Constraint Satisfaction Problems (CSP)
(Where we postpone making difficult decisions until they become easy to make)

R&N: Chap. 5
Slides from Jean-Claude Latombe at Stanford University (used with permission)

What we will try to do ....

- Search techniques make choices in some order which often is arbitrary. Often little state information is available to make each of them (states are "black boxes")
- In many problems, the same states can be reached independent of the order in which choices are made ("commutative" actions)
- Can we solve such problems more efficiently by picking the order appropriately? Can we even avoid making any choice? Do we have all the information needed?

Constraint Propagation

- Place a queen in a square
- Remove the attacked squares from future consideration
Constraint Propagation

- Count the number of non-attacked squares in every row and column
- Place a queen in a row or column with minimum number
- Remove the attacked squares from future consideration

Constraint Propagation

- Repeat
Constraint Propagation

What do we need?

- More than just a successor function and a goal test
- We also need:
  - A means to propagate the constraints imposed by one queen's position on the positions of the other queens
  - An early failure test
- Explicit representation of constraints
- Constraint propagation algorithms
**Constraint Satisfaction Problem (CSP)**
- Set of variables \( \{X_1, X_2, \ldots, X_n\} \)
- Each variable \( X_i \) has a domain \( D_i \) of possible values. Usually, \( D_i \) is finite.
- Set of constraints \( \{C_1, C_2, \ldots, C_p\} \)
- Each constraint relates a subset of variables by specifying the valid combinations of their values.
- Goal: Assign a value to every variable such that all constraints are satisfied.

**Map Coloring**
- 7 variables \( \{WA, NT, SA, Q, NSW, V, T\} \)
- Each variable has the same domain: \( \{\text{red, green, blue}\} \)
- No two adjacent variables have the same value:
  - \( WA \neq NT, WA \neq SA, NT \neq SA, NT \neq Q, SA \neq Q, SA \neq NSW, SA \neq V, Q \neq NSW, NSW \neq V \)

**8-Queen Problem**
- 8 variables \( X_i, i = 1 \) to 8
- The domain of each variable is: \( \{1, 2, \ldots, 8\} \)
- Constraints are of the forms:
  - \( X_i = k \Rightarrow X_j \neq k \) for all \( j = 1 \) to 8, \( j \neq i \)
  - Similar constraints for diagonals
- All constraints are binary.
Street Puzzle

Ni = (English, Spaniard, Japanese, Italian, Norwegian)
Ci = (Red, Green, White, Yellow, Blue)
Di = (Tea, Coffee, Milk, Fruit-juice, Water)
Ji = (Painter, Sculptor, Diplomat, Violinist, Doctor)
Ai = (Dog, Snails, Fox, Horse, Zebra)

The Englishman lives in the Red house
The Spaniard has a Dog
The Japanese is a Painter
The Italian drinks Tea
The Norwegian lives in the first house on the left
The owner of the Green house drinks Coffee
The Green house is on the right of the White house
The Sculptor breeds Snails
The Diplomat lives in the Yellow house
The owner of the middle house drinks Milk
The Norwegian lives next door to the Blue house
The Violinist drinks Fruit juice
The Horse is in the house next to the Diplomat’s
Who owns the Zebra?
Who drinks Water?

Who owns the Zebra?
Who drinks Water?

Who owns the Zebra?
Who drinks Water?

Who owns the Zebra?
Who drinks Water?
Street Puzzle

N = {English, Spaniard, Japanese, Italian, Norwegian}
C = {Red, Green, White, Yellow, Blue}
D = {Tea, Coffee, Milk, Fruit-juice, Water}
J = {Painter, Sculptor, Diplomat, Violinist, Doctor}
A = {Dog, Snails, Fox, Horse, Zebra}

The Englishman lives in the Red house
The Spaniard has a Dog
The Japanese is a Painter
The Italian drinks Tea
The Norwegian lives in the first house on the left
The owner of the Green house drinks Coffee
The Green house is on the right of the White house
The Sculptor breeds Snails
The Diplomat lives in the Yellow house
The owner of the middle house drinks Milk
The Norwegian lives next door to the Blue house
The Violinist drinks Fruit juice
The Fox is in the house next to the Doctor's
The Horse is next to the Diplomat's

Unary constraints
Task Scheduling

Four tasks $T_1, T_2, T_3$, and $T_4$ are related by time constraints:

- $T_1$ must be done during $T_3$
- $T_3$ must be achieved before $T_4$ starts
- $T_2$ must overlap with $T_3$
- $T_4$ must start after $T_1$ is complete

Are the constraints compatible?

What are the possible time relations between two tasks?

What if the tasks use resources in limited supply?

How to formulate this problem as a CSP?

3-SAT

- $n$ Boolean variables $u_1, ..., u_n$
- $p$ constrains of the form $u_i^* \lor u_j^* \lor u_k^* = 1$
- where $u^*$ stands for either $u$ or $\neg u$

Known to be NP-complete

Finite vs. Infinite CSP

- **Finite CSP**: each variable has a finite domain of values
- **Infinite CSP**: some or all variables have an infinite domain

E.g., linear programming problems over the reals:

- $\sum_{i=1}^{\infty} a_i x_i = 0$
- $\sum_{i=1}^{\infty} b_i x_i = 0$

We will only consider finite CSP.
CSP as a Search Problem

- $n$ variables $X_1, \ldots, X_n$
- Valid assignment: $(X_{i_1} = v_{i_1}, \ldots, X_{i_k} = v_{i_k})$ for $0 \leq k \leq n$ such that the values $v_{i_1}, \ldots, v_{i_k}$ satisfy all constraints relating the variables $X_{i_1}, \ldots, X_{i_k}$
- Complete assignment: one where $k = n$
  - [if all variable domains have size $d$, there are $O(d^n)$ complete assignments]
- States: valid assignments
- Initial state: empty assignment $\emptyset$, i.e., $k = 0$
- Successor of a state:
  - $(X_{i_1} = v_{i_1}, \ldots, X_{i_k} = v_{i_k}) \rightarrow (X_{i_1} = v_{i_1}, \ldots, X_{i_k} = v_{i_k}, X_{i_{k+1}} = v_{i_{k+1}})$
- Goal test: $k = n$

A Key property of CSP: Commutativity

The order in which variables are assigned values has no impact on the reachable complete valid assignments.

Hence:

1) One can expand a node $N$ by first selecting one variable $X$ not in the assignment $A$ associated with $N$ and then assigning every value $v$ in the domain of $X$.

[\Rightarrow \text{big reduction in branching factor}]
- 4 variables $X_1, \ldots, X_4$
- Let the valid assignment of $N$ be:
  \[ A = \{ X_1 \gets v_1, X_3 \gets v_3 \} \]
- For example pick variable $X_4$
- Let the domain of $X_4$ be $(v_{41}, v_{42}, v_{43})$
- The successors of $A$ are all the valid assignments among:
  \[ \{ X_1 \gets v_1, X_3 \gets v_3, X_4 \gets v_{41} \} \]
  \[ \{ X_1 \gets v_1, X_3 \gets v_3, X_4 \gets v_{42} \} \]
  \[ \{ X_1 \gets v_1, X_3 \gets v_3, X_4 \gets v_{43} \} \]

1) One can expand a node $N$ by first selecting one variable $X$ not in the assignment $A$ associated with $N$ and then assigning every value $v$ in the domain of $X$.

2) One need not store the path to a node.

A Key property of CSP: Commutativity

The order in which variables are assigned values has no impact on the reachable complete valid assignments.

Hence:

- One can expand a node $N$ by first selecting one variable $X$ not in the assignment $A$ associated with $N$ and then assigning every value $v$ in the domain of $X$.

\[ \rightarrow \text{big reduction in branching factor} \]

- One need not store the path to a node.

\[ \rightarrow \text{Backtracking search algorithm} \]
Backtracking Search

Essentially a simplified depth-first algorithm using recursion

Assignment = {}
Backtracking Search
(3 variables)

Assignment = \{(X_1, v_{11}), (X_3, v_{31})\}

Then, the search algorithm backtracks to the previous (X_3) variable and tries another value.

Assignment = \{(X_1, v_{11}), (X_2, v_{22})\}

Assume that no value of X_2 leads to a valid assignment.

Assignment = \{(X_1, v_{11}), (X_3, v_{32})\}
The search algorithm backtracks to the previous variable (X3) and tries another value. But assume that X3 has only two possible values. The algorithm backtracks to X1.

Assume again that no value of X2 leads to a valid assignment.

Backtracking Search
(3 variables)
Assignment = {(X1,v12), (X2,v21)}
Assignment = {(X1,v12), (X3,v32)}
Backtracking Search (3 variables)

Assignment: \{(X_1, v_{12}), (X_2, v_{21})\}

The algorithm need not consider the variables in the same order in this sub-tree as in the other.

Assignment: \{(X_1, v_{12}), (X_2, v_{21}), (X_3, v_{32})\}

The algorithm need not consider the values of X_3 in the same order in this sub-tree.
Backtracking Search

(3 variables)

Assignment = \{(X_1, v_{12}), (X_2, v_{21}), (X_3, v_{32})\}

Since there are only three variables, the assignment is complete.

Backtracking Algorithm

CSP-BACKTRACKING(A)
1. If assignment A is complete then return A
2. X \leftarrow select a variable not in A
3. D \leftarrow select an ordering on the domain of X
4. For each value v in D do
   a. Add (X \leftarrow v) to A
   b. If A is valid then
      i. result \leftarrow CSP-BACKTRACKING(A)
      ii. If result \neq failure then return result
   c. Remove (X \leftarrow v) from A
5. Return failure

Call CSP-BACKTRACKING(\{}\)