Title: Constraint Satisfaction Problems

Required reading: AIMA: Chapter 5

Recommended reading:
- Introduction to CSPs (Bartak’s on-line guide)

Introduction to Artificial Intelligence
CSCE 476-876, Spring 2005

URL: www.cse.unl.edu/~choueiry/S05-476-876

Berthe Y. Choueiry (Shu-we-ri)
choueiry@cse.unl.edu, (402) 472-5444

---

Constraint Processing

- Constraint Satisfaction:
  - Modeling and problem definition (Constraint Satisfaction Problem, CSP)
  - Algorithms for constraint propagation
  - Algorithms for search

- Constraint Programming: Languages and tools
  - logic-based
  - object-oriented
  - functional
Courses on Constraint Processing

- Foundations of Constraint Processing, CSCE 421/821
  Fall’04 URL: cse.unl.edu/~choueiry/F04-421-821/
  Fall’03 URL: cse.unl.edu/~choueiry/F03-421-821/
  Fall’02 URL: cse.unl.edu/~choueiry/F02-421-821/
  Fall’01 URL: cse.unl.edu/~choueiry/F01-421-821/
  Fall’00 URL: cse.unl.edu/~choueiry/F00-CSCE990/
  Fall’99 URL: cse.unl.edu/~choueiry/CSE990-05/

- Advanced Constraint Processing, CSCE 990-06
  Spring’03 URL: cse.unl.edu/~choueiry/S03-990-06/

Outline

- Problem definition and examples
- Solution techniques: search and constraint propagation
- Deeper analysis
- Research directions
What is this about?

**Context:** You are a senior in college

**Problem:** You need to register in 4 courses for Spring’2000

**Possibilities:** Many courses offered in Math, CSE, EE, 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:* In CompEng, 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

---

**Constraint Satisfaction**

**Given**

- A set of variables
- 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.*
**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

**Power of Constraints Processing**

- flexibility & expressiveness of representations
- interactivity, users can \[ \left\{ \begin{array}{c}
\text{relax} \\
\text{reinforce}
\end{array} \right\} \text{ constraints} \]

**Related areas:** AI, OR, Algorithmic, DB, Prog. Languages, *etc.*

---

**Definition**

**Given** \( \mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{C}) \):

- \( \mathcal{V} \) a set of variables
  \[ \mathcal{V} = \{V_1, V_2, \ldots, V_n\} \]
- \( \mathcal{D} \) a set of variable domains (domain values)
  \[ \mathcal{D} = \{D_{V_1}, D_{V_2}, \ldots, D_{V_n}\} \]
- \( \mathcal{C} \) a set of constraints
  \[ C_{V_a, V_b, \ldots, V_i} = \{ (x, y, \ldots, z) \} \subseteq D_{V_a} \times D_{V_b} \times \ldots \times D_{V_i} \]

**Query:** can we find one value for each variable such that all constraints are satisfied?

In general, **NP-complete**
Terminology

- Instantiating a variable: \( V_i \leftarrow a \) where \( a \in D_{V_i} \)
- Partial assignment
- Consistent assignment
- Objective function (constrained optimization problem)

Representation

Given \( \mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{C}) \) \quad \begin{align*}
\mathcal{V} &= \{V_1, V_2, \ldots, V_n\} \\
\mathcal{D} &= \{D_{V_1}, D_{V_2}, \ldots, D_{V_n}\} \\
\mathcal{C} &= \text{set of constraints}
\end{align*}

\( C_{V_i, V_j} = \{(x, y)\} \subseteq D_{V_i} \times D_{V_j} \)

Constraint graph

\[ \begin{array}{c}
\text{V1} & \{1, 2, 3, 4\} & \text{V2} & \{3, 6, 7\} \\
\text{V3} & \{3, 4, 9\} & \text{V4} & \{3, 5, 7\}
\end{array} \]
Example I: Temporal reasoning

$$A < B \quad [1 \ldots 10]$$

$$2 < C - A \leq 5$$

$$\Rightarrow \quad c-A \in \{2, 5\}$$

Example II: Map coloring

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

Variables? Domains? Constraints?
**Incremental formulation:** as a search problem

**Initial state:** empty assignment, all variables are unassigned

**Successor function:** a value is assigned to any unassigned variable, provided that it does not conflict with previously assigned variables (back-checking)

**Goal test:** The current assignment is complete (and consistent)

**Path cost:** a constant cost (e.g., 1) for every step, can be zero

— A solution is a complete, consistent assignment.
— Search tree has constant depth \( n \) (# of variables) \( \rightarrow \) DFS!!
— Path for reaching a solution is irrelevant
— A complete-state formulation is OK
— local-search techniques applicable

**Domain types**

Given \( \mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{C}) \) \[
\begin{aligned}
\mathcal{V} &= \{V_1, V_2, \ldots, V_n\} \\
\mathcal{D} &= \{D_{V_1}, D_{V_2}, \ldots, D_{V_n}\} \\
\mathcal{C} &= \text{set of constraints}
\end{aligned}
\]

\[ C_{V_i,V_j} = \{ (x, y) \} \subseteq D_{V_i} \times D_{V_j} \]

**Domains:**

— restricted to \( \{0, 1\} \): Boolean CSPs
— Finite (discrete): enumeration techniques works
— Continuous: sophisticated algebraic techniques are needed
  consistency techniques on domain bounds
Constraint arity

Given $\mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{C})$

\[ \begin{align*}
\mathcal{V} &= \{V_1, V_2, \ldots, V_n\} \\
\mathcal{D} &= \{D_{V_1}, D_{V_2}, \ldots, D_{V_n}\} \\
\mathcal{C} &= \text{set of constraints} \\
C_{V_k, V_l, V_m} &= \{ (x, y, z) \} \subseteq D_{V_k} \times D_{V_l} \times D_{V_m}
\end{align*} \]

Constraints: universal, unary, binary, ternary, ..., global

Representation: Constraint network

Constraint language

Constraints can be defined

- Extensionally: all allowed tuples are listed
  practical for defining arbitrary constraints

- Intensionally: when it is not practical (or even possible) to list
  all tuples
  → define types of common constraints, which can be used
  repeatedly

Examples of types of constraints: linear constraints, nonlinear
constraints, Alldiff (a.k.a. mutex), Atmost, constraints of bounded
differences (e.g., in temporal reasoning), etc.
**Example III: Cryptarithmetic puzzles**

\[ D_{X_1} = D_{X_2} = D_{X_3} = \{0, 1\} \]
\[ D_F = D_T = D_U = D_V = D_R = D_O = [0, 9] \]

\[
\begin{array}{c}
T \ W \ O \\
+ T \ W \ O \\
\hline
F \ O \ U \ R
\end{array}
\]

O + O = R + 10 X1
X1 + W + W = U + 10 X2
X2 + T + T = O + 10 X3
X3 = F

\[
\text{Alldiff}\{F, \ D, \ U, \ V, \ R, \ O\}\]

---

**How to solve a CSP?**

Search!

1. Constructive, systematic search
2. Iterative repair search
**Systematic 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..

![Diagram of systematic search]

For $n$ variables, each of domain size $d$:
- Maximum depth? fixed!
- Maximum number of paths? $size \text{ of search space, size of CSP}$

---

**Before looking at search..**

Consider

1. Importance of modeling/formulating to control the size of the search space
2. Preprocessing: consistency filtering to reduce size of search space
Constraint checking

→ arc-consistency

→ AC-3: $O(n^2d^3)$
→ 3-consistency, $k$-consistency ($k \leq n$)
→ strong $k$-consistency ($\forall i \leq k$)
→ Also, node consistency
→ Constraint filtering, constraint checking, etc..

eliminate non-acceptable tuples prior to search
**Warning:** arc-consistency does not solve the problem

\[
\begin{array}{c}
A \{1, 2, 3\} & \rightarrow & B \{2, 3, 4\} \\
A \{2, 3\} & \rightarrow & B \{2, 3\}
\end{array}
\]

\((A = 2) \land (B = 3)\) still isn’t a solution!

- In general, constraint propagation helps, but does not solve the problem
- As long as constraint checking is affordable (i.e., cost remains negligible vis-a-vis cost of search), it is advantageous to apply before

---

**Back-checking**

Systematic search generates \(d^n\) possibilities

Are all possible combinations acceptable?

→ Expand a partial solution only when consistent

→ **early pruning**
Chronological backtracking

What if only one solution is needed?

→ Depth-first search & chronological backtracking

Terms: current variable $V_c$, past variables $V_p$, future variables $V_f$, current path

→ DFS: soundness? completeness?

Example of BT
**Backtrack(ing) search (BT)**

Refer to algorithm BACKTRACKING-SEARCH

- Implementation: BACKTRACKING-SEARCH
  Careful, recursive, do not implement!!
  Use [Prosser 93] for iterative versions
- Variable ordering heuristic: SELECT-UNASSIGNED-VARIABLE
- Value ordering heuristic: ORDER-DOMAIN-VALUES

**Improving BT**

General purpose methods for:

1. Variable, value ordering
2. Improving backtracking: intelligent backtracking avoids repeating failure
3. Look-ahead techniques: constraint propagation as instantiations are made
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: \[
\begin{align*}
\text{most constrained variable first (reduce branching factor)} \\
\text{most promising value first (find quickly first solution)}
\end{align*}
\]

Examples of ordering heuristics

For variables:
- least domain (LD), aka minimum remaining values (MRV)
- degree
- ratio of domain size to degree (DD)
- width, promise, etc. [Tsang, Chapter 6]

For values:
- min-conflict [Minton, 92]
- promise [Geelen, 94], etc.

Strategies for \[
\begin{align*}
\text{variable ordering} \\
\text{value ordering}
\end{align*}
\] could be \[
\begin{align*}
\text{static} \\
\text{dynamic}
\end{align*}
\]
**Intelligent backtracking**

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

**Backtrack to source of conflict!!**

→ Backjumping, conflict-directed backjumping, etc.
→ Additional data structures that keep track of failure encountered during back-checking [Prosser, 93]

---

**Look-ahead strategies**: partial or full

As instantiations are made, remove the values from the domain of future variables that are not consistent with the current path

→ After instantiating $V_c$, update the domains of future variables

- Forward checking (FC): Apply
  \[\text{Remove-Inconsistent-Values}(V_f, V_c)\] to the future variables connected to $V_c$

- Directed arc-consistency checking (DAC): Repeat forward checking for all variables in $V_f$, while respecting order
  Applicable under static ordering

- Maintaining Arc-Consistency (MAC): Apply AC-3($V_f$)
  In practice, useful when problem has few, tight constraints

→ Special data structures can be used to refresh filtered domains upon backtracking [Prosser, 93]
CSP: a decision problem (NP-complete)

1. Modeling:
   - abstraction and reformulation

2. Preprocessing techniques:
   - eliminate non-acceptable tuples prior to search

3. Search:
   - potentially $d^n$ paths of fixed length
   - chronological backtracking
   - variable/value ordering heuristics

4. Search hybrids:
   - Mixing preprocessing with search: FC, DAC, MAC
Non-systematic search

- **Methodology:** Iterative repair, local search: modifies a global but **inconsistent** solution to decrease the number of violated constraints
- **Example:** MIN-CONFLICTS algorithm in Fig 5.8, page 151. Choose (randomly) a variable in a broken constraint, and change its value using the min-conflict heuristic (which is a value ordering heuristic)
- **Other examples:** Hill climbing, taboo search, simulated annealing, etc.

→ Anytime algorithm
→ Strategies to avoid getting trapped: RandomWalk
→ Strategies to recover: Break-Out, Random restart, etc.
→ Incomplete & not sound

Deep analysis: Uncover particular properties, *e.g.*
- bounds the required level of consistency
- predicts ease/difficulty of solving a given instance

- **C-graph topology:** tree (see Section 5.4), chordal, etc.
- **Constraints type:** Alldiffs, Atmost, functional, monotonic, row-convex, subsets of Allen’s relations, etc.

- **Order parameter** (phase transition)
Exploiting structure: example of deep analysis

Tree-structured CSP

- Cycle-cutset method
- Tree-decomposition method (not discussed, Yaling’s research)

Tree-structured CSP

*Any tree-structured CSP can be solved in time linear in the number of variables.*

- Apply arc-consistency (directional arc-consistency is enough)
- Proceed instantiating the variables from the root to the leaves
- The assignment can be done in a backtrack-free manner
- Runs in $O(nd^2)$, $n$ is #variables and $d$ domain size.
Cycle-cutset method

- Identify a cutset $S$ in the CSP (nodes that when removed yield a tree) and remove the corresponding variables.
- Find a solution to the variables $T$ in the tree.
- Apply DAC from $T$ to $S$.
- Try to solve $S$ (smaller than initial problem).

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?

Cross-fertilization:
- SAT, DB, mathematical programming,
- interval mathematics, planning, etc.

Modeling & Reformulation

Multi agents:
- Distribution and negotiation
  $\rightarrow$ decomposition & alliance formation
CSP in a nutshell (I)

Solution technique: Search
- constructive
- iterative repair

Enhancing search:
- intelligent backtrack
- variable/value ordering
- consistency checking
- hybrid search
- ★ symmetries
- ★ decomposition

CSP in a nutshell (II)

Deep analysis: exploit problem structure
- ★ graph topology
- ★ constraint semantics
- phase transition

Research:
- k-ary constraints, soft constraints
- continuous vs. finite domains
- evaluation of algorithms (empirical)
- cross-fertilization (mathematical program.)
- ★ reformulation and approximation
- ★ architectures (multi-agent, negotiation)
Constraint Logic Programming (CLP)

A merger of

✓ Constraint solving

→ Logic Programming, mostly Horn clauses (e.g., Prolog)

Building blocks

- Constraint: primitives
  - cumulative/capacity (linear ineq), MUTEX, cycle, etc.
  - domain: Booleans, natural/rational/real numbers, finite

- Rules (declarative): a statement is a conjunction of constraints and is tested for satisfiability before execution proceeds further

- Mechanisms: satisfiability, entailment, delaying constraints

Your future: jobs

Commercial companies: i2 Technologies, Ilog, PeopleSoft/Red Pepper, Honeywell, Xerox Corp, etc.

Prestigious research centers: NASA Ames, JPL, BT Labs (UK), IBM Watson+Almaden, etc.

Start your own: i2 technologies started as a small group of researchers doing constraint-based scheduling

Academic: constraint languages, modeling, representation, automated reasoning, etc.
Constraint Processing Techniques are the basis of new languages:

Were you to ask me which programming paradigm is likely to gain most in commercial significance over the next 5 years I’d have to pick Constraint Logic Programming (CLP), even though it’s perhaps currently one of the least known and understood. That’s because CLP has the power to tackle those difficult combinatorial problems encountered for instance in job scheduling, timetabling, and routing which stretch conventional programming techniques beyond their breaking point.

Though CLP is still the subject of intensive research, it’s already being used by large corporations such as manufacturers Michelin and Dassault, the French railway authority SNCF, airlines Swissair, SAS and Cathay Pacific, and Hong Kong International Terminals, the world’s largest privately-owned container terminal.

*Byte, Dick Pountain*