure
```

Improving Backtracking Search (1)

Applying inference

Applying inference

```
function BACKTRACKING-SEARCH(csp) returns solution/failure
    n RECURSIVE-BACKTRACKING({ }, csp)

    n RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
  if assignment is complete then return assignment
  var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
  for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
      if value is consistent with assignment given CONSTRAINTS[csp] then
          add {var = value} to assignment
          result ← RECURSIVE-BACKTRACKING(assignment, csp)
          if result ≠ failure then return result
          remove {var = value} from assignment
  return failure
```

Improving Backtracking Search (2)

```
function BACKTRACKING-SEARCH(csp) returns solution/failure
    return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment given CONSTRAINTS[csp] then
            add {var = value} to assignment
            result ← RECURSIVE-BACKTRACKING(assignment, csp)
            if result ≠ failure then return result
            remove {var = value} from assignment
    return failure
```

Choosing variables with minimum numbers of remaining value

Improving Backtracking Search (3)

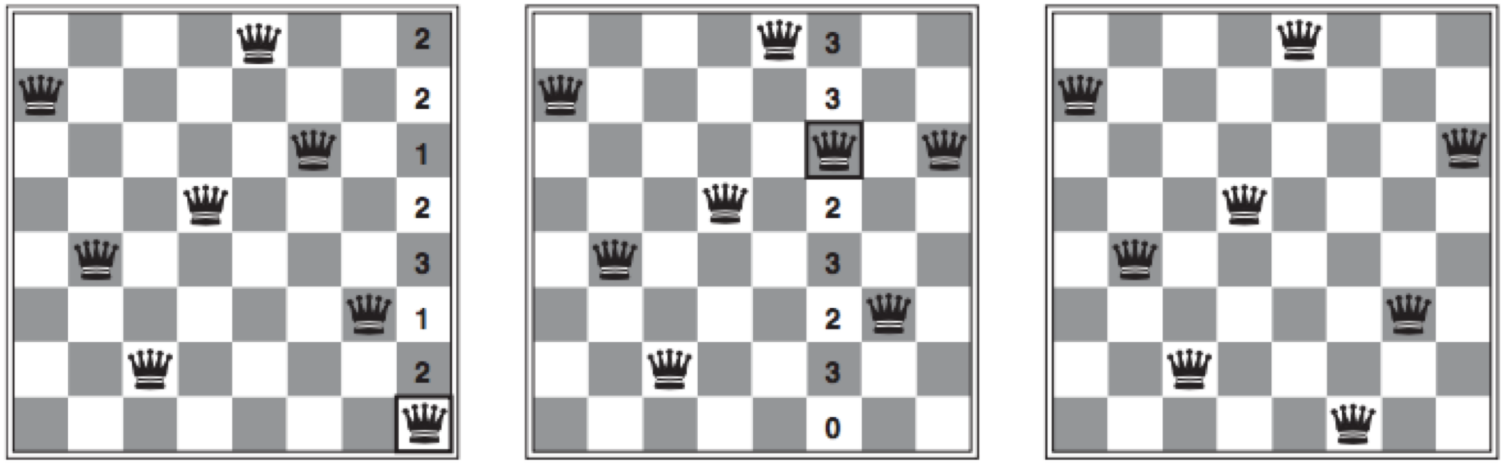
```
function BACKTRACKING-SEARCH(csp) returns solution/failure
    return RECURSIVE-BACKTRACKING({ }, csp)

function RECURSIVE-BACKTRACKING(assignment, csp) returns soln/failure
    if assignment is complete then return assignment
    var ← SELECT-UNASSIGNED-VARIABLE(VARIABLES[csp], assignment, csp)
    for each value in ORDER-DOMAIN-VALUES(var, assignment, csp) do
        if value is consistent with assignment given CONSTRAINTS[csp] then
            add {var = value} to assignment
            result ← RECURSIVE-BACKTRACKING(assignment, csp)
            if result ≠ failure then return result
            remove {var = value} from assignment
    return failure
```

Maintain a conflict set and do backjumping

Local Search for CSP

- CSP can be actually reformulated as a constraint optimization problem, for which the objective function is to minimize the constraint violation.
- Working in the solution space (complete solution formulation)
- Iteratively select a conflicted variable and assign a different value to it.
- Choose the value that leads to the minimum cost.



To be continued