Constraint Satisfaction


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.
  - Constraint programming, CSPs, constraint logic programming…
- 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.
assign distinct digits to the letters S, E, N, D, M, O, R, Y such that $\text{SEND} + \text{MORE} = \text{MONEY}$ holds.

Solution

$\begin{align*}
\text{SEND} & \quad 9 \quad 5 \quad 6 \quad 7 \\
+ \text{MORE} & \quad 1 \quad 0 \quad 8 \quad 5 \\
\hline
\text{MONEY} & \quad 1 \quad 0 \quad 6 \quad 5 \quad 2
\end{align*}$

Modeling

Formalize the problem as a constraint problem:

- number of variables: $n$
- 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

• number of variables: 8

• constraints:

\[ c_1 = \{(S,E,N,D,M,O,R,Y) \in \mathbb{Z}^8 \mid 0 \leq S,\ldots,Y \leq 9 \} \]

\[ c_2 = \{(S,E,N,D,M,O,R,Y) \in \mathbb{Z}^8 \mid 
1000*S + 100*E + 10*N + D 
+ 1000*M + 100*O + 10*R + E 
= 10000*M + 1000*O + 100*N + 10*E + Y \} \]

A Model for MONEY (continued)

• more constraints

\[ c_3 = \{(S,E,N,D,M,O,R,Y) \in \mathbb{Z}^8 \mid S \neq 0 \} \]

\[ c_4 = \{(S,E,N,D,M,O,R,Y) \in \mathbb{Z}^8 \mid M \neq 0 \} \]

\[ c_5 = \{(S,E,N,D,M,O,R,Y) \in \mathbb{Z}^8 \mid S \ldots Y \text{ all different} \} \]

Solution for MONEY

\[ c_1 = \{(S,E,N,D,M,O,R,Y) \in \mathbb{Z}^8 \mid \text{OS}\ldots\text{YS}9 \} \]

\[ c_2 = \{(S,E,N,D,M,O,R,Y) \in \mathbb{Z}^8 \mid 
1000*S + 100*E + 10*N + D 
+ 1000*M + 100*O + 10*R + E 
+ 1000*M + 1000*O + 100*N + 10*E + Y \} \]

\[ c_3 = \{(S,E,N,D,M,O,R,Y) \in \mathbb{Z}^8 \mid S \neq 0 \} \]

\[ c_4 = \{(S,E,N,D,M,O,R,Y) \in \mathbb{Z}^8 \mid M \neq 0 \} \]

\[ c_5 = \{(S,E,N,D,M,O,R,Y) \in \mathbb{Z}^8 \mid S \ldots Y \text{ all different} \} \]

Solution: (9,5,6,7,1,0,8,2) ∈ \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.

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

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

Example: SATisfiability

- Given a set of propositions containing variables, find an assignment of the variables to {false, true} that satisfies them.
- Example, the clauses:
  - A or B or ~C, ~A or D
  - (equivalent to C -> A or B, D -> A)
- 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

Formal definition of a constraint network (CN)

A constraint network (CN) 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 $R_i \subseteq D_{i1} \times D_{i2} \times \ldots \times D_{ik}$
- **Unary** constraint: only involves one variable
- **Binary** constraint: only involves two variables

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 $S$
  - 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.
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.

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 if there is a constraint involving the two variables.
  - Unary constraint appears as self-referential arc

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

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

```
  1  2  3

  4

  5
```
Running Example: XWORD Puzzle

- **Variables and their domains**
  - $X_1$ is 1 across $D_1$ is 5-letter words
  - $X_2$ is 2 down $D_2$ is 4-letter words
  - $X_3$ is 3 down $D_3$ is 3-letter words
  - $X_4$ is 4 across $D_4$ is 4-letter words
  - $X_5$ is 5 across $D_5$ is 2-letter words

- **Constraints (implicit/intensional)**
  - $C_{12}$ is “the 3rd letter of $X_1$ must equal the 1st letter of $X_2$”
  - $C_{24}$ is “the 5th letter of $X_1$ must equal the 1st letter of $X_3$”
  - $C_{25}$ is ...
  - $C_{34}$ is ...

Backtracking: XWORD

Elements of Constraint Programming

Exploiting constraints during tree search

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

S ∈ {0..9}  
E ∈ {0..9}  
N ∈ {0..9}  
D ∈ {0..9}  
M ∈ {0..9}  
O ∈ {0..9}  
R ∈ {0..9}  
Y ∈ {0..9}  

1000*S + 100*E + 10*N + D  
+ 1000*M + 100*O + 10*R + E  
= 10000*M + 1000*O + 100*N + 10*E + Y
$S \in \{9\}$
$E \in \{4, 7\}$
$N \in \{5, 8\}$
$D \in \{2, 8\}$
$M \in \{1\}$
$O \in \{0\}$
$R \in \{2, 8\}$
$Y \in \{2, 8\}$

$1000 \times S + 100 \times E + 10 \times N + D$
$+ 100 \times M + 10 \times O + 10 \times R + E$
$= 10000 \times M + 1000 \times O + 100 \times N + 10 \times E + Y$

$S = 9$
$E \in \{4, 7\}$
$N \in \{5, 8\}$
$D \in \{2, 8\}$
$M = 1$
$O = 0$
$R \in \{2, 8\}$
$Y \in \{2, 8\}$

$1000 \times S + 100 \times E + 10 \times N + D$
$+ 100 \times M + 10 \times O + 10 \times R + E$
$= 10000 \times M + 1000 \times O + 100 \times N + 10 \times E + Y$

$S = 9$
$E \in \{4, 7\}$
$N \in \{5, 8\}$
$D \in \{2, 8\}$
$M = 1$
$O = 0$
$R \in \{2, 8\}$
$Y \in \{2, 8\}$

$1000 \times S + 100 \times E + 10 \times N + D$
$+ 100 \times M + 10 \times O + 10 \times R + E$
$= 10000 \times M + 1000 \times O + 100 \times N + 10 \times E + Y$
Branch

Propagate

Complete Search Tree
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

Finite Domain Constraint Satisfaction Problems

A finite domain constraint problem consists of:

• number of variables: $n$
• constraints: $c_1, \ldots, c_m \subseteq \mathbb{Z}^n$

The problem is to find

$a = (v_1, \ldots, v_n) \in \mathbb{Z}^n$ such that

$a \in c_i$, for all $1 \leq i \leq m$

Programming Systems for Finite Domain Constraint Programming

• Finite domain constraint programming library
  – PECOS (Puget 1992)
  – ILOG Solver (Puget 1993)
• Finite domain constraint programming languages
  – CHIP (Dincbas, Hentenryck, Simonis, Aggoun 1988)
  – SICStus Prolog (Haridi, Carlson 1995)
  – Oz (Smolka and others 1995)
  – OPL (van Hentenryck 1998)
**Constraint Solving**

Given: a satisfiable constraint C and a new constraint C'. Constraint solving means deciding whether $C \land C'$ is satisfiable.

Example:

$C$: $n > 2$

$C'$: $a^n + b^n = c^n$

---

**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 $X \in S$, 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

all different (S, E, N, D, M, O, R, Y)

1000*S + 100*E + 10*N + D
+ 1000*M + 100*O + 10*R + E
= 10000*M + 1000*O + 100*N + 10*E + Y

and so on and so on

Basic Constraints and Propagators

all different (S, E, N, D, M, O, R, Y)

1000*S + 100*E + 10*N + D
+ 1000*M + 100*O + 10*R + E
= 10000*M + 1000*O + 100*N + 10*E + Y

Standard Search Formulation

Let's start with the straightforward approach, then fix it:

States are defined by the values assigned so far

• Initial state: the empty assignment {}
• Successor function: assign a value to an unassigned variable that does not conflict with current assignment
• Fail if no legal assignments
• Goal test: the current assignment is complete

1. This is the same for all CSPs
2. Every solution appears at depth n with n variables
3. Path is irrelevant, so can also use complete-state formulation
4. b = (n - !)d in depth, hence n!d 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 CSPs with single-variable assignments is called backtracking search
- Backtracking search is the basic uninformed algorithm for CSPs
- Can solve n-queens for \( n = 25 \)

Backtracking example

```python
# Example of backtracking search

def backtracking_search(problem):
    # Implementation of backtracking search
    pass
```
Backtracking example

Backtracking example

Backtracking example
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
Forward checking

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

Constraint propagation

- Forward checking propagates information from assigned to unassigned variables, but doesn't provide early detection for all failures:

- NT and SA cannot both be blue!
- Constraint propagation repeatedly enforces constraints locally
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?

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.

Completeness of Propagation

- General arithmetic constraints are undecidable (Hilbert’s Tenth Problem).
- Propagation may not exhibit all inconsistencies.
- Example:
  \[ c_1: n > 2 \]
  \[ c_2: \ a^n + b^2 = c^3 \]
Example: Complete All Different

• C: \( w \in \{1, 2, 3, 4\} \)
  \( x \in \{2, 3, 4\} \)
  \( y \in \{2, 3, 4\} \)
  \( z \in \{2, 3, 4\} \)
• P: all_different(w, x, y, z)

Example: Complete All Different

• C: \( w \in \{1, 2, 3, 4\} \)
  \( x \in \{2, 3, 4\} \)
  \( y \in \{2, 3, 4\} \)
  \( z \in \{2, 3, 4\} \)
• P: all_different(w, x, y, z)
• Most efficient known algorithm: \( O(|X|^2 d_{\max}^2) \)
  Regin [1994], using graph matching

Basic Constraints vs. Propagators

• Basic constraints
  – are conjunctions of constraints of the form \( X \in S \), where \( S \) is a finite set of integers
  – enjoy complete constraint solving
• Propagators
  – can be arbitrarily expressive (arithmetic, symbolic)
  – implementation typically fast but incomplete
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 - Revisited, Operations Research, to appear

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 the other
- 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 $2 \times 9$ 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)

• 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 (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 (cont’d)

• 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 Clem in the second date.
• Duke has bye in the first date 16.
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.

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

<table>
<thead>
<tr>
<th></th>
<th>D</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

=>

<table>
<thead>
<tr>
<th></th>
<th>D</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>E</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>D</th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

Optimization

- 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

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

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

holds and $\textit{MONEY}$ is maximal.

---

**Modeling**

Formalize the problem as a constraint optimization problem:

- **Number of variables:** $n$
- **Constraints:** $c_1, \ldots, c_n \in \mathbb{Z}^n$
- **Optimization constraints:** $d_1, \ldots, d_m \in \mathbb{Z}^n \rightarrow 2^{\mathbb{Z}^n}$

Given a solution $a$, and an optimization constraint $d_i$, the constraint $d_i(a) \in \mathbb{Z}^n$ contains only those assignments $b$ for which $b$ is better than $a$.

---

**A Model for MONEY**

- **number of variables:** 8
- **constraints:**

\[
\begin{align*}
c_1 &= \{(S, E, N, D, M, O, T, Y) \in \mathbb{Z}^8 \mid 0 \leq S, \ldots, Y \leq 9 \} \\
c_2 &= \{(S, E, N, D, M, O, T, Y) \in \mathbb{Z}^8 \mid \\
&\quad 1000 \times S + 100 \times E + 10 \times N + D \\
&\quad + 1000 \times M + 100 \times O + 10 \times S + T \\
&\quad = 1000 \times M + 100 \times O + 10 \times S + 10 \times E + Y \}
\end{align*}
\]
A Model for MONEY (continued)

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

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

Propagation Algorithms

Identify a propagation algorithm to implement the optimization constraints

Example: SEND + MOST = MONEY

\[
  d: (s, e, n, d, m, o, t, y) \rightarrow \\
  \{(S, E, N, D, M, O, T, Y) \in \mathbb{Z}^8 \times \mathbb{Z}^8 \mid \\
  10000*m + 1000*o + 100*n + 10*e + y \\
  < 10000*M + 1000*O + 100*N + 10*E + Y \}
  \]

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: Naïve 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 x of X, there is a value y for Y that satisfies the constraint represented by the arc.
  – A graph is arc-consistent if all arcs are arc-consistent

• 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**
  • \( X \rightarrow Y \) is consistent if for every value x of X there is some allowed y
Arc consistency

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

• If \( X \) loses a value, neighbors of \( X \) need to be rechecked

Arc consistency

• Simplest form of propagation makes each arc consistent
• \( X \rightarrow Y \) is consistent iff
  for every value \( x \) 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

Functions AC-3() removes the CSP, possibly with reduced domains
Input: \( a \), a binary CSP with variables \( X_1, X_2, \ldots, X_n \)
Local variables: queue, a queue of arcs, initially all the arcs in \( a \)

while queue is not empty do
  \((X_i, X_j) \leftarrow \) front of queue
  if REM-DISCORSER-VALUE(\( X_i, X_j \)) then
    for each \( X_k \) in Neighbors(\( X_i \)) do
      add \((X_i, X_k)\) to queue

Function REM-DISCORRER-VALUE(\( X_i, X_j \)) returns true if removers a value
\( \text{removed} \leftarrow \) false
for each \( x \) in Domain(\( X_i \)) do
  if no value \( y \) in Domain(\( X_j \)) allows \( (x, y) \) to satisfy constraint(\( X_i, X_j \))
    then delete \( x \) from Domain(\( X_i \)); \( \text{removed} \leftarrow \) true
    return \( \text{removed} \)

• 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 = ...

Constraint propagation:
XWORD example

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 \( J \leq K \)
- Node consistency = strong 1-consistency
- Arc consistency = strong 2-consistency
- Path consistency = strong 3-consistency
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.

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

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.