Constraint Satisfaction 101

Berthe Y. Choueiry (Shu-we-ri)
Constraint Processing

✓ Constraint Satisfaction
  – Computational problem
    • Constraint Satisfaction Problem (CSP)
    • Constraint Optimization Problem (COP)
    • Extensions: dynamic/conditional constraints, preferences, etc.
  – Mainly solved with
    • Search algorithms: Backtrack search, local search
    • Propagation algorithms: filtering, consistency checking

✗ Constraint Programming
  – Mainly Constraint Logic Programming (CLP)
  – Highly declarative, provide
    • High-level constraint primitives (but also user-defined)
    • Solution engines that can be general or highly specialized (but user is not supposed to mess with them)
Outline

• Motivating example, application areas
• CSP: Definition, representation
• Some simple modeling examples
• More on definition and formal characterization
• Basic solving techniques
• Advanced solving techniques
• Issues & research directions
• CSPs in a nutshell
Motivating example

- **Context**: You are in the MS program at USC

- **Problem**: You need to register in 3 courses for the Spring semester

- **Possibilities**: Many courses offered in Math, CS, EE, BA, etc.

- **Constraints**: restrict the choices you can make
  - **Unary**: Courses have prerequisites you have/don't have
    Courses/instructors you like/dislike
  - **Binary**: Courses are scheduled at the same time
  - **n-ary**: 4 courses from 5 tracks such as at least 3 tracks are covered

- **You have choices, but are restricted by constraints**
  - Make the right decisions 😊
Motivating example (cont’d)

• **Given**
  – A set of variables: 3 courses at USC
  – For each variable, a set of choices (values)
  – A set of constraints that restrict the combinations of values the variables can take at the same time

• **Questions**
  – Does a solution exist? (classical decision problem)
  – How two or more solutions differ? How to change specific choices without perturbing the solution?
  – If there is no solution, what are the sources of conflicts? Which constraints should be retracted?
  – etc.
Practical applications

- Radio resource management (RRM)
- Databases (computing joins, view updates)
- Temporal and spatial reasoning
- Planning, scheduling, resource allocation
- Design and configuration
- Graphics, visualization, interfaces
- Hardware verification and software engineering
- HC Interaction and decision support
- Molecular biology
- Robotics, machine vision and computational linguistics
- Transportation
- Qualitative and diagnostic reasoning
- ... But also, Sudoku & Minesweeper
Constraint Processing

• **is about ...**
  – Solving a decision problem…
  – … While allowing the user to state arbitrary constraints in an expressive way and
  – Providing concise and high-level feedback about alternatives and conflicts

• **Related areas:**
  – AI, OR, Algorithmic, DB, TCS, Prog. Languages, etc.
Power of Constraints Processing

- Flexibility & expressiveness of representations
- Interactivity

users can \( \{ \text{relax, reinforce} \} \) constraints
Outline

• Motivating example, application areas
• **CSP: Definition, representation**
• Some simple modeling examples
• More on definition and formal characterization
• Basic solving techniques
• Advanced solving techniques
• Issues & research directions
• CSPs in a nutshell
Definition of a CSP

- **Given** $P = (V, D, C)$
  - $V$ is a set of variables, $V = \{V_1, V_2, \ldots, V_n\}$
  - $D$ is a set of variable domains (domain values)
    $$D = \{D_{V_1}, D_{V_2}, \ldots, D_{V_n}\}$$
  - $C$ is a set of constraints, $C = \{C_1, C_2, \ldots, C_l\}$
    with $C_{V_a, V_b, \ldots, V_i} = \{(x, y, \ldots, z)\} \subseteq D_{V_a} \times D_{V_b} \times \ldots \times D_{V_i}$

- **Query**: can we find a value for each variable such that all constraints are satisfied?
Other queries…

• Find a solution
• Find all solutions
• Find the number of solutions
• Find a set of constraints that can be removed so that a solution exists
• Etc.
Representation I

• Given \( P = (V, D, C) \), where

\[
V = \{V_1, V_2, \ldots, V_n\}
\]
\[
D = \{D_{V_1}, D_{V_2}, \ldots, D_{V_n}\}
\]
\[
C = \{C_1, C_2, \ldots, C_l\}
\]

• Find a consistent assignment for variables

Constraint Graph

• Variable \( \rightarrow \) node (vertex)

• Domain \( \rightarrow \) node label

• Constraint \( \rightarrow \) arc (edge) between nodes
Representation II

Binary constraints

- $\{1,2,3,4\} < \{3,6,7\}$
- $\{3,4,9\} < \{3,5,7\}$
- $v_1 + v_3 < 9$
- $v_2 > v_4$

Non-Binary constraints

- $\{1,2,3,4\} < \{3,6,7\}$
- $v_1 + v_3 < 9$
- $v_2 < v_3$
- $v_2 > v_4$
- $v_1 + v_2 + v_4 < 10$

Constraints: unary, binary, ternary, … , global.
Outline

• Motivating example, application areas
• CSP: Definition, representation
• Some simple modeling examples
• More on definition and formal characterization
• Basic solving techniques
• Advanced solving techniques
• Issues & research directions
• CSPs in a nutshell
Example: Temporal reasoning

- Give one solution: ........
- Satisfaction, yes/no: decision problem
Example: Map coloring

Using 3 colors (R, G, & B), color the US map such that no two adjacent states have the same color

- Variables?
- Domains?
- Constraints?
Example: Map coloring (cont’d)

Using 3 colors (R, G, & B), color the US map such that no two adjacent states have the same color.
Example: Resource Allocation

What is the CSP formulation?
Example: RA (cont’d)
Example: Sudoku

Given:

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>1</td>
<td></td>
<td></td>
<td>2</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>7</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td></td>
<td>7</td>
<td>8</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>9</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>3</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Query: Fill the empty cell such that 1..9 appear in each row, column, and unit w/o repetition

• One model
  – 81 variables: C₁,₁... C₉,₉
  – Domains: {1,2,3,...,8,9}
  – Constraints:
    • all-diff constraints, 9-arity
    • One constraint per row
    • One constraint per column
    • One constraint per (3x3) unit
Example: Cryptarithmetic puzzles

- Domains
  - $D_{x_1} = D_{x_2} = D_{x_3} = \{0,1\} \quad (a)$
  - $D_F=D_T=D_U=D_W=D_R=D_O=[0,9]$

- Constraints
  - $O+O = R+10X_1$
  - $X_1+W+W = U+10X_2$
  - $X_2+T+T = O+10X_3$
  - $X_3=F$
  - $\text{Alldiff}\{F,D,U,V,R,O\}$
More examples

• Product configuration: car, Xerox copier, etc.
• Join operation in relational DB is a CSP
• Interactive systems (Data-flow constraints)
  – Spreadsheets
  – Graphical layout systems and animation
  – Graphical user interfaces
• Molecular biology (bioinformatics)
  – Threading, etc
Outline

• Motivating example, application areas
• CSP: Definition, representation
• Some simple modeling examples
• More on definition and formal characterization
• Basic solving techniques
• Advanced solving techniques
• Issues & research directions
• CSPs in a nutshell
Domain types

- $\mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$ where
  \[
  \mathcal{V} = \{V_1, V_2, \ldots, V_n\}
  \]
  \[
  \mathcal{D} = \{D_{V_1}, D_{V_2}, \ldots, D_{V_n}\}
  \]
  \[
  \mathcal{C} = \{C_1, C_2, \ldots, C_l\}
  \]

  **with** $C_{V_k, V_l, \ldots, V_m} = \{(x, y, \ldots, z)\} \subseteq D_{V_k} \times D_{V_l} \times \ldots \times D_{V_m}$

- Domains:
  - Restricted to $\{0,1\}$: **Boolean** CSPs
  - Finite (discrete), enumeration works
  - Continuous, sophisticated algebraic techniques are needed
    - Consistency techniques on domain bounds
Constraint terminology

- **Arity:**
  - universal, unary, binary, ternary, ..., global

- **Scope:**
  - The set of variables to which the constraint applies

- **Definition:**
  - Extension: all allowed tuples are listed
  - Constrained Decision Diagrams [Cheng & Yap, AAAI 05]
  - Intention: when it is not practical (or possible) to list all tuples
    - Define types/templates of common constraints to be used repeatedly: linear constraints, All-Diff (mutex), Atmost, TSP-constraint, cycle-constraint, etc.

- **Implementation:**
  - Predicate: $\text{CHECK}(V_i \leftarrow a, V_j \leftarrow b)$
  - List set of tuples (table)
  - Bit-matrix, etc.
CSP: complexity

- Decision problem
  - Constraints are verifiable in polynomial time
  - NP-complete in general by reduction from 3SAT
  - Special cases may be tractable
- Optimization problem
  - CSP + objective function
  - In general, NP-hard
Outline

• Motivating example, application areas
• CSP: Definition, representation
• Some simple modeling examples
• More on definition and formal characterization
• **Basic solving techniques**
  • Modeling
  • Constraint Propagation
  • Constructive, systematic search
  • Iterative improvement, local search
• Advanced solving techniques
• Issues & research directions
• CSPs in a nutshell
Solving a CSP

• Search
  – Constructive, exhaustive search
  – Iterative repair, local search

• Before starting search
  – Consider the importance of modeling/formulation, which determines the size of the search space
  – Consider constraint propagation/filtering to reduce the size of the search space
Importance of modeling

- **N-queen**: formulation 1
  - Variables? 4 Rows
  - Domains? 4 Column positions
  - Size of CSP? $4 \times 4 \times 4 \times 4 = 4^4 = 2^8$

- **N-queens**: formulation 2
  - Variables? 16 Cells
  - Domains? \{0,1\}
  - Size of CSP? $2^{16}$
Constraint checking: motivating example

Consider \( C_{B,C} : \text{REVISE}(D_B, C_{B,C}) \) and \( \text{REVISE}(D_C, C_{B,C}) \)
**Procedure REVISE** (binary constraints)

**Procedure REVISE**(i, j):

Revises the domains of a variable i

Begin

DELETE ← false

For each x ∈ Di do

Begin

FOUND ← false

Repeat until FOUND

For each y ∈ Dj do

FOUND ← CHECK(i←x, j←y)

If not FOUND

Begin

delete x from Di

DELETE ← true

End

End

Return DELETE

End

• REVISE is directional, O(a^2)
Arc-Consistency

→ Arc-consistency: every combination of two adjacent variables
  – Repeat **REVISE** until quiescence (guaranteed on finite domains)
    \[ D_{Vi} \leftarrow D_{Vi} \cap \pi_{Vi} (C_{Vi,Vj} \bigotimes D_{Vj}) \]
    – Quadratic complexity
→ Eliminate non-acceptable tuples prior to search
→ Results in a CSP with smaller domains, yielding a (potentially) smaller search space
→ **Warning**: arc-consistency does not solve the problem

\[ \{1, 2, 3\} \rightarrow \{2, 3, 4\} \rightarrow \{2, 3\} \]

\[ (A=2) \land (B=3) \text{ still is not a solution!} \]
Consistency properties

• Arc-consistency (2-consistency)
  Every value in the domain of every variable is consistent with at least one value of every other variable

• Path-consistency (3-consistency)
  Every consistent tuple of values for every 2 variables is consistent with at least one value in the domain of every 3rd variable

• $k$-consistency
  Every consistent tuple of values for every $(k-1)$ variables is consistent with at least one value in the domain of every $k$th variable

• Other properties:
  strong $k$-consistency, $(i,j)$-consistency, singleton arc-consistency, neighborhood inverse consistency, minimality, decomposability, consistency, etc.

• Alert: terminology often confuses properties & algorithms
Systematic, exhaustive search

→ Starting from a root node
→ Consider all values for a variable $V_1$
→ For every value for $V_1$, consider all values for $V_2$
→ etc..

→ For $n$ variables, each of domain size $d$
→ Maximum depth? $fixed!$
→ Maximum number of paths? $size of search space, size of CSP$
Back-checking for early pruning

- Systematic search generates $d^n$ possibilities
- Are all possibilities acceptable?

→ Expand a partial solution only when it is consistent
→ This yields early pruning of inconsistent paths
Backtrack(ing) search (BT)

What if only one solution is needed?

- Depth-first search & Chronological backtracking

- **DFS**: Soundness? Completeness?
General techniques for ...

... improving the performance of backtrack search, which is an exponential-time process

1. Ordering heuristics: variable, value ordering
2. Intelligent backtracking avoids repeating failure
3. Look-ahead techniques: propagate constraints as variables are instantiated
Ordering heuristics

- Which variable to expand first?

**Exp**: \( V_1, V_2, D_{V_1} = \{a, b, c, d\}, D_{V_2} = \{a, b\} \)

**Sol**: \( \{(V_1 = c), (V_2 = a)\} \) and \( \{(V_1 = c), (V_2 = b)\} \)

- Heuristics:
  - most constrained variable first (reduce branching factor)
  - most promising value first (find quickly first solution)
Examples of ordering heuristics

• Most constrained variable first
  – Least domain (LD), aka minimum remaining values (MRV)
  – Smallest degree
  – Ratio of domain size to degree (DD)
  – Graph width, promise, etc.

• Most promising value first
  – Min-conflict [Minton+ ‘92]
  – Promise [Geelen ’02], etc.

Strategies for \{ variable ordering \} \{ value ordering \} could be \{ static \} \{ dynamic \}
Intelligent Backtracking

What if the reason for failure was higher up in the tree?

Backtrack to source of conflict !!
Look-ahead strategies

- Principle: Interleave constraint propagation with search
- Terminology
  - $V_c$ the current variable, $V_f$ a future variable
- Mechanism
  - Instantiate $V_c$
  - Update the domain of some/all future variables
- Advantages
  1. Removes the need for back-checking
  2. Reduces the size of the future ‘subproblem’
  3. Domain anihilation causes rapid backtracking and reduces thrashing
- Strategies
  - Forward Checking (FC): partial look-ahead
    - Instantiate a variable, $\text{REVISE}(V_f, V_c)$ for each $V_f$ connected to $V_c$
  - Directed Arc-Consistency (DAC): partial look-ahead
  - Maintaining Arc-Consistency (MAC): full look-ahead
- Alert
  - Special data structures are needed to refresh the filtered domains upon backtracking
Back-checking during search

- Search tree with only backtrack search?

Root node

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

Solution!
26 nodes visited.
Forward Checking: search + REVISE

Search Tree with domains filter by Forward Checking

Root node

Solution!

8 nodes visited.
Backtrack search: summary

- Properties:
  - Constructive, systematic, exhaustive
  - In general sound and complete
  - Improvements: ordering heuristics, intelligent backtracking, look-ahead

- Problems
  - Worst-case time complexity prohibitive
  - Often unable to solve large problems. Theoretical soundness & completeness do not mean much in practice

- → Local search
Non-systematic search

• **Iterative repair, local search**: modifies a global but inconsistent solution to decrease the number of violated constraints

• **Examples**: Hill climbing, tabu search, simulated annealing, GSAT, WalkSAT, Genetic Algorithms, Swarm intelligence, etc.

• **Features**: Incomplete & not sound
  – Advantage: anytime algorithm
  – Shortcoming: cannot tell that no solution exists
Components of local search

- **State**
  - is a complete assignment of values to variables, a possibly inconsistent solution

- **Possible moves**
  - are modifications to the current state, typically by changing the value of a single variable. Thus, ‘local’ repair
  - Examples:
    - SAT: Flipping the value of a Boolean variable (GSAT),
    - CSPs: Min-conflict heuristic (variations)

- **Evaluation (cost) function**
  - rates the quality of the possible moves, typically in the number of violated constraints
Generic mechanism

- **Cost function**: number of broken constraints

- **General principle**
  - Start with a **full** but **arbitrary** assignment of values to variables
  - Reduce inconsistencies by **local repairs** (heuristic)
  - Repeat until
    - A solution is found (global minimum)
    - The solution cannot be repaired (local minimum)
    - … You run out of patience (max-tries)

- **A.k.a.**
  - Iterative repair (decision problems)
  - Iterative improvement (optimization problems)
Dealing with local optima

• Random walk
  – Next state is chosen randomly

• Breakout strategy
  – increase the weight of a constraint at each iteration if it continues to be broken

• Random restarts
  – Restart search often from a new random initial state

• Local beam search, etc.
Outline

• Motivating example, application areas
• CSP: Definition, representation
• Some simple modeling examples
• More on definition and formal characterization
• Basic solving techniques
• Advanced solving techniques
  – Decomposition
  – Deep analysis
  – Distributed CSPs
• Issues & research directions
• CSPs in a nutshell
Decomposition

- Decomposition
  - Conjunctive
  - Disjunctive

- Conjunctive

- Disjunctive
Properties of decompositions

\[ P \rightarrow \{P_1, P_2, \ldots, P_i\} \]

- **Consistent:** No constraint is removed
- **Simplifying:** \( \text{Size}(P_i) < \text{Size}(P) \)
- **Semi-complete:** At least one solution is kept
- **Complete:** No solution is lost
- **Redundant:** Solutions replicated in \( \{P_i\} \)
- **Reducible:** may be \( < \text{Size}(P) \)
Deep analysis

Uncover particular properties, e.g.
- Islands of tractability
- Predict ease/difficulty of solving a given instance

1. Efficient solvers/propagation algorithms that exploit a formal property of the CSP
   - Structure, topology of the constraint graph
     - Tree, chordal graphs, k-trees, bounded-width/induced width, etc.
   - Types, semantics of the constraints
     - All-diffs, linear inequalities, subsets of Allen's relations, functional, monotonic, row-convex, etc.

2. Prediction of problem difficulty based on a ‘macro’ parameter:
   - Order parameter and phase transition
Phase transition

[Cheeseman et al. ‘91]

- Significant increase of cost around critical value
- In CSPs, order parameter is constraint tightness
- Algorithms compared around phase transition
Distributed CSPs

• Mainly in search
  – Asynchronous BT (e.g., work of Yokoo)
  – Fine grain local search (ERA, by Liu)
  – Privacy of constraints

• More purist multi-agent approaches
  – In scheduling & resource allocation
  – Based on decomposition of problem/solvers
Outline

• Motivating example, application areas
• CSP: Definition, representation
• Some simple modeling examples
• More on definition and formal characterization
• Basic solving techniques
• Advanced solving techniques
• Issues & research directions
• CSPs in a nutshell
Research directions

- **Preceding** (i.e., search, backtrack, iterative repair, V/V/ordering, consistency checking, decomposition, symmetries & interchangeability, deep analysis)
- **Evaluation of algorithms**
  - worst-case analysis vs. empirical studies
  - random problems vs. real-world problems
- **Cross-fertilization**
  - DB, SAT & theoretical computer science (TCS), mathematical programming, interval mathematics, logical inference, applications, etc.
- **Modeling & Reformulation**
- **Extensions**
  - Non-binary, conditional, soft constraints & preferences, etc
- **Multi agents**
  - Distribution & decomposition
  - Negotiation & alliance formation
Outline

• Motivating example, application areas
• CSP: Definition, representation
• Some simple modeling examples
• More on definition and formal characterization
• Basic solving techniques
• Advanced solving techniques
• Issues & research directions
• CSPs in a nutshell
CSP in a nutshell (I)

Definition: \( P = (V, D, C) \)

\[
V = \{V_1, V_2, \ldots, V_n\} \\
D = \{D_{V_1}, D_{V_2}, \ldots, D_{V_n}\} \\
C = \{C_1, C_2, \ldots, C_l\}
\]

- Constraint graph, constraint network
- Finite/continuous domains
- Binary, non-binary constraints

Examples: map coloring, puzzles, resource allocation, temporal reasoning, product configuration, databases, spreadsheets, graphical layouts, graphical user-interfaces, bioinformatics, etc.
CSP in a nutshell (II)

Solution technique: Search

{ constructive
iterative repair

Enhancing search:

{ Consistency checking
Variable/value ordering
Intelligent backtracking
Look-ahead strategies
♥ Symmetries
♥ Decomposition

Nov 14, 2006
Constraint Satisfaction 101
CSP in a nutshell (III)

Deep analysis:
exploit problem structure

Research

- $k$-ary constraints, soft constraints
- continuous vs. finite domains
- evaluation of algorithms (empirical)
- cross-fertilization (mathematical program.)
- preferences and soft constraints
  ♥ reformulation and approximation
  ♥ architectures (multi-agent, negotiation)

♥ Graph topology
♥ Constraint semantics
Phase transition