Constraint Satisfaction

Reading: Russell & Norvig Chapter 5,
Kumar, “Algorithms for constraint satisfaction problems: A survey”

Overview

- Constraint Processing offers a powerful problem-solving paradigm
  - View a problem as a set of variables to which we have to assign values
  - that satisfy a number of problem-specific constraints
- Algorithms for CSPs
  - Backtracking (systematic search)
  - Constraint propagation (k-consistency)
  - Variable ordering heuristics
  - Backjumping and dependency-directed backtracking

Informal Definition of CSP

- CSP = Constraint Satisfaction Problem
- Given:
  1) a finite set of variables
  2) each with a domain of possible values (often finite)
  3) 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 the constraints are all 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.

SEND + MORE = MONEY

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

\[ \begin{align*}
  \text{SEND} &+ \text{MORE} \\
  &\quad = \text{MONEY}
\end{align*} \]

holds.

A Model for MONEY

- number of variables: 8
- constraints:
  - \( c_0 = (x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7) \in \mathbb{Z}^8 \mid \text{other constraints} \)
  - \( c_1 = (x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7) \in \mathbb{Z}^8 \)
  - \( c_2 = (1000x_0 + 100x_1 + 10x_2 + x_3 + 1000x_4 + 100x_5 + 10x_6 + x_7) \)
  - \( c_3 = (x_0 + x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7) \neq 0 \)
  - \( c_4 = (x_0 + x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7) \neq 0 \)
  - \( c_5 = (x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7) \) all different

Solution:

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

Modeling

Formalize the problem as a constraint problem:

- number of variables: \( n \)
- constraints: \( c_0 \rightarrow c_n \in \mathbb{Z}^n \)
- problem: Find \( x = (x_0, x_1, x_2, \ldots, x_n) \) such that \( x = c_i \) for all \( 1 \leq i \leq n \)

Solution for MONEY

\( c_0 = (x_0, x_1, x_2, x_3, x_4, x_5, x_6, x_7) \in \mathbb{Z}^8 \)

Solution:

\( (9, 5, 6, 7, 1, 0, 8, 2) \in \mathbb{Z}^8 \)
Informal Example: Map Coloring
• Color the following map using three colors (red, green, blue) such that no two adjacent regions have the same color.

Real-world problems
• Scheduling
• Temporal reasoning
• Building design
• Planning
• Optimization/satisfaction
• Vision

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

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 CN into an equivalent CN that is easier to solve.

Example: SATisfiability
• Given a set of propositions containing variables, find an assignment of the variables to {true, false} that satisfies them.
• Example, the clauses:
  – A or B or ~C
  – ~A or D
• Are satisfied by
  – A = true
  – C = false
  – D = false

Formal definition of a constraint network (CN)
A constraint network (CN) consists of
• a set of variables X = {x1, x2, … xn}
  – each with an associated domain of values {d1, d2, … dn}.
  – The domains are typically finite
• a set of constraints {c1, c2 … cm} where
  – each constraint defines a predicate which is a relation over a particular subset of X.
  – E.g., C1 involves variables {X1, X2, … Xk} and defines the relation R1 ⊆ D1 x D2 x … Dk
• Unary constraint: only involves one variable
• Binary constraint: only involves two variables

Binary CSP
• A binary CSP is a CSP in which all of the constraints are binary or unary.
• Any non-binary CSP can be converted into a binary CSP by introducing additional variables.
• A binary CSP can be represented as a constraint graph, which has a node for each variable and an arc between two nodes if and only there is a constraint involving the two variables.
  – Unary constraint appears as self-referential arc

Formal definition of a CN (cont.)
• Instantiations
  – An instantiation of a subset of variables S is an assignment of a legal domain value to each variable in in S.
  – An instantiation is legal iff it does not violate any (relevant) constraints.
• A solution is an instantiation of all of the variables in the network.

Solving Constraint Problems
• Systematic search
  – Generate and test
  – Backtracking
• Constraint propagation (consistency)
• Variable ordering heuristics
• Backjumping and dependency-directed backtracking
Generate and test

- Try each possible combination until you find one that works:
  - Hoses - hike - run - hike - no
  - Hoses - hike - run - hike - be
  - Hoses - hike - run - hike - us

- Doesn’t check constraints until all variables have been instantiated
- Very inefficient way to explore the space of possibilities

Running Example: XWORD Puzzle

- Variables and their domains:
  - X1 is 1 across D1 is 5-letter words
  - X2 is 2 down D2 is 4-letter words
  - X3 is 3 down D3 is 3-letter words
  - X4 is 4 across D4 is 4-letter words
  - X5 is 5 across D5 is 2-letter words

- Constraints (implicit/intensional):
  - C12 is “the 3rd letter of X1 must equal the 1st letter of X2”
  - C13 is “the 5th letter of X1 must equal the 1st letter of X3”

Systematic search: Backtracking (a.k.a. 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

Example: Crossword Puzzle

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Elements of Constraint Programming

- Use propagation algorithms for constraints
- Employ branching algorithm
- Execute exploration algorithm

Propagate
Finite Domain Constraint Satisfaction Problems

A finite domain constraint problem consists of:
• number of variables: \( n \)
• constraints: \( c_1, \ldots, c_m \)

The problem is to find \( a = (v_1, \ldots, v_n) \in \mathbb{Z}^n \) such that
\[ a = c_i, \quad \text{for all } 1 \leq i \leq n \]

Programming Systems for Finite Domain Constraint Programming

• Finite domain constraint programming library
  – Delphi (Papa 1993)
  – CLP (Dahl, Hoorens, Sintbrands, Agmon 1993)
  – CLP (Bachmair, Hentenryck, Simonis, Agmon 1988)
  – MFC (Ott, Mckinley, Carbon 1995)
  – OPL (see Hentenryck 1999)
**Constraint Solving**

Given a satisfiable constraint \( C \) and a new constraint \( C' \), constraint solving means deciding whether \( C \land C' \) is satisfiable.

**Example:**

\[
C: \quad n > 2
\]

\[
C': \quad an + b n = c n
\]

**Constraint Solving**

Constraint solving is not possible for general constraints. Constraint programming separates constraints into

- **Basic constraints:** complete constraint solving
- **Non-basic constraints:** propagation (incomplete); search needed

**Basic Constraints in Finite Domain**

Constraint Programming

- **Basic constraints** are conjunctions of constraints of the form \( X = S \), where \( S \) is a finite set of integers.
- **Constraint solving** is done by intersecting domains.

**Example:**

\[
C = (X \in \{1..10\} \land Y \in \{9..20\})
\]

\[
C' = (X \in \{9..15\} \land Y \in \{14..30\})
\]

In practice, we keep a solved form, storing the current domain of every variable.

---

**Basic Constraints and Propagators**

- **Basic Constraints** in Finite Domain

  - Basic constraints are conjunctions of constraints of the form \( X = S \), where \( S \) is a finite set of integers.
  - Constraint solving is done by intersecting domains.

  **Example:**

  \[
  C = (X \in \{1..10\} \land Y \in \{9..20\})
  \]

  \[
  C' = (X \in \{9..15\} \land Y \in \{14..30\})
  \]

  - In practice, we keep a solved form, storing the current domain of every variable.

**Standard Search Formulation**

Uses one of the straightforward approaches, e.g.

- **Initial state:** the empty assignment \( \{\} \)
- **Successor function:** assign a value to an unassigned variable that does not conflict with current assignment
- **Goal test:** the current assignment is complete

  - This is the same for all CSPs
  - Every solution appears at depth \( n \) with \( n \) variables
  - Use depth-first search

  - Path is irrelevant, so can also use complete-state formulation

  - \( b = (n - l) \cdot d \) at depth \( l \), hence \( n! \cdot d^n \) leaves
Backtracking search

- Variable assignments are commutative, i.e., \[ WA = \text{red} \text{ then } NT = \text{green} \] is the same as \[ NT = \text{green} \text{ then } WA = \text{red} \]
- Only need to consider assignments to a single variable at each node
- Depth-first search for CNF with single-variable assignments is called backtracking search
- Backtracking search is the basic uninformed algorithm for CNFs
- Can solve n-queens for \( n \approx 25 \)

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

Forward checking

- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values

Issues in Propagation

- Expressivity: What kind of information can be expressed as propagators?
- Completeness: What behavior can be expected from propagation?
- Efficiency: How much computational resources does propagation consume?

Forward checking

- Idea:
  - Keep track of remaining legal values for unassigned variables
  - Terminate search when any variable has no legal values

Completeness of Propagation

- Given: Basic constraint C and propagator P.
- Propagation is complete, if for every variable x and every value v in the domain of x, there is an assignment in which x=v that satisfies C and P.
- Complete propagation is also called domain-consistency or arc-consistency.

Constraint propagation

- Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:
  - NT and
  - Constraint propagation repeatedly updates constraints locally

Completeness of Propagation

- General arithmetic constraints are undecidable (Hilbert's Tenth Problem).
- Propagation may not exhibit all inconsistencies.
- Example:
  \[ a \cdot x > 2 \quad \text{and} \quad a \cdot x > 2 \]
Example: Complete All Different

\begin{itemize}
  \item C: \ w \in \{1, 2, 3, 4\}
  \item x \in \{2, 3, 4\}
  \item y \in \{2, 3, 4\}
  \item z \in \{2, 3, 4\}
  \item P: all\_different(w, x, y, z)
\end{itemize}

**ACC 1997/98: A Success Story of Constraint Programming**

- Integer programming + enumeration, 24 hours
- Constraint programming, less than 1 minute
- Henz: Scheduling a Major College Basketball Conference - Revised, Operations Research, to appear

**Basic Constraints vs. Propagators**

- Basic constraints
  - conjuncts of constraints of the form \(x \in S\), where \(x\) is a finite set of integers
  - enjoy complete constraint solving
- Propagators
  - can be arbitrarily expressive (arithmetic, symbolic)
  - implementation depends on fact but incomplete

**Round Robin Tournament Planning Problems**

- \(n\) teams, each playing a fixed number of times \(r\) against every other team
- \(r = 1\): single, \(r = 2\): double round robin
- Each match is home match for one and away match for another
- Dense round robin:
  - At each date, each team plays at most once.
  - The number of dates is minimal.

**The ACC 1997/98 Problem**

- 9 teams participate in tournament
- Dense double round robin:
  - there are 2 * 9 dates
  - at each date, each team plays either home, away or has a "bye"
- Alternating weekday and weekend matches

**The ACC 1997/98 Problem (cont’d)**

- 9 teams participate in tournament
- Dense double round robin:
  - there are 2 * 9 dates
  - at each date, each team plays either home, away or has a "bye"
- Alternating weekday and weekend matches

**ACC 1997/98 Problem (cont’d)**

- No team can play away on both last dates.
- No team may have more than three away matches in a row.
- No team may have more than three away matches or byes in a row.
- No team may have more than four home matches or byes in a row.

**The ACC 1997/98 Problem (cont’d)**

- No team can play away on both last dates.
- No team may have more than two away matches in a row.
- No team may have more than two home matches in a row.
- No team may have more than three away matches or byes in a row.
- No team may have more than four home matches or byes in a row.

**The ACC 1997/98 Problem**

- Of the weekends, each team plays four at home, four away, and one bye.
- Each team must have home matches or byes at least on two of the first five weekends.
- Every team except FSU has a traditional rival. The rival pairs are Clem-GT, Duke-UNC, UMD-UVA and NCSt-Wake. In the last date, every team except FSU plays against its rival, unless it plays against FSU or has a bye.

**The ACC 1997/98 Problem (cont’d)**

- Of the weekends, each team plays four at home, four away, and one bye.
- Each team must have home matches or byes at least on two of the first five weekends.
- Every team except FSU has a traditional rival. The rival pairs are Clem-GT, Duke-UNC, UMD-UVA and NCSt-Wake. In the last date, every team except FSU plays against its rival, unless it plays against FSU or has a bye.

**The ACC 1997/98 Problem**

- The following pairings must occur at least once in dates 11 to 18: Duke-GT, Duke-Wake, GT-UNC, UNC-Wake.
- No team plays in two consecutive dates away against Duke and UNC. No team plays in three consecutive dates against Duke UNC and Wake.
- UNC plays Duke in last date and date 11.
- UNC plays Clinton in the second date.
- Duke has bye in the first date.
The ACC 1997/98 Problem (cont’d)
• Wake does not play home in date 17.
• Wake has a bye in the first date.
• Clem, Duke, UMD and Wake do not play away in the last date.
• Clem, FSU, GT and Wake do not play away in the first date.
• Neither FSU nor NCSt have a bye in the last date.
• UNC does not have a bye in the first date.

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.

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.

Example: Map Coloring
• Variables: A, B, C, D. E all of domain RGB
• Domain: 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

Optimization: Example
• Modeling: define optimization function
• 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

Send + Most = Money
Assign distinct digits to the letters S, E, N, D, M, O, T, Y such that
SEND
+ MOST
---
MONEY
holds and
MONEY is maximal.

Example: Money
• Number of variables: 8
• Constraints: D_1, ..., D_8 \in \mathbb{Z}^8
• Optimization constraint: \sum_{i=1}^{8} x_i \leq 2^d
Given a solution a, an optimization constraint D_i, and the constraint \text{Di}(a) \subseteq \mathbb{Z}^n contains only those assignments \bar{a} for which \bar{a} is better than a.

A Model for Money
• number of variables: 8
• constraints:
  \begin{align*}
  &s_i = 1000x + 100y + 10z + w \leq 2^8 \quad \text{for} \; 0 \leq x, \ldots, w \leq 9 \\
  &s_i = 1000x + \ldots + w \leq 2^8 \quad \text{for} \; 0 \leq x, \ldots, w \leq 9 \\
  &s_i = 1000x + 100y + 10z + w \leq 2^8 \\
  &s_i = 1000x + 100y + 10z + w \leq 2^8
  \end{align*}
A Model for MONEY (continued)

• more constraints
  \( c_3 = \{(S,E,N,D,M,O,T,Y) \in \mathbb{Z}^8 | S \neq 0 \} \)
  \( c_4 = \{(S,E,N,D,M,O,T,Y) \in \mathbb{Z}^8 | M \neq 0 \} \)
  \( c_5 = \{(S,E,N,D,M,O,T,Y) \in \mathbb{Z}^8 | S \ldots Y \text{ all different} \} \)

• optimization constraint:
  \( d: (s,e,n,d,m,o,t,y) \rightarrow \{(S,E,N,D,M,O,T,Y) \in \mathbb{Z}^8 | 10000m + 1000o + 100n + 10e + y < 10000M + 1000O + 100N + 10E + Y \} \)

Propagation Algorithms

Identify a propagation algorithm to implement the optimization constraints

Example: SEND + MOST = MONEY

\( d: (s,e,n,d,m,o,t,y) \leftrightarrow \{(s,e,n,d,m,o,t,y) \in \mathbb{Z}^8 | 10000m + 1000o + 100n + 10e + y < 10000M + 1000O + 100N + 10E + Y \} \)

Given a solution \( a \), choose propagation algorithm for \( d(a) \).

Branch and Bound

Identify a branching algorithm that finds good solutions early.

Example: SEND + MOST = MONEY

Idea: Naive enumeration in the order M, O, N, E, Y.
Try highest values first.

Consistency

• Node consistency: A node \( X \) is node-consistent if every value in the domain of \( X \) is consistent with \( X \)'s unary constraints.
  A graph is node-consistent if all nodes are node-consistent.

• Arc consistency: An arc \((X,Y)\) is arc-consistent if, for every value of \( X \), there is a value of \( Y \) that satisfies the constraint represented by the arc.
  To create arc consistency, we perform constraint propagation: that is, we repeatedly reduce the domain of each variable to be consistent with its arcs.

Arc consistency

• Simplest form of propagation makes each arc consistent
  \( X \rightleftharpoons Y \) is consistent iff for every value of \( X \) there is some allowed \( y \)

• If \( X \) loses a value, neighbors of \( X \) need to be rechecked
  • Arc consistency detects failure earlier than forward checking
  • Can be run as a preprocessor or after each assignment

Arc consistency algorithm AC-3

\( X = \{x_1, x_2, \ldots, x_n\} \) a set of variables
\( \mathcal{C} = \{C_1, C_2, \ldots, C_m\} \) a set of constraints

Algorithm:

1. Initialize \( C = \mathcal{C} \)
2. While \( C \neq \emptyset \):
   a. Choose a constraint \( C_i \) from \( C \)
   b. Choose a variable \( X \) from the left side of \( C_i \)
   c. Choose a value \( x \) from the domain of \( X \)
   d. Remove \( x \) from the domain of \( x \)
   e. Remove \( C_i \) from \( C \)
   f. If \( C_i \) is empty, remove \( C_i \) from \( C \)
   g. If \( X \) has no other constraints, remove \( X \) from \( X \)

Time complexity: \( O(n^2d^3) \)
Variables: X1, X2, X3, X4, X5

Domains:
- D1 = {hoses, laser, sheet, snail, steer}
- D2 = {hike, aron, keet, earn, same}
- D3 = {run, sun, let, yes, eat, ten}
- D4 = {hike, aron, keet, earn, same}
- D5 = {no, be, us, it}

Constraints (explicit/extensional):
- C12 = {(hoses, same), (laser, same), (sheet, earn), (steer, earn)}
- C13 = ...
- C15 = ...

Why do we care?
1. If we have a CSP with N variables that is known to be strongly N-consistent, we can solve it without backtracking.
2. For any CSP that is strongly K-consistent, if we find an appropriate variable ordering (one with "small enough" branching factor), we can solve the CSP without backtracking.

Constraint propagation: XWORD example

Improving Backtracking
- Use other search techniques: uniform cost, A*, ...
- Variable ordering can help improve backtracking.
- Typical heuristics:
  - Prefer variables which maximally constrain the rest of the search space.
  - When picking a value, choose the least constraining value.

K-consistency
- K-consistency generalizes the notion of arc consistency to sets of more than two variables.
- A graph is K-consistent if, for legal values of any K-1 variables in the graph, and for any Kth variable V_k, there is a legal value for V_k.
- Strong K-consistency = J-consistency for all J<=K
- Node consistency = strong 1-consistency
- Arc consistency = strong 2-consistency
- Path consistency = strong 3-consistency

The Future
- Constraint programming will become a standard technique in OR for solving combinatorial problems, along with local search and integer programming.
- Constraint programming techniques will be tightly integrated with integer programming and local search.