Title: Constraint Satisfaction Problems
Required reading: AIMA: Chapter 5

Recommended reading:
— Introduction to CSPs (Bartak's on-line guide)
— "Algorithms for Constraints Satisfaction problems: A Survey" by Vipin Kumar. AI Magazine, Vol 13, No 1, 32-44, 1992.
— Constraint Programming: In Pursuit of the Holy Grail. Bartak

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
  Spring03́ 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 *4 courses at UNL*
- 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 $\left\{ \begin{array}{c} \text{relax} \\ \text{reinforce} \end{array} \right\}$ 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

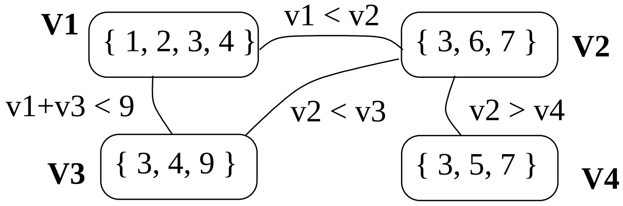
- Instantiating a variable: $V_i \leftarrow a$ where $a \in D_{V_i}$
- Partial assignment
- Consistent assignment
- Objective function (constrained optimization problem)

Representation

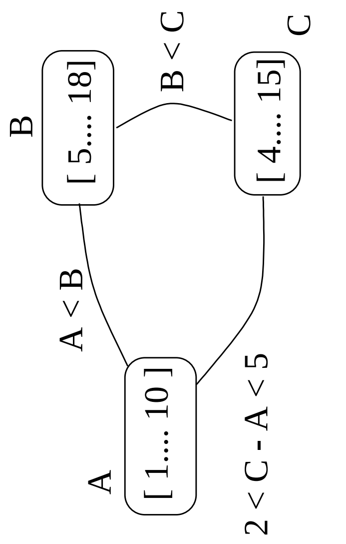
Given $\mathcal{P} = (\mathcal{V}, \mathcal{D}, \mathcal{C}) \begin{cases} \mathcal{V} = \{V_1, V_2, \ldots, V_n\} \\ \mathcal{D} = \{D_{V_1}, D_{V_2}, \ldots, D_{V_n}\} \\ \mathcal{C} \ \text{ set of constraints} \end{cases}$

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

Constraint **graph**



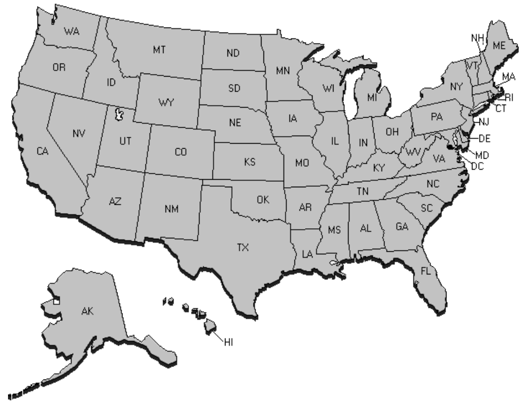
Example I: Temporal reasoning

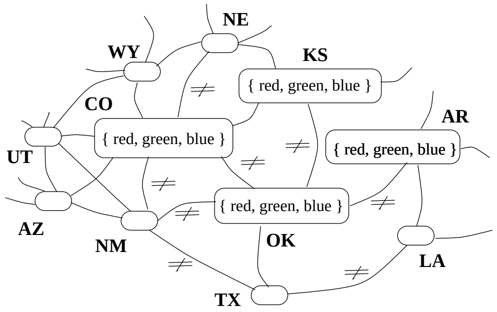


$\longrightarrow$ c-A $\in$ [2, 5] is a constraint of bounded differences

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) → DFS!!
— Path for reaching a solution is irrelevant
— A complete-state formulation is OK
— local-search techniques applicable

Domain types

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

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

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

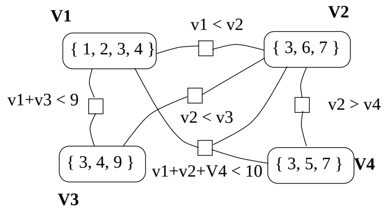
Constraint arity

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

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

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

Representation: Constraint network



Constraint language

Constraints can be defined

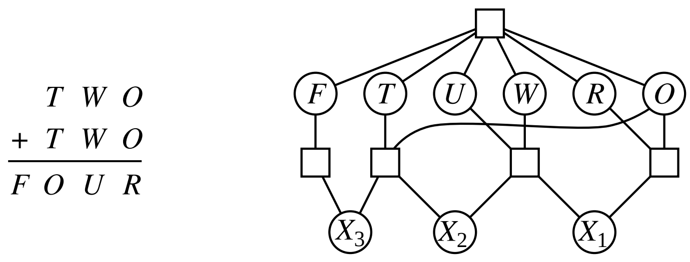
- Extensionally: all allowed tuples are listed
  practical for defining arbitrary constraints
- Intensionally: when it is not practical (or even possible) to list all tuples
  $\rightarrow$ define types of common constraints, which can be used repeatedly

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

$D_{X1} = D_{X2} = D_{X3} = \{0, 1\}$

$D_F = D_T = D_U = D_V = D_R = D_O = [0, 9]$



(a) (b)

O + O = R + 10 X1

X1 + W + W = U + 10 X2

X2 + T +T = O + 10 X3

X3 = F

Alldiff({F, D, U, V, R, O})

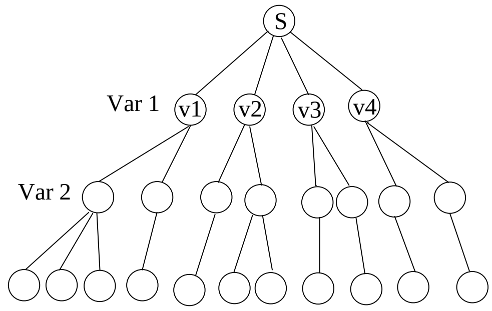
How to solve a CSP?

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..

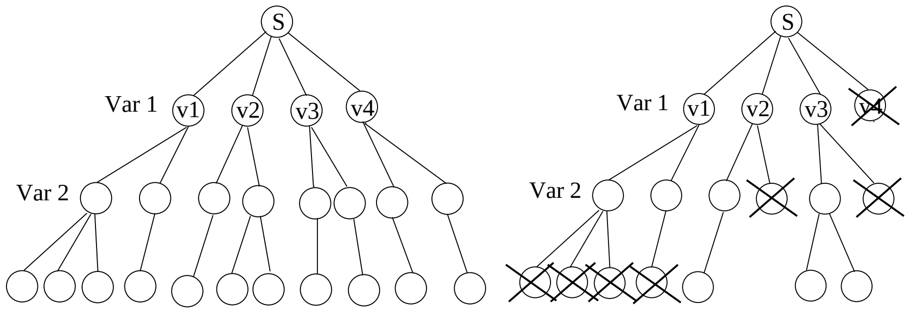


For n variables, each of domain size d:
- Maximum depth? *fixed!*
- Maximum number of paths? *size of search space, size of CSP*

Back-checking

Systematic search generates d^n possibilities

Are all possible combinations acceptable?



→ Expand a partial solution only when consistent

⟶ **early pruning**

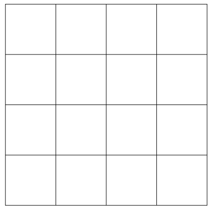
Before looking at search..

Consider

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

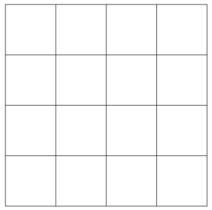
Importance of modeling

N-**queens**: formulation 1



Variables?
Domains?
Size of CSP?

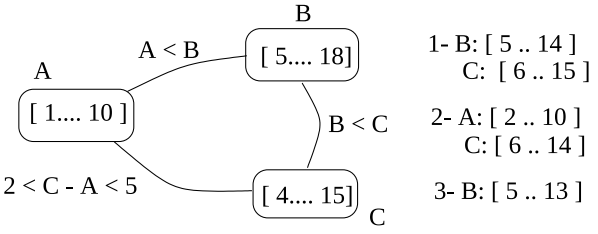
N-**queens**: formulation 2



variables?
domains?
size of csp?

Constraint checking

⟶ Constraint filtering, constraint checking, etc..
eliminate non-acceptable tuples prior to search



⟶ Arc-Consistency:

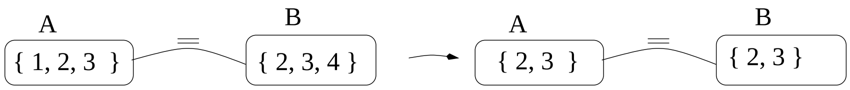
$$\text{AC}(C_{V_1,V_2}) = \text{Revise}(V_1,V_2) \text{ and } \text{Revise}(V_2,V_1)$$

⟶ CSP is AC when all constraints are AC.

⟶ Algorithms: AC-1, AC-2, **AC-3**, ..., **AC-7** and back to **AC-3**

⟶ AC-3: $O(n^2d^3)$

Warning: arc-consistency does not solve the problem



(A=2) $\wedge$ (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

Levels of consistency

Node consistency: every value in the domain of a variable is consistent with the unary constraints defined on the variable

Arc-consistency: For any value in the domain of any variable, there is at least one value in the domain of any other variable with which it is consistent.

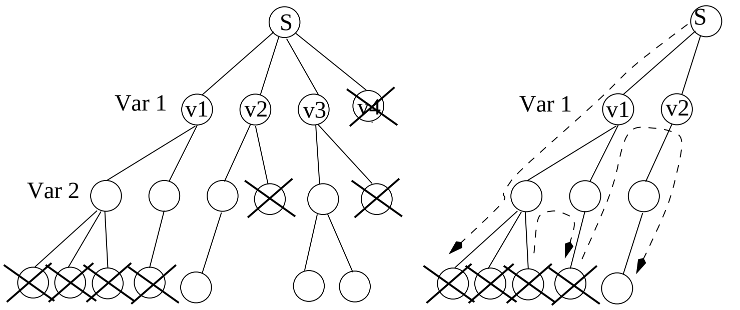
3-consistency: For any two consistent values in the domains of any two variables, there is at least one value in the domain of any third variable with which they are consistent.

k-consistency: $(k \leq n)$
For any (k-1) consistent values in the domains of any (k-1) variables, there is at least one value in the domain of any k^{th} variable with which they are consistent.

Strong k-consistency: k-consistency $\forall i \leq k$

Chronological backtracking

What if only <u>one</u> solution is needed?

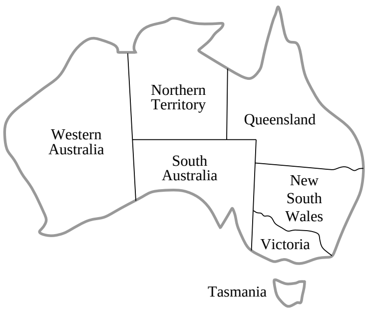


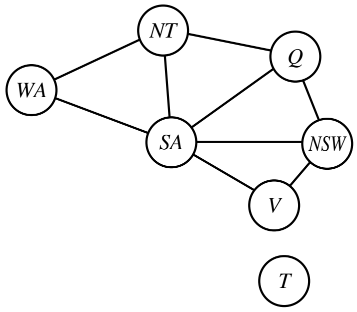
$\longrightarrow$ **Depth-first search & chronological backtracking]]** $\longrightarrow$

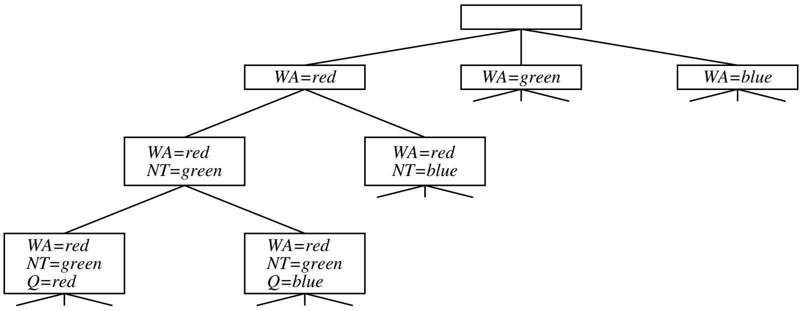
Terms: current variable V_c, past variables $\mathcal{V}_p$, future variables $\mathcal{V}_f$, current path

$\rightarrow$ 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

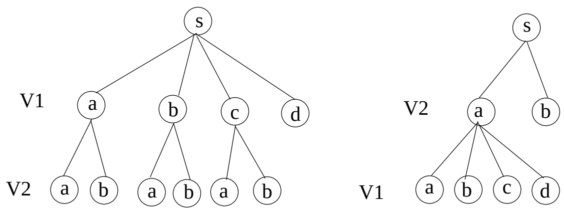
General purpose methods for:

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: $\begin{cases} \text{most \underline{constrained} variable first (reduce branching factor)} \\ \text{most \underline{promising} value first (find quickly first solution)} \end{cases}$

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:

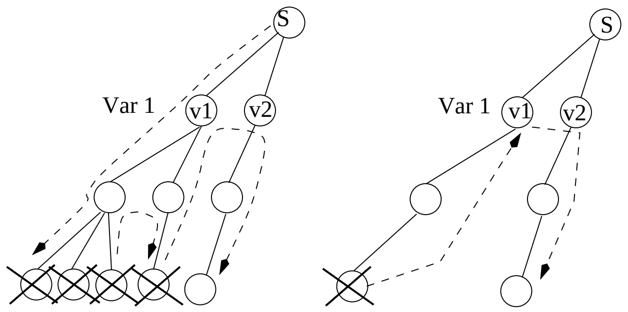
- min-conflict [Minton, 92]
- promise [Geelen, 94], etc.

Strategies for $\left\{\begin{array}{l}\text{variable ordering}\\ \text{value ordering}\end{array}\right\}$ could be $\left\{\begin{array}{l}\text{static}\\ \text{dynamic}\end{array}\right.$

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

Terminology

- V_c is the current variable
- $\mathcal{V}_f$ is the set of future variables, V_f is a future variable
- Instantiate V_c, update the domains of (some) future variables

Strategies

- Forward checking (FC): partial look-ahead
- Directed arc-consistency checking (DAC): partial look-ahead
- Maintaining Arc-Consistency (MAC): full look-ahead

→ Special data structures can be used to refresh filtered domains upon backtracking [Prosser, 93]

Forward checking (FC)

→ Apply REVISE(V_f, V_c) to the each variable V_f connected to V_c
→ In AIMA, it is REMOVE-INCONSISTENT-VALUES(V_f, V_c)

Procedure:

- Instantiate V_c
- Apply REVISE(V_f, V_c) to the each variable V_f

Direct Arc-Consistency (DAC)

→ Repeat forward checking on all $V_f \in \mathcal{V}_f$ while respecting order
→ Applicable under static ordering

Procedure:

- Choose a variable ordering
- Instantiate V_c
- Apply FC to V_c
- Move to next variable V_f in ordering, and apply FC to V_f.
  Repeat for all variables in $\mathcal{V}_f$ in the specified order.

Maintaining Arc-Consistency (MAC)

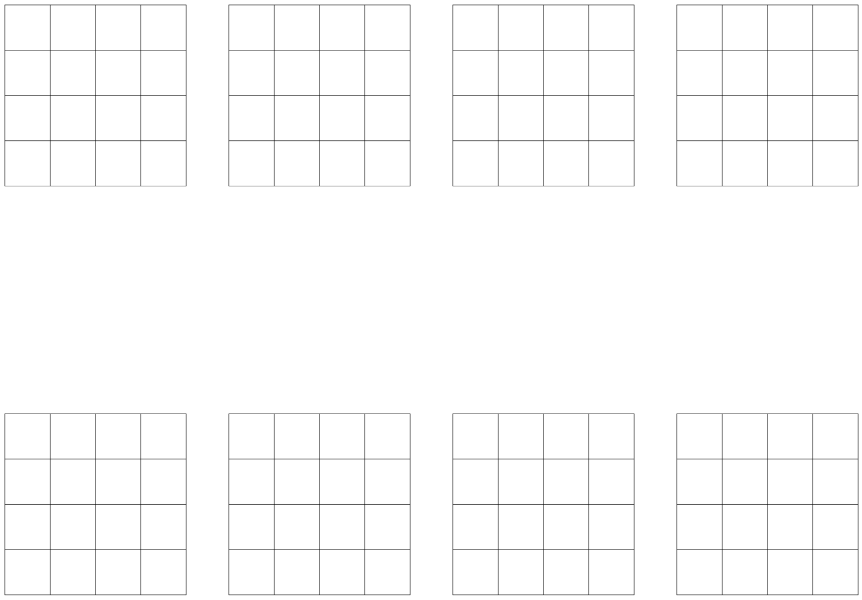
→ Maintain AC in the subproblem induced by $\mathcal{V}_f \cup \{V_c\}$
→ In practice, useful when problem has few, tight constraints

Procedure:

- Instantiate V_c
- Apply AC-3($\mathcal{V}_f \cup \{V_c\}$)
  Every constraint revision uses two operations: Revise(V_a, V_b) and Revise(V_b, V_a)
  Updates domains of all variables in subproblems

Search (V) *Forward checking*

Why not filter right away effects of an action?



CSP: a decision problem (NP-complete)

1- Modeling:
— abstraction and reformulation

2- Preprocessing techniques:
— eliminate non-acceptable tuples <u>prior</u> to search

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

4- Search 'hybrids':
— Mixing preprocessing with search: FC, DAC, MAC

Non-systematic search

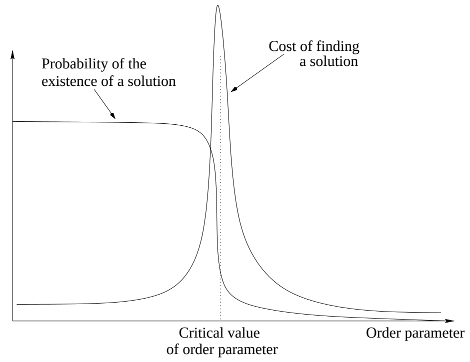
- **Methodology:** Iterative repair, local search: modifies a global but <u>**inconsistent**</u> solution to decrease the number of violated constraints
- **Example:** MIN-CONFLICTS algorithm in Fig 5.8, page 151. Choose (randomly) a variable in a broken constraint, and change its value using the min-conflict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

$\longrightarrow$ Anytime algorithm
$\longrightarrow$ Strategies to avoid getting trapped: RandomWalk
$\longrightarrow$ Strategies to recover: Break-Out, Random restart, etc.
$\longrightarrow$ 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
→ 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 *but also user-defined*
  - 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: *i*2 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: *i*2 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*