Constraint Satisfaction and Backtrack Search

22c:31 Algorithms

Overview
- Constraint Satisfaction Problems (CSP) share some common features and have specialized methods
  - View a problem as a set of variables to which we have to assign values that satisfy a number of problem-specific constraints.
  - Constraint solvers, constraint logic programming…
- Algorithms for CSP
  - Backtracking (systematic search)
  - Variable ordering heuristics

Informal Definition of CSP
- CSP = Constraint Satisfaction Problem
- Given \( \langle V, D, C \rangle \)
  1. \( V \): a finite set of variables
  2. \( D \): a domain of possible values (often finite)
  3. \( C \): a set of constraints that limit the values the variables can take on
- A solution is an assignment of a value to each variable such that all the constraints are satisfied.
- Tasks might be to decide if a solution exists, to find a solution, to find all solutions, or to find the “best solution” according to some metric \( f \).

Example: Path of Length \( k \)
- Given an undirected graph \( G = (V, E) \), does \( G \) have a simple path of length \( k \)?
- Variables: \( x_0, x_1, \ldots, x_k \)
- Domain of variables: \( V \)
- Constraints:
  - (a) all values to \( x_i \) are distinct;
  - (b) \( (x_i, x_{i+1}) \) is in \( E \).

Example: Vertex Cover of Size \( k \)
- Given an undirected graph \( G = (V, E) \), does \( G \) have a vertex cover of size \( k \)?
- Variables: \( X = \{ x_1, x_2, \ldots, x_n \} \), where \( n = |V| \)
- Domain of variables: \( \{ \text{true}, \text{false} \} \)
- Constraints:
  - (a) \( x_i \) is in the vertex cover if \( x_i \) is true;
  - (b) For each edge \( (a, b) \) of \( E \), \( a \) or \( b \) is in \( X \).
Example: Clique of Size k

- Given an undirected graph $G = (V, E)$, does $G$ have a clique of size $k$?
- Variables: $X = \{ x_1, x_2, \ldots, x_k \}$
- Domain of variables: $V$
- Constraints:
  - (a) all values to $x_i$ are distinct;
  - (b) $(x_i, x_j)$ is in $E$ for any $i$ and $j$.

Example: Map Coloring

- Color the following map using three colors (red, green, blue) such that no two adjacent regions have the same color.

- Variables: $A, B, C, D, E$ all of domain RGB
- Domains: RGB = \{red, green, blue\}
- Constraints:
  - $A \neq B, A \neq C, A \neq E, A \neq D, B \neq C, C \neq D, D \neq E$
- One solution: $A$=red, $B$=green, $C$=blue, $D$=green, $E$=blue

N-queens Example (4 in our case)

- Standard test case in CSP research
- Variables are the rows: $r_1, r_2, r_3, r_4$
- Values are the columns: \{1, 2, 3, 4\}
- So, the constraints include:
  - $C_{r_1, r_2} = \{(1,3),(1,4),(2,4),(3,1),(4,1),(4,2)\}$
  - $C_{r_1, r_3} = \{(1,2),(1,4),(2,1),(2,3),(3,2),(3,4), (4,1),(4,3)\}$
  - Etc.
  - What do these constraints mean?

Example: SATisfiability

- Given a set of propositional variables and Boolean formulas, find an assignment of the variables to \{false, true\} that satisfies the formulas.
- Example:
  - Boolean variables = \{ A, B, C, D\}
  - Boolean formulas: $A \lor B \lor \neg C, \neg A \lor D, B \lor C \lor D$
  - (the first two equivalent to $C \Rightarrow A \lor B, A \Rightarrow D$)
  - Are satisfied by
    - $A$ = false
    - $B$ = true
    - $C$ = false
    - $D$ = false

Real-world problems

- Scheduling
- Temporal reasoning
- Building design
- Planning
- Optimization/satisfaction
- Vision
- Graph layout
- Network management
- Natural language processing
- Molecular biology / genomics
- VLSI design
A constraint satisfaction problem (CSP) consists of:

- a set of variables \( X = \{x_1, x_2, \ldots, x_n\} \)
  - each with an associated domain of values \( \{d_1, d_2, \ldots, d_n\} \).
  - The domains are typically finite
- a set of constraints \( \{c_1, c_2, \ldots, c_m\} \) where
  - each constraint defines a predicate which is a relation over a particular subset of \( X \).
  - E.g., \( c_i \) involves variables \( \{X_{i1}, X_{i2}, \ldots, X_{ik}\} \) and defines the relation \( R_i \subseteq D_{i1} \times D_{i2} \times \cdots \times D_{ik} \)

Instantiations:
- An instantiation of a subset of variables \( S \) is an assignment of a domain value to each variable in \( S \).
- An instantiation is legal iff it does not violate any (relevant) constraints.
- A solution is a legal instantiation of all of the variables in the network.

Typical Tasks for CSP:

- Solutions:
  - Does a solution exist?
  - Find one solution
  - Find all solutions
  - Given a partial instantiation, do any of the above
- Transform the CSP into an equivalent CSP that is easier to solve.

Constraint Solving is Hard:

Constraint solving is not possible for general constraints.
Example (Fermat's Last Theorem):
\[
C: \quad n > 2
\]
\[
C': \quad a^n + b^n = c^n
\]

Constraint programming separates constraints into
- basic constraints: complete constraint solving
- non-basic constraints: propagation (incomplete); search needed

Systematic search: Backtracking (backtrack depth-first search):

- Consider the variables in some order
- Pick an unassigned variable and give it a provisional value such that it is consistent with all of the constraints
- If no such assignment can be made, we've reached a dead end and need to backtrack to the previous variable
- Continue this process until a solution is found or we backtrack to the initial variable and have exhausted all possible values.

  - DFS: When backtrack, a node is marked as "processed"
  - CSP: When backtrack, a node is unmarked as "discovered"

Backtrack Procedure:

\[
\text{Backtrack}(a, k) \quad \{ \text{ // A is a vector of length k} \\
\text{if } (k > n) \quad \{ \\
\text{if } (A \text{ is a solution) print(A)} \\
\text{else} \quad \{ \\
\text{compute } S_k \quad \text{the domain of } a_k \\
\text{while } S_k \not= \text{empty do} \\
\text{A}_k = \text{ an element in } S_k \\
\text{S}_k = S_k - \{A_k\} \\
\text{Backtrack}(a, k+1) \\
\text{\} \} \}
\]
Problems with backtracking

- Thrashing: keep repeating the same failed variable assignments
  - Consistency checking can help
  - Intelligent backtracking schemes can also help
- Inefficiency: can explore areas of the search space that aren’t likely to succeed
  - Variable ordering can help

Improving backtracking efficiency

- General-purpose methods can give huge gains in speed:
  - Which variable should be assigned next?
  - In what order should its values be tried?
  - Can we detect inevitable failure early?
**Most constrained variable**

- Most constrained variable: choose the variable with the fewest legal values

  - a.k.a. minimum remaining values (MRV) heuristic

---

**Most constraining variable**

- Tie-breaker among most constrained variables
- Most constraining variable:
  - choose the variable with the most constraints on remaining variables

---

**Least constraining value**

- Given a variable, choose the least constraining value:
  - the one that rules out the fewest values in the remaining variables

---

**Symmetry Breaking**

Often, the most efficient model admits many different solutions that are essentially the same (“symmetric” to each other).

Symmetry breaking tries to improve the performance of search by eliminating such symmetries.

---

**Example: Map Coloring**

- Variables: A, B, C, D, E all of domain RGB
- Domains: RGB = {red, green, blue}
- Constraints: A ≠ B, A ≠ C, A ≠ E, A ≠ D, B ≠ C, C ≠ D, D ≠ E
- One solution: A = red, B = green, C = blue, D = green, E = blue

---

**Performance of Symmetry Breaking**

- All solution search: Symmetry breaking usually improves performance; often dramatically
- One solution search: Symmetry breaking may or may not improve performance
The Sudoku Puzzle

- Number place
- Main properties
  - NP-complete [Yato 03]
  - Well-formed Sudoku: has 1 solution [Simonis 05]
- Minimal Sudoku
  - In a 9x9 Sudoku: smallest known number of givens is 17 [Royle]
- Symmetrical puzzles
  - Many axes of symmetry
  - Position of the givens, not their values
  - Often makes the puzzle non-minimal
- Level of difficulties
  - Varied ranking systems exist
  - Minimality and difficulty not related

Sudoku as a CSP

- Variables are the cells
- Domains are sets \{1,\ldots,9\}
- Two models
  - Binary constraints: 810 all-different binary constraint between variables
  - Non-binary constraints: 27 all-different 9-ary constraints

Solving Sudoku as a CSP

- Search
  - Builds solutions by enumerating consistent combinations
- Constraint propagation
  - Removes values that do not appear in a solution
  - Example, arc-consistency:

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Search

- Backtrack search
  - Systematically enumerate solutions by instantiating one variable after another
  - Backtracks when a constraint is broken
  - Is sound and complete (guaranteed to find solution if one exists)
- Propagation
  - Cleans up domain of ‘future’ variables, during search, given current instantiations

The way people play

- ‘Cross-hash,’ sweep through lines, columns, and blocks
- Pencil in possible positions of values
- Generally, look for patterns, some intricate, and give them funny names:
  - Naked single, locked pairs, swordfish, medusa, etc.
- ‘Identified’ dozens of strategies
  - Many fall under a unique constraint propagation technique
- But humans do not seem to be able to carry simple inference (i.e., propagation) in a systematic way for more than a few steps

SEND + MORE = MONEY

Assign distinct digits to the letters
\[ S, E, N, D, M, O, R, Y \]
such that
\[
\begin{align*}
S & \text{ E N D} \\
+ & \text{ M O R E} \\
\hline
& \text{ M O N E Y}
\end{align*}
\]
holds.
SEND + MORE = MONEY

Assign distinct digits to the letters
S, E, N, D, M, O, R, Y
such that
SEND + MORE = MONEY
holds.

Solution

\[
\begin{align*}
S &= 9 \\
E &= 5 \\
N &= 6 \\
D &= 7 \\
M &= 1 \\
O &= 0 \\
R &= 8 \\
Y &= 2
\end{align*}
\]

Modeling

Formalize the problem as a CSP:

- Variables: \(v_1, \ldots, v_n\)
- Domains: \(Z\), integers
- Constraints: \(c_1, \ldots, c_m\) \(\subseteq \mathbb{Z}^n\)
- Problem: Find \(a = (v_1, \ldots, v_n) \in \mathbb{Z}^n\) such that \(a \in c_i\), for all \(1 \leq i \leq m\)

A Model for MONEY

- Variables: \(\{S, E, N, D, M, O, R, Y\}\)
- Constraints:
  1. \(c_1 = \{(S, E, N, D, M, O, R, Y) \in \mathbb{Z}^8 \mid 0 \leq S, \ldots, Y \leq 9\}\)
  2. \(c_2 = \{(S, E, N, D, M, O, R, Y) \in \mathbb{Z}^8 \mid 1000S + 100E + 10N + D + 1000M + 100O + 10R + E = 10000M + 1000O + 100N + 10E + Y\}\)
  3. \(c_3 = \{(S, E, N, D, M, O, R, Y) \in \mathbb{Z}^8 \mid S \neq 0\}\)
  4. \(c_4 = \{(S, E, N, D, M, O, R, Y) \in \mathbb{Z}^8 \mid M \neq 0\}\)
  5. \(c_5 = \{(S, E, N, D, M, O, R, Y) \in \mathbb{Z}^8 \mid S, \ldots, Y \text{ all different}\}\)

Solution for MONEY

\[
\begin{align*}
c_1 &= \{(S, E, N, D, M, O, R, Y) \in \mathbb{Z}^8 \mid 0 \leq S, \ldots, Y \leq 9\}\) \\
c_2 &= \{(S, E, N, D, M, O, R, Y) \in \mathbb{Z}^8 \mid 1000S + 100E + 10N + D + 1000M + 100O + 10R + E = 10000M + 1000O + 100N + 10E + Y\}\)
\end{align*}
\]

Solution:
(9, 5, 6, 7, 1, 0, 8, 2) \(\in \mathbb{Z}^8\)
Propagate

\[
\begin{align*}
\text{SEND} + \text{MORE} &= \text{MONEY} \\
S &\in \{0 \ldots 9\} \\
E &\in \{0 \ldots 9\} \\
N &\in \{0 \ldots 9\} \\
D &\in \{0 \ldots 9\} \\
M &\in \{0 \ldots 9\} \\
O &\in \{0 \ldots 9\} \\
R &\in \{0 \ldots 9\} \\
Y &\in \{0 \ldots 9\} \\
\end{align*}
\]

\[
1000S + 100E + 10N + D \\
+ 1000M + 100O + 10R + E \\
= 10000M + 1000O + 100N + 10E + Y
\]
Optimization Problem

- Let CSP = (V, D, C)
  1. V: a finite set of variables
  2. D: a domain of possible values (often finite)
  3. C: a set of constraints that limit the values the variables can take on
- Define a numeric function \( f(V) \)
- A solution is an assignment of a value to each variable such that all the constraints are satisfied and \( f(V) \) is minimal (maximal).

Optimization: Longest Paths

What is the longest (simple) path from A to B?

DFS: When backtrack, a node is marked as “processed”
CSP: When backtrack, a node is unmarked as “discovered”

Algorithms for Optimization

Key Components:
- Propagation algorithms: identify propagation algorithms for optimization function
- Branching algorithms: identify branching algorithms that lead to good solutions early
- Exploration algorithms: extend existing exploration algorithms to achieve optimization

Optimization: Example

\[ \text{SEND} + \text{MOST} = \text{MONEY} \]
SEND + MOST = MONEY

Assign distinct digits to the letters S, E, N, D, M, O, T, Y such that

\[
\begin{array}{c}
S E N D \\
+ M O S T \\
\hline
M O N E Y
\end{array}
\]

holds and
MONEY is maximal.

Branch and Bound

Identify a branching algorithm that finds good solutions early.

Example: TSP: Traveling Salesman Problem

Idea: Use the earlier found value as a bound for the rest branches.