Artificial Intelligence

Blai Bonet

Universidad Simón Bolívar, Caracas, Venezuela

Constraint satisfaction problems (CSPs)

Goals for the lecture

- Constraint satisfaction problem (CSP)
- Types of CSPs and constraints
- Translation of CSPs
- Backtracking algorithms with heuristics for variable selection
- Inference: forward checking, arc consistency
- Solving CSPs by pure inference

Informal description

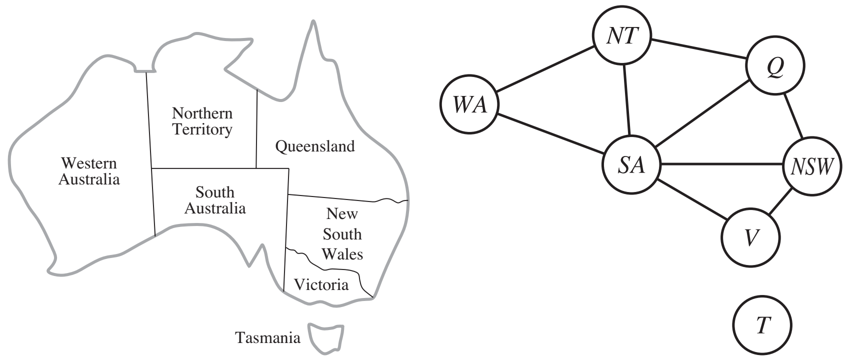
CSP is **assignment problem** defined by:

– a set of **variables** with **domains**

– a set of **constraints**

Task: find assignment of variables to values that **satisfy** all constraints

Example: Map coloring



[Image from Russell & Norvig. *Artificial Intelligence: A Modern Approach*]

Formal model

CSP is given by tuple $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ where:

– $\mathcal{X} = \{X_1, \ldots, X_n\}$ is finite set of variables

– $\mathcal{D} = \{D_1, \ldots, D_n\}$ is set of domains, domain D_i for variable X_i

– $\mathcal{C} = \{C_1, \ldots, C_m\}$ is set of constraints that specify **allowable combinations** of values

Each constraint C is pair $\langle \text{scope}, R \rangle$ where scope is tuple over X that specifies the variables involved in C, and $R \subseteq \prod_{X_i \in \text{scope}} D_i$ defines the allowable combinations for variables in scope

E.g., if X and Y are binary variables with domain $\{0, 1\}$, the constraint $\langle (X, Y), \{(0, 1), (1, 0)\} \rangle$ expresses $X \neq Y$

Formulation of map coloring

- Variables $\mathcal{X} = \{WA, NT, Q, NSW, V, SA, T\}$
- Domains for all variables given by colors red, $blue$ and $green$
- For each two variables X and Y **connected by edge**, there is a contraint with scope (X, Y) and the following relation that requires that the colors of X and Y must be different:

$$\{(red, blue), (red, green), (blue, red), (blue, green), (green, red), (green, blue)\}$$

Solving CSPs by search

CSPs can be solved by performing search on the space of **partial assignments** of variables to values:

– initial state is **empty assignment**

– goal states are **complete assignments** that satisfy all constraints

– successor function **extends** partial assignment with new variable, provided that resulting assignment is consistent (i.e. doesn't violate a constraint)

– uniform costs

If there are n variables, all goal states (if any) appear at depth n

IDA* is discarded. DFBnB could be considered but there are no **meaningful heuristics** since all costs are equal. We'll do a depth-first traversal but extended with some form of "inference" to **prune branches in search tree**

Alternative model for (local) search

Another search space is obtained by considering only **complete assignments** instead of partial ones

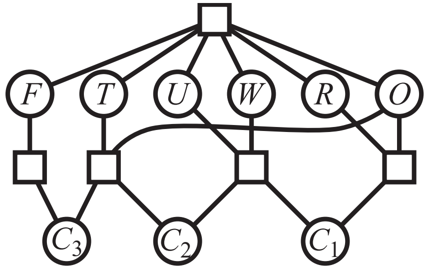
Edges connect assignment that differ in the assignment for one or more variables (typically just 1 variable)

Initial state is any assignment while goal states correspond to assignments that satisfy all constraints

Formulation used by **local search methods** that in some cases are very effecive but **incomplete**

Example: Cryptoarithmetic

$$\begin{array}{r} T\ W\ O \\ +\ T\ W\ O \\ \hline F\ O\ U\ R \end{array}$$

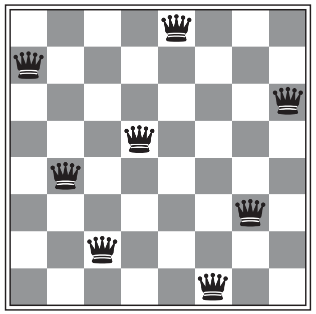


[Image from Russell & Norvig. *Artificial Intelligence: A Modern Approach*]

Example: 8-Queens

Place 8 queens in an empty chess board in a way that no queen attacks another

Can it be done?



[Image from Russell & Norvig. *Artificial Intelligence: A Modern Approach*]

CSP Variations

Simplest CSPs have **finite and discrete domains**

Infinite discrete domains can be considered, but constraints cannot be represented explicitly and **constraint languages** are used

Continuous domains such as real values can also be considered

Some special cases:

- Real-valued variables with **linear constraints** (e.g. $X + 3Y \leq Z$) can be solved efficiently with **linear programming**
- Integer-valued variables with linear constraints can be solved using **integer programming** methods (intractable in worst case)
- Special cases like real-valued variables with **convex constraints**

Constraint types and constraint graph

A constraint whose scope is singleton is **unary constraint**

A **binary constraint** relates two variables (scope size is 2)

A **constraint of order** k relates k variables; for $k > 2$, it is a **higher-order** constraint

Constraint graph for CSP $(\mathcal{X}, \mathcal{D}, \mathcal{C})$ is **(undirected) graph** with vertices given by $\mathcal{X}$ and edges (X_i, X_j) iff there is a constraint whose scope contains i and j

CSP with binary constraints

Any CSP $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ can be mapped into **equivalent** CSP $P' = (\mathcal{X}', \mathcal{D}', \mathcal{C}')$ with $\mathcal{X}' \supseteq \mathcal{X}$ (i.e. with possibly more variables) but with **binary constraints**

Equivalent means:

– any solution for P can be extended into a solution for P'

– any solution for P' corresponds to a solution for P (i.e. if ν is a solution for P', then its projection $\nu|_{\mathcal{X}}$ over $\mathcal{X}$ is a solution for P)

Mapping CSPs to binary CSPs

For $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ with n vars and m constraints, define $P' = (\mathcal{X}', \mathcal{D}', \mathcal{C}')$:

- $\mathcal{X}' = \{X'_i : X_i \in \mathcal{X}\} \cup \{X_{n+j} : 1 \leq j \leq m\}$ (one **new var** per constr.)
- Domains for original vars: $D'_i = D_i \cap \cap\{R_j : \text{scope}_j = (X_i)\}$, $i = 1 \ldots n$
- Domains for new vars: $D'_{n+j} = R_j$, $j = 1 \ldots m$ (var X'_{n+j} has domain given by tuples permitted by constraint C_j: $D'_{n+j} \subseteq \Pi_{X_i \in \text{scope}_j} D_i$)
- **Binary constraints:** for each (i, j) such that $X_i \in \text{scope}_j$, add constraint $C'_{i,j} = \langle \text{scope}'_{i,j}, R'_{i,j} \rangle$ where:
  – $\text{scope}'_{i,j} = (X'_i, X'_{n+j})$
  – $R'_{i,j} = \{(x_i, t) \in D'_i \times D'_{n+j} : t[X_i] = x_i\}$

Example: Mapping CSP to binary CSP

Problem P with variables $\mathcal{X} = \{X_1, X_2, X_3\}$ over domain $D = \{0, 1, 2\}$ and two constraints: $X_3 = X_1 + X_2 \mod 3$, and $X_2 + X_3 \geq 1$

Transformed problem is $P' = (\mathcal{X}' = \{X'_i : 1 \leq i \leq 5\}, \mathcal{D}', \mathcal{C}')$ where

– $D'_i = D$ for $i = 1, 2, 3$

– $D'_4 = \{(x_1, x_2, x_3) \in D^3 : x_3 = x_1 + x_2 \mod 3\}$

– $D'_5 = \{(x_2, x_3) \in D^2 : x_2 + x_3 \geq 1\}$

– $C'_{14} = \langle (X'_1, X'_4), R'_{14} \rangle$ with $R'_{14} = \{(x_1, (x_1, x_2, x_3)) : x_1 \in D, (x_1, x_2, x_3) \in D'_4\}$

– $C'_{24} = \langle (X'_2, X'_4), R'_{24} \rangle$ with $R'_{24} = \{(x_2, (x_1, x_2, x_3)) : x_2 \in D, (x_1, x_2, x_3) \in D'_4\}$

– $C'_{34} = \langle (X'_3, X'_4), R'_{34} \rangle$ with $R'_{34} = \{(x_3, (x_1, x_2, x_3)) : x_3 \in D, (x_1, x_2, x_3) \in D'_4\}$

– $C'_{25} = \langle (X'_2, X'_5), R'_{25} \rangle$ with $R'_{25} = \{(x_2, (x_2, x_3)) : x_2 \in D, (x_2, x_3) \in D'_5\}$

– $C'_{35} = \langle (X'_3, X'_5), R'_{35} \rangle$ with $R'_{35} = \{(x_3, (x_2, x_3)) : x_3 \in D, (x_2, x_3) \in D'_5\}$

Example: Sudoku

	1	2	3	4	5	6	7	8	9
A			3		2		6		
B	9			3		5			1
C			1	8		6	4		
D			8	1		2	9		
E	7								8
F			6	7		8	2		
G			2	6		9	5		
H	8			2		3			9
I			5		1		3		

	1	2	3	4	5	6	7	8	9
A	4	8	3	9	2	1	6	5	7
B	9	6	7	3	4	5	8	2	1
C	2	5	1	8	7	6	4	9	3
D	5	4	8	1	3	2	9	7	6
E	7	2	9	5	6	4	1	3	8
F	1	3	6	7	9	8	2	4	5
G	3	7	2	6	8	9	5	1	4
H	8	1	4	2	5	3	7	6	9
I	6	9	5	4	1	7	3	8	2

[Image from Russell & Norvig. *Artificial Intelligence: A Modern Approach*]

Naive backtracking algorithm

For given node n, children of n correspond to different **extensions with one variable** of the assignment associated to n

```
naive-backtrack(assignment A, csp P)
    if A is complete assignment then return A

    foreach variable X unassigned by A
        foreach value in domain of X
            if X = value is consistent with A wrt P
                A' := A union { X = value }
                result := naive-backtrack(A', P)
                if result != FAIL then return result

    return FAIL
```

For n variables and $d = \max_i |D_i|$, branching factor at root is $O(nd)$, at second level $O((n-1)d)$, etc. Total number of leaves is $O(n!d^n)$, yet number of assignments is only $O(d^n)$

Basic backtracking algorithm

For given node n, children of n correspond to the different values for **fixed unassigned variable**

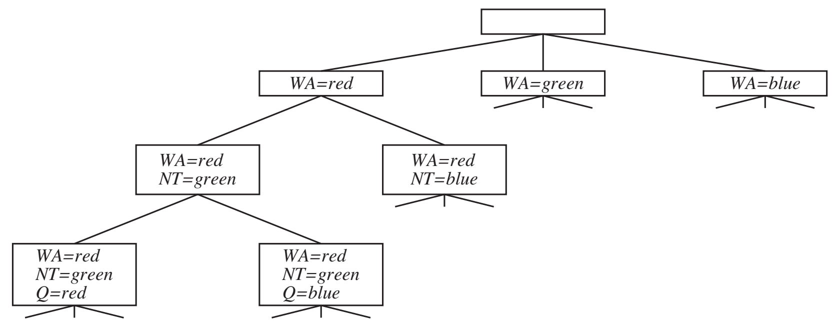
```
backtrack(assignment A, csp P)
    if A is complete assignment then return A

    X := select-unassigned-variable(A, P)
    foreach value in domain of X
        if X = value is consistent with A wrt P
            A' := A union { X = value }
            result := backtrack(A', P)
            if result != FAIL then return result

    return FAIL
```

Branching factor is $O(d)$, where $d = \max_i |D_i|$. With n variables, number of leaves is $O(d^n)$ and equal to number of assignments

Example: Backtracking



[Image from Russell & Norvig. *Artificial Intelligence: A Modern Approach*]

Critical issues when implementing solution

- Which variable should be chosen at each node? How should its values be ordered for the recursion?
- What are the implications of current assignment for still unassigned variables?
- When branch fails, can the search avoid repeating the failure in next branches?

Variable ordering

Idea is to choose the **most constrained variable** in order to detect a failure (backtrack) as soon as possible

It is better to fail high on a branch than deep. Heuristic is called **MRV (Minimum Remaining Values)**, Most Constrained Variable, or "fail-first"

Another idea is to choose variable involved in most constraints. It can be combined with MRV as a **tie-breaker**:

If two variables have the same number of remaining values (MRV criterion), prefer the one involved in more constraints

Value ordering

Once a variable is selected, its values must be ordered

Least-constraining value is an effective heuristic:

Prefer values that **rule out fewest values** *for neighbor variables in constraint graph*

Motivation is that once a variable is fixed, the algorithm should try to find a solution as fast as possible

Combining search with inference

We can solve a CSP by either:

– perform **pure search** with the backtracking algorithm

– perform **pure inference** (as shown later)

Both methods are correct but do not scale up to big problems

State-of-the-art solvers combine search and **limited but efficient** forms of inference in order to reduce the search space

Forward checking

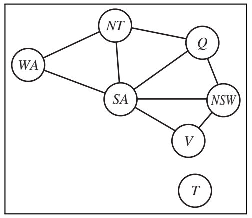
Each node in search tree keeps **current domains** for unassigned variables

Whenever variable X_i is assigned, FC looks at each unassigned variable X_j that is connected to X_i by a constraint, and deletes from D_j all values that are **inconsistent** with value chosen for X_i

Partner of MRV heuristic: select the next variable to assign as one with smallest current domain

Example: Backtracking with forward checking

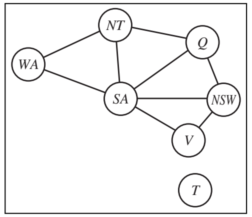
Variable selection with **MRV heuristic**



	WA	NT	Q	NSW	V	SA	T
Initial domains	RGB	RGB	RGB	RGB	RGB	RGB	RGB
After $SA =$ R	GB	GB	GB	GB	GB	R	RGB
After $NT =$ G	B	G	B	GB	GB	R	RGB
After $Q =$ B	B	G	B	G	GB	R	RGB
After $NSW =$ G	B	G	B	G	B	R	RGB
After $WA =$ B	B	G	B	G	B	R	RGB
After $V =$ B	B	G	B	G	B	R	RGB
After $T =$ R	B	G	B	G	B	R	R

Example: Backtracking with forward checking

Variable/value selection: $WA =$ R, $Q =$ G, ...

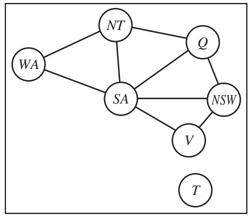


	WA	NT	Q	NSW	V	SA	T
Initial domains	RGB	RGB	RGB	RGB	RGB	RGB	RGB
After $WA =$ R	R	GB	RGB	RGB	RGB	GB	RGB
After $Q =$ G	R	B	G	R B	RGB	B	RGB
After $V =$ B	R	B	G	R	B	—	RGB

***** BACKTRACK *****

Example: Backtracking with forward checking

Variable/value selection: $WA =$ R, $Q =$ G, ...

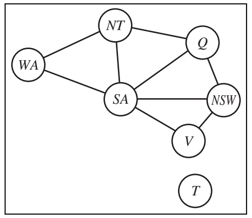


	WA	NT	Q	NSW	V	SA	T
Initial domains	RGB	RGB	RGB	RGB	RGB	RGB	RGB
After $WA =$ R	R	GB	RGB	RGB	RGB	GB	RGB
After $Q =$ G	R	B	G	R B	RGB	B	RGB
After $V =$ R	R	B	G	B	R	B	RGB
After $NT =$ B	R	B	G	B	R	—	RGB

***** BACKTRACK *****

Example: Backtracking with forward checking

Variable/value selection: $WA = \text{R}$, $Q = \text{G}$, ...

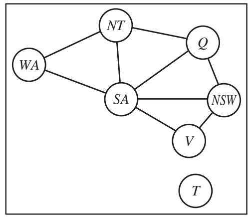


	WA	NT	Q	NSW	V	SA	T
Initial domains	RGB	RGB	RGB	RGB	RGB	RGB	RGB
After $WA = \text{R}$	R	GB	RGB	RGB	RGB	GB	RGB
After $Q = \text{G}$	R	B	G	R B	RGB	B	RGB
After $V = \text{G}$	R	B	G	B	G	B	RGB
After $T = \text{R}$	R	B	G	B	G	B	R
After $NT = \text{B}$	R	B	G	B	G	—	R

***** BACKTRACK *****

Example: Backtracking with forward checking

Variable/value selection: $WA = \text{R}$, $Q = \text{G}$, ...

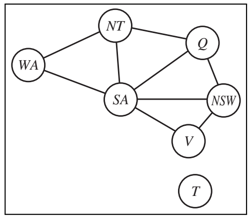


	WA	NT	Q	NSW	V	SA	T
Initial domains	RGB	RGB	RGB	RGB	RGB	RGB	RGB
After $WA = \text{R}$	R	GB	RGB	RGB	RGB	GB	RGB
After $Q = \text{G}$	R	B	G	R B	RGB	B	RGB
After $V = \text{G}$	R	B	G	B	G	B	RGB
After $T = \text{G}$	R	B	G	B	G	B	G
After $NT = \text{B}$	R	B	G	B	G	—	G

***** BACKTRACK *****

Example: Backtracking with forward checking

Variable/value selection: $WA = \text{R}$, $Q = \text{G}$, ...



	WA	NT	Q	NSW	V	SA	T
Initial domains	RGB	RGB	RGB	RGB	RGB	RGB	RGB
After $WA = \text{R}$	R	GB	RGB	RGB	RGB	GB	RGB
After $Q = \text{G}$	R	B	G	R B	RGB	B	RGB
After $V = \text{G}$	R	B	G	B	G	B	RGB
After $T = \text{B}$	R	B	G	B	G	B	B
After $NT = \text{B}$	R	B	G	B	G	—	B

***** BACKTRACK *****

Chronological and non-chronological backtracking

When search reaches **terminal node** that doesn't correspond to complete assignment (i.e. **conflict node**), the search **backtracks** to **most recent decision point**

Most recent decision point may not be **reason for conflict**

A better idea is to **analyze the conflict** and backtrack to most recent decision point that caused the conflict

Such backtracking is called **non-chronological conflict-based backtracking** and also **conflict-directed backjumping**

Constraint propagation: Arc consistency

Arc consistency is a **property of CSPs:**

- CSP $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ is **arc consistent** iff for each pair of variables X_i and X_j connected in constraint graph, the **arc** (X_i, X_j) is **consistent in** P

- Arc (X_i, X_j) is consistent in P iff for each value x_i of X_i, there exists a value x_j of X_j such that the partial assignment $(X_i = x_i, X_j = x_j)$ is consistent with all constraints (i.e. it doesn't violate any constraint)

For each satisfiable CSP P, there is a CSP P' equivalent to P and with the same variables as P that is arc consistent

An algorithm for arc consistency transforms P into equivalent P' or detects that P has no solution. There are many such algorithms

Arc consistency: AC3

```
bool AC3(csp P)
    Queue Q
    Insert in Q all arcs (X,Y) in constraint graph
    while Q is not empty
        Let (X,Y) := Q.pop()
        if reduce-arc(X,Y)
            if Domain[X] == ∅ then return false
            foreach Z such that (Z,X) is edge in constraint graph
                Insert arc (Z,X) in Q
    return true

bool reduce-arc(variable X, variable Y)
    removed := false
    foreach x in Domain[X]
        found := false
        foreach y in Domain[Y]
            if (X=x,Y=y) satisfies all constraints between X and Y
                found := true
                break
        if not found
            Remove x from Domain[X]
            removed := true
    return removed
```


Analysis of AC3

Consider CSP $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ with n variables, and let $d = \max_i |D_i|$:

- Time for reduce-arc(X,Y) is $O(d^2)$ assuming that takes constant time to check whether partial assignment $(X = x, Y = y)$ is consistent with all constraints

- There are $O(n^2)$ initial insertions in the queue

- Arc (Z, X) is re-inserted when a value of X is removed. Since there are $O(d)$ values for X, arc (Z, X) is re-inserted $O(d)$ times

- Number of iterations bounded by $O(n^2 + n^2 d) = O(n^2 d)$

- **Total time is** $O(n^2 d^3)$

Combining search with AC3

Two ways of combining search with AC3:

– Before search starts: make CSP arc-consistent and then do search

– During search: enforce arc consistency at each node during search (known as **Maintaining Arc Consistency or MAC**)

First option is enough in easy problems while the second is necessary for difficult ones

AC4: Keep track of supports

Algorithm for arc consistency that runs in time $O(n^2d^2)$ which is **optimal** since lower bound $\Omega(n^2d^2)$ holds

Idea:

– Keep counters $n(i,x,j)$ for each constraint with scope $\{X_i, X_j\}$ and value $x \in D_i$ that stores **number of values** of X_j that are consistent with $X_i = x$

– Use queue to track values $X = x$ have **lost support**

– Revise counters efficiently

Arc consistency: AC4

```
bool AC4(csp P)
    Queue Q

    % initialization
    Calculate value of counters n(X,x,Y). If n(X,x,Y) = 0,
    remove x from Domain[X] and enqueue pair (X,x)

    while Q is not empty
        Let (X,x) := Q.pop()

        if Domain[X] is empty then
            return false                         % CSP has no solution

        % value x was removed from Domain[X]
        foreach (Z,X)
            foreach z in Domain[Z]
                if (Z=z,X=x) is consistent then
                    Decrement counter n(Z,z,X)
                    if n(Z,z,X) == 0 then
                        Remove z from Domain[Z]
                        Enqueue (Z,z) in Q
    return true
```


Analysis of AC4

Consider CSP $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ with n variables, and let $d = \max_i |D_i|$:

- Time for initialization is $O(n^2d^2)$
- Time of inner loop is $O(nd)$
- Pair (X,x) is added to queue when value x is removed from D_X. Maximum number of pairs in Q is thus $O(nd)$
- **Total time is $O(n^2d^2 + n^2d^2) = O(n^2d^2)$**

Inference for CSPs

We show how to solve CSPs using **pure inference**

Along the way, we identify **tractable subclasses** of CSPs that are solved in polynomial time

High-order consistency

Arc consistency can be generalized to k-consistency

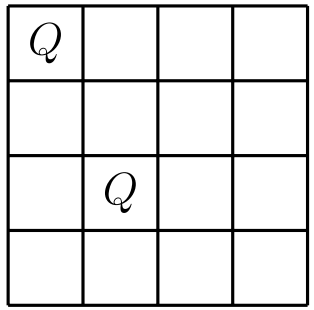
CSP P is **k-consistent** iff for any set of $k-1$ variables and each consistent assignment for them, the assignment can be **consistently extended** over any other variable

Under this definition:

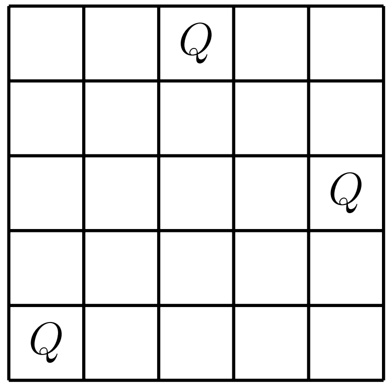
- P is 1-consistent iff for each variable X and each unary constraint C for X, there is value x for X that satisfies C
- P is 2-consistent iff P is arc consistent
- ...

P is **strongly k-consistent** iff it is i-consistent for $i = 1, 2, \ldots, k$

Examples: Queens



Arc-consistent but not 3-consistent



2- and 3-consistent but not 4-consistent

Establishing k-consistency (naive algorithm)

```
bool k-consistency(csp P)
    change := true
    while change
        change := false
        foreach subset S of k-1 variables
            foreach variable X not in S
                change := change || k-revise(S,X)

    if domain of some variable is empty then
        return false
    else
        return true

bool k-revise(S,X)
    change := false
    foreach consistent valuation v of S
        if there is no value x for X such that {v,X=x} is consistent
            Mark valuation v as forbidden
            change := true
    return change
```


Remarks on establishing k-consistency

If there is no constraint in which forbidden (partial) valuation ν can be eliminated, algorithm is discovering an **implied constraint**

If CSP has only binary constraints, after establishing k-consistency new constraints of order $k-1$ may appear

Forbidden valuations (also called **no-goods**) are recorded in existing constraints or stored in memory

Establishing k-consistency takes time $O((2nd)^{2k})$ where n is number of variables and d is maximum cardinality of domains

i-consistency does not imply j-consistency for $j < k$

Solving CSPs by pure inference

Let $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ be a CSP with n variables that is **strongly n-consistent**

The following **backtrack-free** algorithm finds a solution for P or determines P has no solution

1. Let $X_1, X_2, \ldots, X_n$ be order for variables (any order will do), and let ν be **empty partial assignment**
2. If domain of X_1 is empty, return **FAILURE**
3. For $i = 1, 2, \ldots, n$:
   - **Select value** x_i for X_i that is **consistent** with partial valuation ν
   - **Extend** partial valuation ν with $X_i = x_i$
4. Return valuation ν

Correctness of inference algorithm

We show that a value x_i for X_i that is consistent with the current valuation ν can be found for $i = 1, 2, \ldots, n$ (step 4):

- Claim is true for first iteration as ν is the empty valuation, the problem is 1-consistent, and $D_1 \neq \emptyset$

- Consider the $(i+1)$th iteration and let ν be current partial valuation at beginning of $(i+1)$th iteration. By induction, ν is **consistent**

  By strong n-consistency, problem is $(i+1)$-consistent. Therefore, any consistent valuation for $\{X_1, \ldots, X_i\}$, like ν, can be extended into consistent valuation for any other variable, like X_{i+1}

  Then, there is a value x_{i+1} for X_{i+1} that is consistent with ν and the valuation can be extended with $X_{i+1} = x_{i+1}$

At the end ν is a **complete and consistent** assignment; i.e. ν is a solution

Strong consistency and existence of solutions

Let $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ be a CSP with n variables

If P is strongly n-consistent and domain of some variable is non-empty, then P has solution

Tree structure

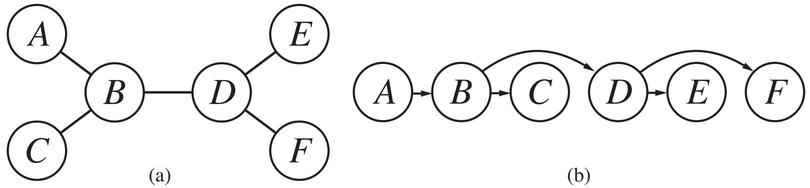
If constraint graph is tree, CSP can be solved in $O(nd^3)$ time

1. Designate any vertex in constraint graph as root and order the vertices (variables) **topologically** so that each vertex appears in the order **after its parent** (it can be done since graph is tree)
2. Enforce **strong arc consistency** in $O(nd^3)$ time (trees have $O(n)$ edges)
3. If domain of first variable is empty, return **FAILURE**
4. Assign values from first to last variable in the order in **backtrack-free** manner as before:

   X_1 can be assigned because the problem is 1-consistent and $D_1 \neq \emptyset$

   At stage $i+1$ for X_{i+1}, variable X_{i+1} has only one parent X_j with $j < i$ (as the graph is tree). Since problem is 2-consistent, current assignment can be consistently extended with $X_{i+1} = x_{i+1}$ for some $x_{i+1} \in D_{i+1}$

Topological sort of a tree



(a) (b)

[Image from Russell & Norvig. *Artificial Intelligence: A Modern Approach*]

Improved algorithm for tree structure

We can improve algorithm by using **directed arc consistency**

CSP is **directed arc consistent** for order $(X_1, X_2, \ldots, X_n)$ iff every arc (X_i, X_j) in constraint graph, for $i < j$, is consistent

1. Topologically order variables as before as $(X_1, X_2, \ldots, X_n)$
2. (Make problem directed arc consistent.) For $j = n$ to 2:
   - Call `reduce-arc(parent(X[j]), X[j])` to make arc $(\text{parent}(X_j), X_j)$ consistent
   - If domain of parent(X_j) is empty, return **FAILURE**
3. (Construct valuation in backtrack-free manner.) For $i = 1$ to n:
   - Select value x_i for X_i that is **consistent** with assignment of $parent(X_i)$. This can be done because X_i has unique parent X_j, with $j < i$, and the directed arc consistency established in step 2

Analysis: each of the $O(n)$ calls to `reduce-arc()` takes time $O(d^2)$. The other steps are done in linear time. **Total time is thus** $O(nd^2)$

Directional consistency

Strong n-consistency is more than needed as variables are assigned values along **fixed variable ordering**

Like improved algorithm for trees, we can enforce appropriate level of consistency along fixed ordering

Width of graph

Let $G = (V, E)$ be undirected graph and $\prec$ be order relation on V:

- $\prec$-width of vertex v in V: #edges from v to $\prec$-smaller vertices
- $\prec$-width of G: maximum $\prec$-width of vertex in G
- width of G: minimum $\prec$-width of G over all possible orderings $\prec$

Width of CSP P is width of its constraint graph

Improved inference algorithm

Let $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ be CSP with constraint graph G

If P is strongly k-consistent, P has width $< k$, and all domains are non-empty, then P has solution

1. Let $\prec$ be ordering on $\mathcal{X}$ such that G has $\prec$-width $\leq k - 1$
2. Let $X_1, X_2, \ldots, X_n$ be $\prec$-ordering for variables and ν be empty valuation
3. If domain of X_1 is empty, return **FAILURE**
4. For $i = 1, 2, \ldots, n$:
   - **Select value** x_i for X_i that is **consistent** with partial valuation ν
   - **Extend** partial valuation ν with $X_i = x_i$
5. Return valuation ν

Remarks for improved inference algorithm

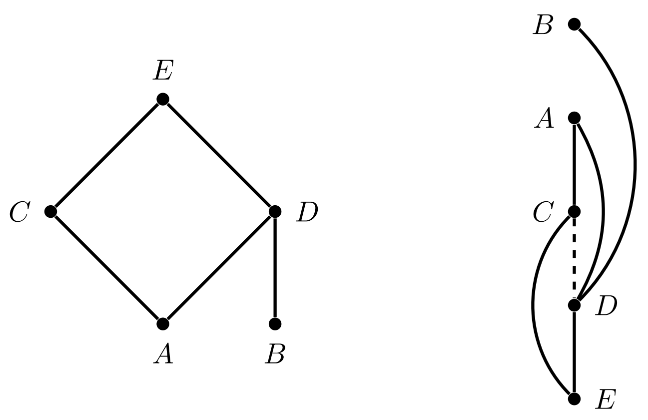
Requires strong k-consistency instead of strong n-consistency ($k \leq n$)

Enforcing strong k-consistency on CSP P may **increase width** of P

We want:

- Selecting variable ordering **dynamically**
- Adjust consistency of each node in **adaptive way**
- Handle width increments in sound manner

Example: Adaptive consistency



Ordering (E, D, C, A, B)

Dechter and Pearl's adaptive consistency

Let $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ be CSP and $(X_1, \ldots, X_n)$ be ordering of $\mathcal{X}$

1. For $i = n, \ldots, 1$ do steps (2)–(5)
2. If domain X_i is empty, return **FAILURE**
3. Compute $Parents(X_i) = \{X_j : j < i \text{ and } X_j \text{ is connected to } X_i\}$
4. Add edges between all pairs of variables in $Parents(X_i)$ (among those not already connected)
5. Perform consistency$(Parents(X_i), X_i)$
6. Find solution (or determine none exists) in **backtrack-free** manner along order $(X_1, \ldots, X_n)$

Ordering doesn't need to fixed a priori, a good ordering can be **discovered** along execution; obtaining best ordering is **NP-hard**

Other approaches

- "Remove" variables until constraint graph becomes tree that can be solved by algorithm for trees. This is called **cutset conditioning**

- Construct a **tree decomposition** of CSP made of independent subproblems, solve each subproblem independently, and combine solutions into global solution

Cutset conditioning

1. **Choose** set S of variables such that after their removal, the constraint graph becomes a tree. S is called **cycle cutset** of constraint graph

2. For each valuation $\nu = \nu_S$ of S, **reduce** P into P_ν by **instantiating** variables in S to values in ν

3. **Solve** P_ν and return overall solution if found

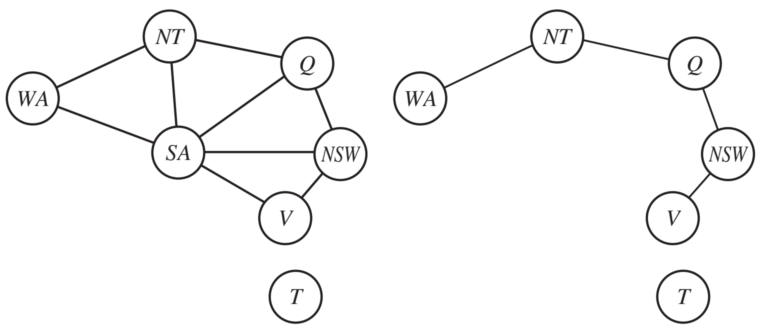
4. If there is no valuation $\nu = \nu_S$ such that P_ν is solvable, return FAILURE

5. If $|S| = c$, reduced CSP can be solved in time $O((n-c)d^2)$ using **directed arc consistency**. Since there are $O(d^c)$ valuations for S, overall algorithm takes time $O((n-c)d^{2+c})$

There is no a priori bound on the size S of a minimum cycle cutset

Finding cycle cutset of **minimum size** is **NP-hard**

Cycle cutset in example



[Image from Russell & Norvig. *Artificial Intelligence: A Modern Approach*]

Tree decomposition

Tree decomposition of CSP $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ is collection of subproblems where each subproblem, defined over subset of variables, is such that:

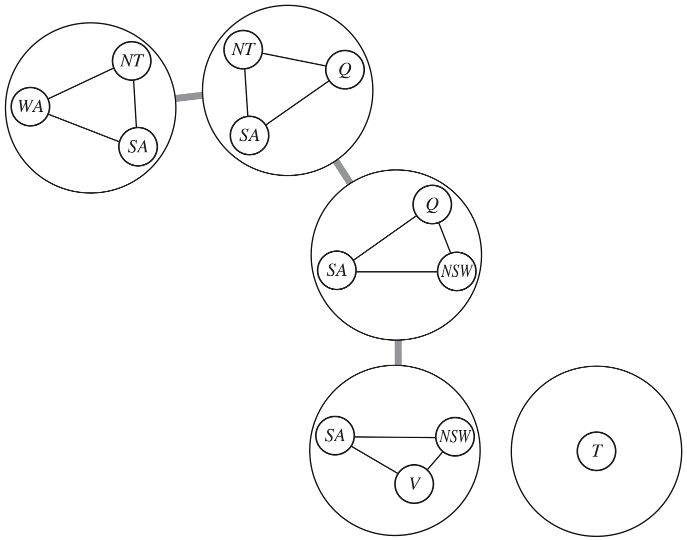
– Each variable appears in at least one subproblem

– Each constraint $C \in \mathcal{C}$, there is at least one subproblem whose set of variables contains the scope of C

– Subproblems sharing variables are organized into **tree structure**

– If variable X_i appears in two subproblems, X_i then appears in each subproblem along the unique path that connects both subproblems

Tree decomposition of example



[Image from Russell & Norvig. *Artificial Intelligence: A Modern Approach*]

Solving CSPs by tree decompositions

Given CSP $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ and tree decomposition for P, construct new **binary CSP** $P' = (\mathcal{X}', \mathcal{D}', \mathcal{C}')$ as follows:

– There is **one variable for each subproblem** in tree decomposition; the ith subproblem corresponds to variable X'_i

– Domain D'_i for variable X'_i corresponds to **all solutions of the ith subproblem** (ith subproblem is viewed as a reduced CSP)

– If ith and jth subproblems are connected (because they share at least one variable), there is **binary constraint** in $\mathcal{D}'$ with scope (X'_i, X'_j) and relation given by all tuples (t'_i, t'_j) such that

  – $t'_i \in D'_i$ and $t'_j \in D'_j$

  – $t'_i[X_k] = t'_j[X_k]$ for every varaible X_k that appears in both subproblems (i.e. solutions to subproblems must agree on shared variables)

Analysis

Let $P = (\mathcal{X}, \mathcal{D}, \mathcal{C})$ be CSP, T be tree decomposition for P with k subproblems, and c be **maximum subproblem size**

– Constructing P' takes time $O(kd^c)$ as there are k subproblems and each subproblem involves $O(d^c)$ valuations over its variables

– Problem P' has k variables, each domain has size $O(d^c)$, and P' has tree structure

– P' can be solved by **directed arc consistency** in time $O(kd^{2c})$

– Total time is thus $O(kd^{2c})$

There is no a priori bound on the maximum subproblem size

Finding **best tree decomposition** is **NP-hard**

Consistency and relational databases

```
bool reduce-arc(variable X, variable Y)
    removed := false
    foreach x in Domain[X]
        found := false
        foreach y in Domain[Y]
            if (X=x,Y=y) satisfies all constraints between X and Y
                found := true
                break
        if not found
            Remove x from Domain[X]
            removed := true
    return removed
```

If R_{XY} expresses all constraints between X and Y, `reduce-arc(X,Y)` is equivalent to

$$D_X \;:=\; D_X \cap \pi_X(R_{XY} \bowtie D_Y)$$

where π_X is **projection** on X, and $\bowtie$ is **relational join**

Generalizing arc consistency

For non-binary constraints, arc consistency may be too weak

Example: for $X_1, X_2, X_3 \geq 0$, $X_3 \geq 13$, and $X_1 + X_2 + X_3 \leq 15$, arc consistency is not able to infer $X_1 \leq 2$ and $X_2 \leq 2$

AC: $D_X := D_X \cap \pi_X(R_{XY} \bowtie D_Y)$

Generalized AC: $D_X := D_X \cap \pi_X(R_S \bowtie D_{S\setminus\{X\}})$

Relational AC: $R_{S\setminus\{X\}} := R_{S\setminus\{X\}} \cap \pi_{S\setminus\{X\}}(R_S \bowtie D_X)$

where R_S is constraint such that $X \in S$

Example: Generalized AC infers $X_1 \leq 2$ and $X_2 \leq 2$, while Relational AC infers $X_1 + X_2 \leq 2$

Summary

- CSP is a fundamental problem in AI
- CSPs with binary constraints are universal
- CSPs are intractable in general
- CSPs can be solved by either pure search or pure inference
- Solving CSPs backtrack free after enforcing consistency
- Consistency and relational databases, and generalizations of AC
- **State-of-the-art solvers** = search + limited/efficient inference