Constraint Satisfaction Problems

GEIST

Katadra Automatyki
Akademia Górniczo-Hutnicza

22 listopada 2011

Lecture Plan

1. **Introduction to CSP**
   - Intuitions through Examples
2. Backtracking Search and Heuristics
3. Exploring Problem Structure
4. Conclusions

References

1. Stuart J. Russel, Peter Norvig: *Artificial Intelligence. A Modern Approach.* Second Edition. Prentice Hall, New Jersey, 2003. Chapter 5.
2. Rina Dechter: *Constraint Processing*.Morgan Kaufmann Publishers; An imprint of Elsevier Science, San Francisco, 2003.
3. Krzysztof R. Apt: *Principles of Constraint Programming*. Cambridge University Press, Cambridge, 2003.
4. Krzysztof R. Apt and Mark G. Wallace: *Constraint Logic Programming using* ECL^iPS^e. Cambridge University Press, Cambridge, 2007.
5. Francesca Rossi, Peter van Beek and Toby Walsh (Eds.): Handbook of Constraint Programming. Elsevier, 2006.
6. Antoni Niederliński: *Programowanie w logice z ograniczeniami. Łagodne wprowadzenie dla platformy ECLiPSe*. WWW: pkjs.com.pl, Gliwice, 2010.

- `http://www.pwlzo.pl/`

Some Example Problems

What in common?

- Map Coloring Problems,
- Cryptoarithmetic Problems,
- Scheduling Problems,
- Timetable Design Problems,
- Configuration Problems (hardware, software),
- Radio Frequency Assignment,
- Crossword Puzzles,
- Sudoku,
- Einstein or Zebra Probelm.

SEND + MORE = MONEY

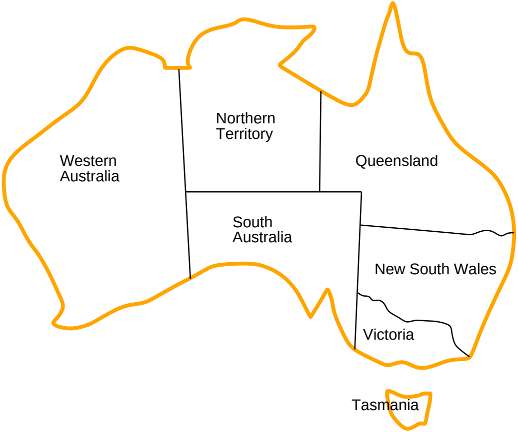
What in common?

- `SEND+MORE=MONEY`,
- Variables: S, E, N, D, M, O, R, Y.
- Domains: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$,
- All variables are different,
- $S \neq 0$, $M \neq 0$,
- All constraints must be satisfied.

Characteristic Features

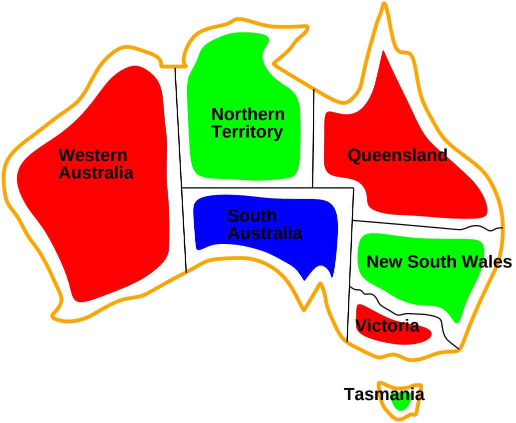
- only the **final solution** counts (no path to it),
- there can be 0, 1 or many solutions (all are equivalent),
- **strong combinatorial explosion**.

Australia — Map Colouring Problem



- **Variables:** WA, NT, Q, NSW, V, SA, T;
- **Domains:** $D_i = \{red, green, blue\}$;
- **Constraints:** adjacent regions must have different colors e.g., $WA \neq NT$ (if the language allows this), or $(WA, NT) \in \{(red, green), (red, blue), (green, red), (green, blue), \ldots\}$.

Australia — Map Coloring Problem Solution



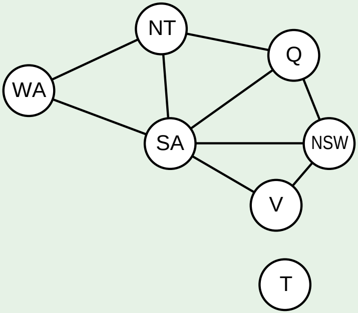
Solutions are assignments satisfying all constraints, e.g., $\{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green\}$

Tools - Constraint Graph

Constraint Graph — represents constraints

- variables ⟶ nodes,
- binary constraints ⟶ arcs.

Australia: Constraint Graph



Varietes of CSPs

Discrete variables

- Boolean CSPs, incl. Boolean satisfiability (NP-complete)
  infinite domains (integers, strings, etc.)
- job scheduling, variables are start/end days for each job,
- need a *constraint language*, e.g., $StartJob_1 + 5 \leq StartJob_3$,
- *linear* constraints solvable, *nonlinear* undecidable.

Continuous variables

- e.g., start/end times for Hubble Telescope observations,
- linear constraints solvable in poly time by LP methods.

Problem: Combinatorial Explosion!

Potential solutions number $= card(D_1 \times D_2 \times \ldots \times D_n)$
Computational complexity $= O(d^n)$

Varieties of constraints

Typical types of constraints

- **Unary** constraints involve a single variable, e.g.

$$SA \neq green,$$

- **Binary** constraints involve pairs of variables, e.g.

$$SA \neq WA,$$

- **Higher-order** constraints involve 3 or more variables, e.g. cryptarithmetic column constraints of the form

$$Z = mod(X + Y + C), \qquad C' = carry(X + Y + C)$$

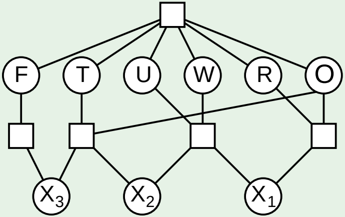
- **Preferences** (soft constraints), e.g., *red* is better than *green* often representable by a cost for each variable assignment $\rightarrow$ constrained optimization problems.

Example: Cryptarithmetic

A simple example

```
    T W O
  + T W O
  -------
  F O U R
```

(a)



(b)

- Variables: $F\ T\ U\ W\ R\ O\ X_1\ X_2\ X_3$
- Domains: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$
- Constraints: $alldiff(F, T, U, W, R, O)$
  $O + O = R + 10 \cdot X_1$, etc.

Lecture Plan

1. Introduction to CSP
2. **Backtracking Search and Heuristics**
   - Real Worls CPSs
   - Formal Stuff
   - Backtracking search
   - Improving backtacking search: heuristics
3. Exploring Problem Structure
4. Conclusions

Real World CPSs

Example Application Areas

- Assignment problems
  e.g., who teaches what class?,
- Timetabling problems
  e.g., which class is offered when and where?,
- Hardware configuration,
- Spreadsheets,
- Transportation scheduling,
- Factory scheduling,
- Floorplanning,
- Scheduling Problems,
- Timetable Design Problems,
- Configuration Problems (e.g. Renault Megane Case),
- Radio Frequency Assignment,
- Packing problems,
- Layout problems.

CSP: Definition

CSP statement

- $X = \{X_1, X_2, \ldots, X_n\}$ — a set of variables,
- $D = \{D_1, D_2, \ldots, D_n\}$ — their domains,
- $C = \{(S_i, R_i)\colon i = 1, 2, \ldots, n\}$ — constraints,
  - S_i — scope — a selection of variables,
  - R_i — relation defined over Cartesian Product of domains appropriate for the scope variables,

CSP solution

A solution to CSP given by (X, D, C) is any assignment of values to variables of X of the form

$$\{X_1 = d_1, X_2 = d_2, \ldots, X_n = d_n\},$$

such that $d_i \in D_i$, and for any constraint in $(S_i, R_i) \in C$, R_i is satisfied by the appropriate projection of the solution vector $(d_1, d_2, \ldots, d_n)$ over variables of S_i.

Standard Search Formulation (incremental)

Basic approach

States are defined by the values assigned so far:

- **Initial state**: the empty assignment, $\emptyset$,
- **Successor function**: assign a value to an unassigned variable that does not conflict with current assignment,
- $\implies$ fail if no legal assignments (not fixable!),
- **Goal test**: the current assignment is complete and consistent, i.e.
- **Consistency**': all the constraints are satisfied.

1. This is the same for all CSPs!
2. Every solution appears at depth n with n variables,
3. $\implies$ use depth-first search (DFS),
4. Path is irrelevant, so can also use complete-state formulation,
5. $b = (n - \ell)d$ at depth ℓ, hence $n!d^n$ leaves!!!

Backtracking Search

Commutative variable assignment

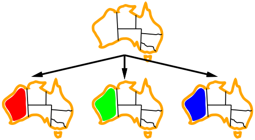
Variable assignments are *commutative*, i.e.,
$WA = red$ then $NT = green$ same as $NT = green$ then $WA = red$

Search

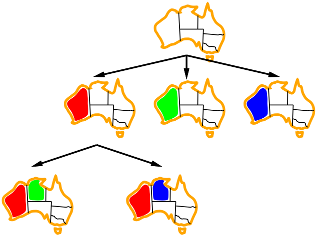
- Only need to consider assignments to a single variable at each node,
- $\implies b = d$ and there are d^n leaves,
- Depth-first search for CSPs with single-variable assignments is called *backtracking search*
- Backtracking search is the basic uninformed algorithm for CSPs,
- Can solve n-queens for $n \approx 25$.

Bactraking Search

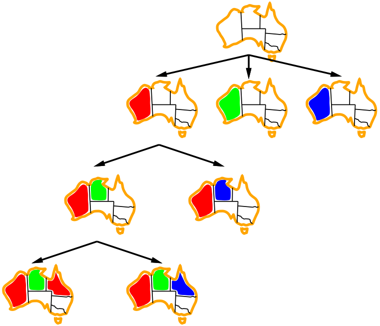
Bactraking Search



Bactraking Search



Bactraking Search



Improving Backtracking Efficiency

What can be improved?

General-purpose methods can give huge gains in speed:

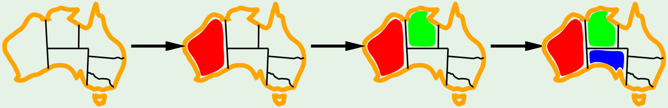
1. Which variable should be assigned next?
2. In what order should its values be tried?
3. Can we detect inevitable failure early?
4. Can we take advantage of problem structure?

MVR: Minimum Remaining Values

MVR heuristic (MCV: Most Constrained Variable)

- choose the variable with the *fewest* legal values

Example

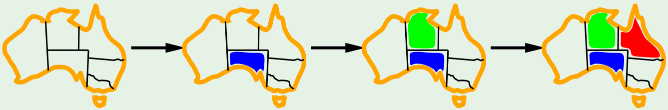


Degree Heuristic

Tie-breaker among MRV variables

- choose the variable with the *most constraints* on remaining variables

Example

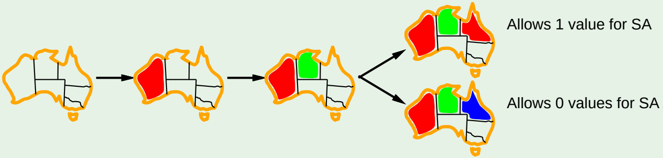


LCR: Least Constraining Value

LCR heuristic

- given a variable, choose the least constraining value: the one that rules out the fewest values in the remaining variables

Example



These simple heuristics makes 1000 queens feasible!

Forward Checking

FC idea

- keep track of remaining legal values for unassigned variables,
- terminate search when any variable has no legal values.

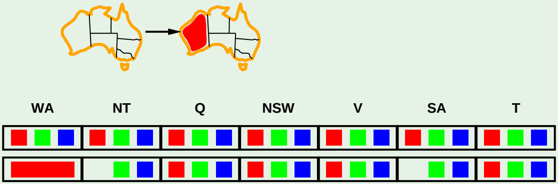
Example

Forward Checking

FC idea

- keep track of remaining legal values for unassigned variables,
- terminate search when any variable has no legal values.

Example

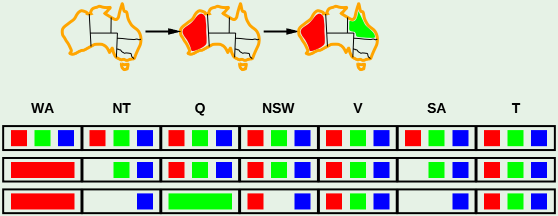


Forward Checking

FC idea

- keep track of remaining legal values for unassigned variables,
- terminate search when any variable has no legal values.

Example

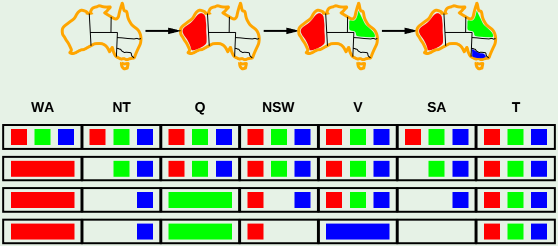


Forward Checking

FC idea

- keep track of remaining legal values for unassigned variables,
- terminate search when any variable has no legal values.

Example

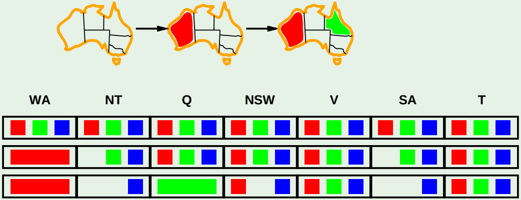


Constraint Propagation

Forward checking limitations

- forward checking propagates information from assigned to unassigned variables,
- but does not provide early detection for all failures:

Example



Hidden problem

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

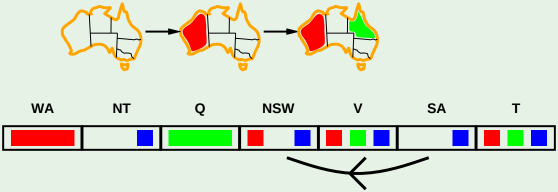
Arc Consistency

AC idea

Simplest form of propagation makes each arc **consistent**

- $X \to Y$ is consistent iff
  for *every* value $x \in X$ there is *some* allowed $y \in Y$

Example



AC Tips

- 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.

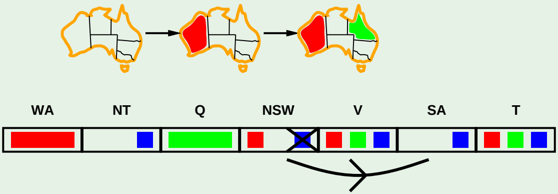
Arc Consistency

AC idea

Simplest form of propagation makes each arc **consistent**

- $X \rightarrow Y$ is consistent iff
  for *every* value $x \in X$ there is *some* allowed $y \in Y$

Example



AC Tips

- 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.

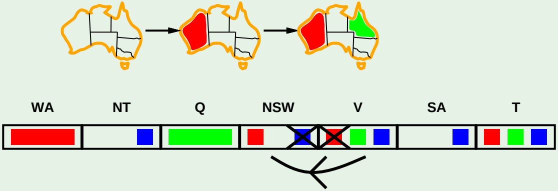
Arc Consistency

AC idea

Simplest form of propagation makes each arc **consistent**

- $X \to Y$ is consistent iff
  for *every* value $x \in X$ there is *some* allowed $y \in Y$

Example



AC Tips

- 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.

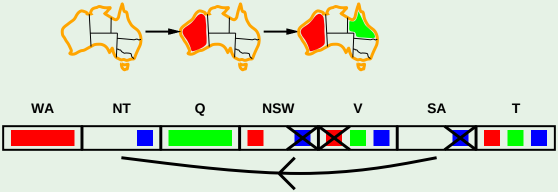
Arc Consistency

AC idea

Simplest form of propagation makes each arc **consistent**

- $X \rightarrow Y$ is consistent iff
  for *every* value $x \in X$ there is *some* allowed $y \in Y$

Example



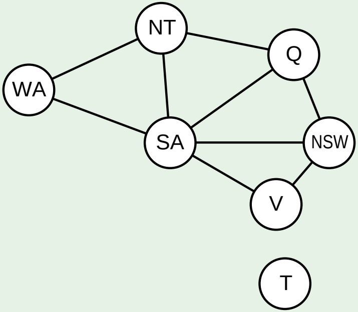
AC Tips

- 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.

Lecture Plan

Problem Structure

Australia: problem structure



- Tasmania and mainland are *independent subproblems*. They are identifiable as *connected components* of constraint graph.

Problem structure contd.

Problem decomposition

- Problem decoposition is always a good idea!
- Separated problems are better then overlapping ones!
- Many small problems are better then few but bigger!

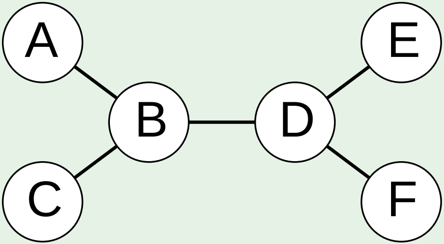
Some hints

- Suppose each subproblem has c variables out of n total.
- Worst-case solution cost is $n/c \cdot d^c$, *linear* in n

E.g., $n = 80$, $d = 2$, $c = 20$
2^{80} = 4 billion years at 10 million nodes/sec
$4 \cdot 2^{20}$ = 0.4 seconds at 10 million nodes/sec

Tree-structured CSPs

Example



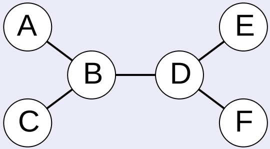
A theorem for trees

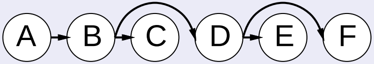
Theorem: if the constraint graph has no loops, the CSP can be solved in $O(n\,d^2)$ time.
Compare to general CSPs, where worst-case time is $O(d^n)$
This property also applies to logical and probabilistic reasoning: an important example of the relation between syntactic restrictions and the complexity of reasoning.

Algorithm for tree-structured CSPs

Algorithm outline

1. Choose a variable as root, order variables from root to leaves such that every node's parent precedes it in the ordering.





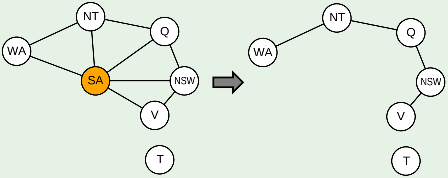
2. For j from n down to 2, apply $\mathit{Arc\text{-}Consistency}(\mathit{Parent}(X_j), X_j)$.
3. For j from 1 to n, assign X_j consistently with $\mathit{Parent}(X_j)$.

Nearly tree-structured CSPs

Conditioning

Conditioning: instantiate a variable, prune its neighbors' domains.

Example



Cutset conditioning

Cutset conditioning: instantiate (in all ways) a set of variables such that the remaining constraint graph is a tree

Cutset size $c \implies$ runtime $O(d^c \cdot (n-c)d^2)$, very fast for small c.

Iterative algorithms for CSPs

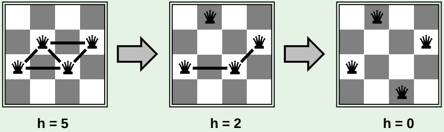
Iterative procedures

- Hill-climbing, simulated annealing typically work with “complete” states, i.e., all variables assigned;
- To apply to CSPs:
  allow states with unsatisfied constraints operators *reassign* variable values;
- Variable selection: randomly select any conflicted variable;
- Value selection by *min-conflicts* heuristic:
  choose value that violates the fewest constraints i.e., hillclimb with $h(n)$ = total number of violated constraints.

Example: 4-Queens

4-queens

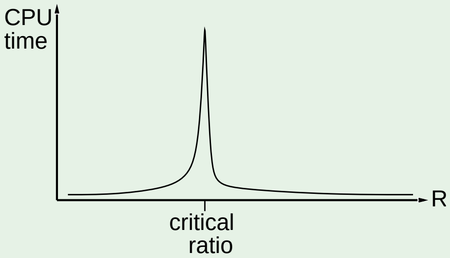
- **States**: 4 queens in 4 columns ($4^4 = 256$ states).
- **Operators**: move queen in column.
- **Goal test**: no attacks.
- **Evaluation**: $h(n)$ = number of attacks.



Performance of min-conflicts

- Given random initial state, can solve n-queens in almost constant time for arbitrary n with high probability (e.g., $n =$ 10,000,000)
- The same appears to be true for any randomly-generated CSP *except* in a narrow range of the ratio

$$R = \frac{\text{number of constraints}}{\text{number of variables}}$$



Lecture Plan

Final Remarks

In summary:

1. CSPs are a special kind of problem:
   states defined by values of a fixed set of variables
   goal test defined by *constraints* on variable values.
2. Backtracking = depth-first search with one variable assigned per node.
3. Variable ordering and value selection heuristics help significantly.
4. Forward checking prevents assignments that guarantee later failure.
5. Constraint propagation (e.g., arc consistency) does additional work
   to constrain values and detect inconsistencies.
6. The CSP representation allows analysis of problem structure.
7. Tree-structured CSPs can be solved in linear time.
8. Iterative min-conflicts is usually effective in practice.