Constraint Satisfaction
Moving to a different formalism...

\[
\begin{array}{c}
\text{SEND} \\
+ \text{ MORE} \\
\hline
\text{MONEY}
\end{array}
\]

Consider state space for cryptarithmetic (e.g. DFS).

Is this (DFS) how humans tackle the problem?

Human problem solving appears more sophisticated! For example, we derive new constraints on the fly.
→ little or no search!
Constraint Satisfaction Problems (CSP)

A powerful representation for (discrete) search problems

A **Constraint Satisfaction Problem (CSP)** is defined by:
- \( X \) is a set of \( n \) variables \( X_1, X_2, \ldots, X_n \) each defined by a finite domain \( D_1, D_2, \ldots, D_n \) of possible values.

- \( C \) is a set of constraints \( C_1, C_2, \ldots, C_m \). Each \( C_i \) involves a subset of the variables; specifies the allowable combinations of values for that subset.

A **solution** is an assignment of values to the variables that satisfies all constraints.
Cryptarithmetic as a CSP

Variables:

\[ T \in \{0,\ldots,9\}; \quad W \in \{0,\ldots,9\}; \quad O \in \{0,\ldots,9\}; \]
\[ F \in \{0,\ldots,9\}; \quad U \in \{0,\ldots,9\}; \quad R \in \{0,\ldots,9\}; \]
\[ X_1 \in \{0,\ldots,1\}; \quad X_2 \in \{0,\ldots,1\}; \quad X_3 \in \{0,\ldots,1\}; \quad \leftarrow \text{Auxiliary variables} \]

Constraints:

\[ O + O = R + 10 \times X_1 \]
\[ X_1 + W + W = U + 10 \times X_2 \]
\[ X_2 + T + T = O + 10 \times X_3 \]
\[ X_3 = F \]

each letter has a different digit (F \neq T, F \neq U, etc.);
Types of Constraints

**Unary Constraints:**
Restriction on single variable

**Binary Constraints:**
Restriction on pairs of variables

**Higher-Order Constraints:**
Restriction on more than two variables

**Preferences vs. Constraints**
Constraint Hypergraph

\[
\begin{array}{ccc}
\text{TWO} & + & \text{TWO} \\
\hline
\text{FOUR}
\end{array}
\]
Types of variables

• Discrete domains
  – Boolean \{T,F\} $\leftrightarrow$ 3-Sat, K-Sat
  – Finite domains \{a,b,c...\}
  – Infinite (e.g. all integers)
    • constraints represented using language,
    • e.g. $X < Y$, $Y > Z + 5$

• Continuous domains
  – Linear $\leftrightarrow$ linear programming
  – Nonlinear
Constraint Satisfaction Problems (CSP)

For a given CSP the problem is one of the following:

1. find all solutions

2. find one solution
   · just a feasible solution, or
   · A “reasonably good” feasible solution, or
   · the optimal solution given an objective

3. determine if a solution exists
How to View a CSP as a Search Problem?

**Initial State** - state in which all the variables are unassigned.

**Successor function** - assign a value to a variable from a set of possible values.

**Goal test** - check if all the variables are assigned and all the constraints are satisfied.

**Path cost** - assumes constant cost for each step
Branching Factor

**Approach 1** - any unassigned variable at a given state can be assigned a value by an operator: branching factor as high as sum of size of all domains.

**Approach 2** - since order of variable assignment not relevant, consider as the successors of a node just the different values of a *single* unassigned variable: max branching factor = max size of domain.

Prefer BFS or DFS?

\[
B = \text{BFS} \quad D = \text{DFS}
\]
CSP – Goal Decomposed into Constraints

Backtracking Search: a DFS that

• chooses values for variables one at a time
• checks for consistency with the constraints.

Decisions during search:

• Which variable to choose next for assignment.
• Which value to choose next for the variable.
Minimum Remaining Values (MRV)

- Idea: Assign most constrained variable first
- Prune impossible assignments fairly early
- Degree heuristic: choose higher degree first

Which is best order according to MRV heuristic?

- A = NT, SA, WA, Q, NSW, V, T
- B = T, V, SA, NSW, WA, NT, Q
- C = SA, Q, NSW, V, NT, WA, T
Forward Checking

- **Idea:** Reduce domain of unassigned variables based on assigned variables.
  - Each time variable is instantiated, delete from domains of the uninstantiated variables all of those values that conflict with current variable assignment.
- Identify dead ends without having to try them via backtracking
  - E.g. if last variable has zero options, no need to go that deep to find out
General Purpose Heuristics

Variable and value ordering:

Degree heuristic: assign a value to the variable that is involved in the largest number of constraints on other unassigned variables.

Minimum remaining values (MRV): choose the variable with the fewest possible values.

Least-constraining value heuristic: choose a value that rules out the smallest number of values in variables connected to the current variable by constraints.
# Comparison of CSP Algorithms

<table>
<thead>
<tr>
<th>Problem</th>
<th>BT</th>
<th>BT+MRV</th>
<th>BT+FC</th>
<th>BT+FC+MRV</th>
</tr>
</thead>
<tbody>
<tr>
<td>USA</td>
<td>(&gt;1,000K)</td>
<td>(&gt;1,000K)</td>
<td>2K</td>
<td>60</td>
</tr>
<tr>
<td>N-queens</td>
<td>(&gt;40,000K)</td>
<td>13,500K</td>
<td>(&gt;40,000K)</td>
<td>817K</td>
</tr>
</tbody>
</table>
**Constraint Propagation (Arc Consistency)**

- **Arc Consistency** - state is arc-consistent, if every variable has some value that is consistent with each of its constraints (consider pairs of variables)

<table>
<thead>
<tr>
<th></th>
<th>WA</th>
<th>NT</th>
<th>Q</th>
<th>NSW</th>
<th>V</th>
<th>SA</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>Initial Domains</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
</tr>
<tr>
<td>After WA=red</td>
<td>R</td>
<td></td>
<td>GB</td>
<td>RGB</td>
<td>RGB</td>
<td>GB</td>
<td>RGB</td>
</tr>
<tr>
<td>After Q=green</td>
<td>R</td>
<td></td>
<td>G</td>
<td>RGB</td>
<td>RGB</td>
<td>B</td>
<td>RGB</td>
</tr>
<tr>
<td>After V=blue</td>
<td>R</td>
<td></td>
<td>G</td>
<td>R</td>
<td>B</td>
<td></td>
<td>RGB</td>
</tr>
</tbody>
</table>
Constraint Propagation (Arc Consistency)

- **Arc Consistency** - state is arc-consistent, if every variable has some value that is consistent with each of its constraints (consider pairs of variables)

```plaintext
Init: Q is queue with all (directed) arcs (X_i,X_j) in CSP
WHILE Q is not empty
- (X_i,X_j) = remove_first(Q)
- FOREACH x \in \text{dom}(X_i)
  *IF no y \in \text{dom}(X_j) satisfies constraint (X_i,X_j)
  · THEN remove x from \text{dom}(X_i)
- IF \text{dom}(X_i) changed
  *THEN add all arcs (X_k, X_i) \notin Q to Q
```
Example: Arc Consistency

Task: 3-color

Solution:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>A=R</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
<td>RGB</td>
</tr>
<tr>
<td>B=G</td>
<td>(R)</td>
<td>GB</td>
<td>RBG</td>
<td>RBG</td>
<td>GB</td>
<td>GB</td>
</tr>
</tbody>
</table>

\[ D \neq F : D = \{ R, B \} \]
\[ E \neq F : E = \{ G, B \} \]
\[ C \neq D : C = \{ R, B \} \]
Constraint Propagation (K-Consistency)

- **K-Consistency** generalizes arc-consistency (2-consistency).
- Consistency of groups of K variables.
- Path consistency
Constraint learning

• When assignments fail, is there a way to learn new constraints?
  – Conflict-directed back-jumping looks to find the root cause of a failure and adds it as a new constraint
More substructure: Symmetries
Local Search for CSPs

![Diagram showing local search for CSPs with chessboards and numbers]
Remarks

Dramatic recent progress in Constraint Satisfaction. Methods can now handle problems with **10,000** to **100,000** variables, and up to **1,000,000** constraints.