Constraint propagation
backtracking-based search

Roman Barták
Charles University, Prague (CZ)
roman.bartak@mff.cuni.cz
http://ktiml.mff.cuni.cz/~bartak

"Constraint programming represents one of the closest approaches computer science has yet made to the Holy Grail of programming: the user states the problem, the computer solves it."
Eugene C. Freuder, Constraints, April 1997

Holly Grail of Programming

> Computer, solve the SEND, MORE, MONEY problem!
> Here you are. The solution is
> \[9,5,6,7\] + \[1,0,8,5\] = \[1,0,6,5,2\]

a Star Trek view

> Sol = \[9,5,6,7,1,0,8,2\]

today reality

Tutorial outline

- Introduction
  - history, applications, a constraint satisfaction problem
- Depth-first search
  - backtracking, backjumping, backmarking
  - incomplete and discrepancy search
- Consistency
  - node, arc, and path consistencies
  - \(k\)-consistency and global constraints
- Combining search and consistency
  - look-back and look-ahead schemes
  - variable and value ordering
- Conclusions
  - constraint solvers, resources

A bit of history

- Scene labelling (Waltz 1975)
  - feasible interpretation of 3D lines in a 2D drawing
- Interactive graphics (Sutherland 1963, Borning 1981)
  - geometrical objects described using constraints
- Logic programming (Gallaire 1985, Jaffar, Lassez 1987)
  - from unification to constraint satisfaction
**Application areas**

- Molecular biology
  - DNA sequencing
  - Determining protein structures
- Interactive graphic
  - Web layout
- Network configuration
- Assignment problems
  - Personal assignment
  - Stand allocation
- Timetabling
- Scheduling
- Planning

**Constraint technology**

Based on declarative problem description via:
- Variables with domains (sets of possible values)
  - E.g. start of activity with time windows
- Constraints restricting combinations of variables
  - E.g. endA < startB

Constraint optimization via objective function
- E.g. Minimize makespan

Why to use constraint technology?
- Understandable
- Open and extendible
- Proof of concept

---

**Constraint satisfaction problem**

Consists of:
- A finite set of variables
- A finite set of domains - a finite set of values for each variable
- A finite set of constraints
  - Constraint is an arbitrary relation over the set of variables
  - Can be defined extensionally (a set of compatible tuples) or intentionally (formula)

A solution to a CSP is a complete consistent assignment of variables.
- Complete = a value is assigned to every variable
- Consistent = all the constraints are satisfied

---

**Two or more?**

- Binary constraint satisfaction
  - Only binary constraints
  - Any CSP is convertible to a binary CSP
    - Dual encoding (Stergiou & Walsh, 1990)
      - Swapping the role of variables and constraints

- Boolean constraint satisfaction
  - Only Boolean (two valued) domains
  - Any CSP is convertible to a Boolean CSP
    - SAT encoding
      - Boolean variable indicates whether a given value is assigned to the variable
We are looking for a complete consistent assignment.
- start with a consistent assignment (for example, empty one)
- extend the assignment towards a complete assignment

Depth-first search is a technique of searching solution by extending a partial consistent assignment towards a complete consistent assignment.
- assign values gradually to variables
- after each assignment test consistency of the constraints over the assigned variables
- and backtrack upon failure

Backtracking is probably the most widely used complete systematic search algorithm.
- complete = guarantees finding a solution or proving its non-existence

Note: Backtracking

Backtracking is the most widespread complete systematic search algorithm.
- On failure, go back and set the previous variable to a different value.
- The search continues until a solution is found or the search space is exhausted.

Solution: backjumping (jump to the source of the failure)

Example:
A,B,C,D,E::1..10, A=3*E
- backjumping (jump to the source of the failure)
- BT tries all the assignments for B,C,D before finding that A>2

Consistency procedure checks satisfaction of constraints whose variables are already assigned.

We are looking for a complete consistent assignment.
- start with a consistent assignment (for example, empty one)
- extend the assignment towards a complete assignment

Depth-first search is a technique of searching solution by extending a partial consistent assignment towards a complete consistent assignment.
- assign values gradually to variables
- after each assignment test consistency of the constraints over the assigned variables
- and backtrack upon failure

Backtracking is probably the most widely used complete systematic search algorithm.
- complete = guarantees finding a solution or proving its non-existence

Note: Backtracking

Backtracking is the most widespread complete systematic search algorithm.
- On failure, go back and set the previous variable to a different value.
- The search continues until a solution is found or the search space is exhausted.

Solution: backjumping (jump to the source of the failure)

Example:
A,B,C,D,E::1..10, A=3*E
- backjumping (jump to the source of the failure)
- BT tries all the assignments for B,C,D before finding that A>2

Consistency procedure checks satisfaction of constraints whose variables are already assigned.

We are looking for a complete consistent assignment.
- start with a consistent assignment (for example, empty one)
- extend the assignment towards a complete assignment

Depth-first search is a technique of searching solution by extending a partial consistent assignment towards a complete consistent assignment.
- assign values gradually to variables
- after each assignment test consistency of the constraints over the assigned variables
- and backtrack upon failure

Backtracking is probably the most widely used complete systematic search algorithm.
- complete = guarantees finding a solution or proving its non-existence

Note: Backtracking

Backtracking is the most widespread complete systematic search algorithm.
- On failure, go back and set the previous variable to a different value.
- The search continues until a solution is found or the search space is exhausted.

Solution: backjumping (jump to the source of the failure)

Example:
A,B,C,D,E::1..10, A=3*E
- backjumping (jump to the source of the failure)
- BT tries all the assignments for B,C,D before finding that A>2

Consistency procedure checks satisfaction of constraints whose variables are already assigned.

We are looking for a complete consistent assignment.
- start with a consistent assignment (for example, empty one)
- extend the assignment towards a complete assignment

Depth-first search is a technique of searching solution by extending a partial consistent assignment towards a complete consistent assignment.
- assign values gradually to variables
- after each assignment test consistency of the constraints over the assigned variables
- and backtrack upon failure

Backtracking is probably the most widely used complete systematic search algorithm.
- complete = guarantees finding a solution or proving its non-existence

Note: Backtracking

Backtracking is the most widespread complete systematic search algorithm.
- On failure, go back and set the previous variable to a different value.
- The search continues until a solution is found or the search space is exhausted.

Solution: backjumping (jump to the source of the failure)

Example:
A,B,C,D,E::1..10, A=3*E
- backjumping (jump to the source of the failure)
- BT tries all the assignments for B,C,D before finding that A>2

Consistency procedure checks satisfaction of constraints whose variables are already assigned.

We are looking for a complete consistent assignment.
- start with a consistent assignment (for example, empty one)
- extend the assignment towards a complete assignment

Depth-first search is a technique of searching solution by extending a partial consistent assignment towards a complete consistent assignment.
- assign values gradually to variables
- after each assignment test consistency of the constraints over the assigned variables
- and backtrack upon failure

Backtracking is probably the most widely used complete systematic search algorithm.
- complete = guarantees finding a solution or proving its non-existence

Note: Backtracking

Backtracking is the most widespread complete systematic search algorithm.
- On failure, go back and set the previous variable to a different value.
- The search continues until a solution is found or the search space is exhausted.

Solution: backjumping (jump to the source of the failure)

Example:
A,B,C,D,E::1..10, A=3*E
- backjumping (jump to the source of the failure)
- BT tries all the assignments for B,C,D before finding that A>2

Consistency procedure checks satisfaction of constraints whose variables are already assigned.

We are looking for a complete consistent assignment.
- start with a consistent assignment (for example, empty one)
- extend the assignment towards a complete assignment

Depth-first search is a technique of searching solution by extending a partial consistent assignment towards a complete consistent assignment.
- assign values gradually to variables
- after each assignment test consistency of the constraints over the assigned variables
- and backtrack upon failure

Backtracking is probably the most widely used complete systematic search algorithm.
- complete = guarantees finding a solution or proving its non-existence

Note: Backtracking

Backtracking is the most widespread complete systematic search algorithm.
- On failure, go back and set the previous variable to a different value.
- The search continues until a solution is found or the search space is exhausted.

Solution: backjumping (jump to the source of the failure)

Example:
A,B,C,D,E::1..10, A=3*E
- backjumping (jump to the source of the failure)
- BT tries all the assignments for B,C,D before finding that A>2

Consistency procedure checks satisfaction of constraints whose variables are already assigned.
During the second attempt to label C superfluous work is done - it is enough to leave the original value 2, the change of B does not influence C.

\[ \text{Algorithm backjumping} \]

\[ \text{procedure BJ(Unlabeled, Labeled, Constraints, PreviousLevel)} \]
\[ \text{if Unlabeled = {} then return Labeled} \]
\[ \text{pick first X from Unlabeled} \]
\[ \text{Level = PreviousLevel+1} \]
\[ \text{Jump = 0} \]
\[ \text{for each value V from DX do} \]
\[ \text{C = consistent(X/V/Level) \cup Labeled, Constraints, Level} \]
\[ \text{if C is not satisfied by Labeled then} \]
\[ \text{J = \max\{}{} X in C & X/V/L in Labeled & L<Level\} \]
\[ \text{Jump = PreviousLevel} \]
\[ \text{R = BJ(Unlabeled-(X), (X)/Level) \cup Labeled, Constraints, Level} \]
\[ \text{if R = fail/Level then return R \% success or backjump} \]
\[ \text{end if} \]
\[ \text{end for} \]
\[ \text{return fail(Jump) \% jump to the conflicting variable} \]
\[ \text{end BJ} \]

During the second attempt to label C superfluous work is done - it is enough to leave the original value 2, the change of B does not influence C.

\[ \text{Algorithm dynamic BT} \]

\[ \text{procedure (Variables, Constraints)} \]
\[ \text{Labeled = {}; Unlabeled = Variables} \]
\[ \text{while Unlabeled \neq \{\} do} \]
\[ \text{select X in Unlabeled} \]
\[ \text{ValuesX = \{\} - \{\text{values inconsistent with Labeled using Constraints}\}} \]
\[ \text{if ValuesX = {} then} \]
\[ \text{let E be an explanation of the conflict (set of conflicting variables)} \]
\[ \text{if E = {} then failure} \]
\[ \text{else let Y be the most recent variable in E and unassign Y (from Labeled) with eliminating explanation E-Y} \]
\[ \text{remove all the explanations involving Y} \]
\[ \text{end if} \]
\[ \text{else} \]
\[ \text{select V in ValuesX} \]
\[ \text{Unlabeled = Unlabeled-(X)} \]
\[ \text{Labeled = Labeled\cup(X/V)} \]
\[ \text{end if} \]
\[ \text{end while} \]
\[ \text{return Labeled} \]

\[ \text{Redundant work} \]

\[ \text{What is redundant work in chronological backtracking?} \]
\[ \text{\quad \square \ repeated computation whose result has already been obtained} \]

\[ \text{Example:} \]
\[ A,B,C,D :: 1..10, A+8<C, B=5*D \]

\[ \text{Redundant computations - it is not necessary to repeat them because the change of B does not influence C.} \]
Backmarking

- Removes redundant constraint checks by memorizing negative and positive tests:
  - Mark(X,V) is the farthest (instantiated) variable in conflict with the assignment X=V
  - BackTo(X) is the farthest variable to which we backtracked since the last attempt to instantiate X
- Now, some constraint checks can be omitted:
  \[ \text{Mark} < \text{BackTo} \geq \text{BackTo} \]

**Mark(Y,b)**
- **BackTo(Y)**
- **Y/b** is inconsistent with **X/a** (and consistent with all variables above X)
- **Y/b** must be checked with these variables
- **Y/b** is OK here
- **Y/b** must be checked with these variables

Consistency check (BM)

- Only the constraints where any value is changed are re-checked, and the farthest conflicting level is computed.

**Algorithm backmarking**

```plaintext
procedure BM(Unlabeled, Labeled, Constraints, Level)
if Unlabeled = {} then
  return Labeled
pick first X from Unlabeled % fix order of variables
for each value V from DX do
  if Mark(X,V) ≥ BackTo(X) then
    % re-check the value
    if consistent(X/V, Labeled, Constraints, Level) then
      R ← BM(Unlabeled-{X}, Labeled ∪ {X/V/Level}, Constraints, Level+1)
      if R ≠ fail then
        return R % solution found
    end if
  end if
end for
BackTo(X) ← Level-1 % jump will be to the previous variable
for each Y in Unlabeled do
  % tell everyone about the jump
  BackTo(Y) ← min {Level-1, BackTo(Y)}
end for
return fail % return to the previous variable
end BM
```

Incomplete search

- A cutoff limit to stop exploring a (sub-)tree
  - some branches are skipped → incomplete search
- When no solution found, restart with enlarged cutoff limit.

- **Bounded Backtrack Search** (Harvey, 1995)
- restricted number of backtracks
- **Depth-bounded Backtrack Search** (Cheadle et al., 2003)
  - restricted depth where alternatives are explored
- **Iterative Broadening** (Ginsberg and Harvey, 1990)
  - restricted breadth in each node
  - still exponential!
- **Credit Search** (Beldiceanu et al., 1997)
  - limited credit for exploring alternatives
  - credit is split among the alternatives
Heuristics in search

- Observation 1: The search space for real-life problems is so huge that it cannot be fully explored.
- Heuristics - a guide of search
  - they recommend a value for assignment
  - quite often lead to a solution
- What to do upon a failure of the heuristic?
  - BT cares about the end of search (a bottom part of the search tree) so it rather repairs later assignments than the earliest ones thus BT assumes that the heuristic guides it well in the top part
- Observation 2: The heuristics are less reliable in the earlier parts of the search tree (as search proceeds, more information is available).
- Observation 3: The number of heuristic violations is usually small.

Discrepancy
- the heuristic is not followed

Basic principles of discrepancy search:
- change the order of branches to be explored
  - prefer branches with less discrepancies
  - prefer branches with earlier discrepancies

Consistency
- Unary constraints are converted into variables' domains.
- Definition:
  - The vertex representing the variable $X$ is node consistent if every value in the variable's domain $D_x$ satisfies all the unary constraints imposed on the variable $X$.
  - CSP is node consistent if all the vertices are node consistent.
- Algorithm NC:
  ```plaintext
  procedure NC(G)
  for each variable $X$ in nodes(G) do
    if unary constraint on $X$ is inconsistent with $V$ then
      delete $V$ from $D_x$
    end if
  end for
  end NC
  ```
Arc consistency (AC)

Since now we will assume binary CSP only
i.e. a constraint corresponds to an arc (edge) in the constraint network.

Definition:
- The arc (Vi, Vj) is arc consistent iff for each value x from the domain Di there exists a value y in the domain Dj such that the assignment Vi = x and Vj = y satisfies all the binary constraints on Vi, Vj.
- Note: The concept of arc consistency is directional, i.e., arc consistency of (Vi, Vj) does not guarantee consistency of (Vj, Vi).
- CSP is arc consistent iff every arc (Vi, Vj) is arc consistent (in both directions).

Example:

\[
\begin{align*}
X & \in \{1, \ldots, 6\}, Y \in \{1, \ldots, 6\}, Z \in \{1, \ldots, 6\}, \ \ X < Y, \ Z < X - 2
\end{align*}
\]

How to establish arc consistency among the constraints?

Example:

Make all the constraints consistent until any domain is changed.

Algorithm AC-1

How to make (Vi, Vj) arc consistent?

- Delete all the values x from the domain Di that are inconsistent with all the values in Dj (there is no value y in Dj such that the assignment Vi = x, Vj = y satisfies all the binary constraints on Vi, Vj).

Algorithm AC-2

A generalised version of the Waltz's labelling algorithm.

In every step, the arcs going back from a given vertex are processed (i.e. a sub-graph of visited nodes is AC)

Algorithm AC-2

What is wrong with AC-1?

If a single domain is pruned then revisions of all the arcs are repeated even if the pruned domain does not influence most of these arcs.

What arcs should be reconsidered for revisions?

- The arcs whose consistency is affected by the domain pruning, i.e., the arcs pointing to the changed variable.
- We can omit one more arc!
  - Omit the arc running out of the variable whose domain has been changed (this arc is not affected by the domain change).

Algorithm AC-3

Re-revisions can be done more elegantly than in AC-2.
1) one queue of arcs for (re-)revisions is enough
2) only the arcs affected by domain reduction are added to the queue (like AC-2).

Algorithm AC-3

AC-3 schema is the most widely used consistency algorithm but it is still not optimal (time complexity is O(ed^3)).
Observation (AC-3):
- Many pairs of values are tested for consistency in every arc revision.
- These tests are repeated every time the arc is revised.

1. When the arc \( V_2, V_1 \) is revised, the value \( a \) is removed from domain of \( V_2 \).
2. Now the domain of \( V_2 \) should be explored to find out if any value \( a, b, c \) loses the support in \( V_2 \).

The support set for \( a \in D_j \), is the set \( \langle j, b \rangle \mid b \in D_j, (a, b) \in C_j \rangle \)

Cannot we compute the support sets once and then use them during re-revisions?

Using support sets:

Situation:
- we have just processed the arc \( (i, j) \) in \textsc{initialize}

Using the support sets:
- 1. Let \( b_3 \) be deleted from the domain of \( j \) (for some reason).
- 2. Look at \( S_{j,b_3} \) to find out the values that were supported by \( b_3 \) (i.e. \( \langle i, a \rangle \langle i, a_2 \rangle \)).
- 3. Decrease the counter for these values (i.e. tell them that they lost one support).
- 4. If any counter becomes zero (\( a_3 \)) then delete the value and repeat the procedure with the respective value (i.e., go to 1).

Algorithm AC-4

The algorithm AC-4 has optimal worst case time complexity \( O(ed^2) \)!

Unfortunately the average time complexity is not so good ...

Directional arc consistency (DAC)

Observation 1:
AC has a directional character but a CSP is not directional.

Observation 2:
AC has to repeat arc revisions; the total number of revisions depends on the number of arcs but also on the size of domains (while cycle).

Is it possible to weaken AC in such a way that every arc is revised just once?

Definition:
CSP is directional arc consistent using a given order of variables iff every arc \( (i, j) \) such that \( i < j \) is arc consistent.

Again, every arc has to be revised, but revision in one direction is enough now.

Other AC algorithms
- AC-5 (Van Hentenryck, Deville, Teng, 1992)
- AC-6 (Bessière, 1994)
- AC-7 (Bessière, Freuder, Régin, 1999)
- AC-2000 (Bessière & Régin, 2001)
- AC-2001 (Bessière & Régin, 2001)
- AC-3.1 (Zhang & Yap, 2001)
1) Consistency of an arc is required just in one direction.
2) Variables are ordered

There is no directed cycle in the graph!

1) If arcs are explored in a "good" order, no revision has to be repeated!

Algorithm DAC-1

procedure DAC-1(G)
  for j = |nodes(G)| to 1 by -1
    for each arc (i,j) in G such that i<j
      REVISE((i,j))
    end for
  end for
end DAC-1

1 2 3 4 5

1 2

6 5

4

3

DAC visibly covers DAC (if CSP is AC then it is DAC as well)

So, is DAC useful?

Claim:

If the constraint graph forms a tree then DAC is enough to solve the problem without backtracks.

1. Apply DAC in the order from the root to the leaf nodes.
2. Label vertices starting from the root.

DAC guarantees that there is a value for the child node compatible with all the parents.

Relation between DAC and AC

Observation:

CSP is arc consistent iff for some order of the variables, the problem is directional arc consistent in both directions.

Is it possible to achieve AC by applying DAC in both primal and reverse direction?

In general NO, but ...

Example:

X in {1,2}, Y in {1}, Z in {1,2}, X ≠ Z, Y < Z

using the order X, Y, Z
there is no domain change

However if the order Z, Y, X is used first then we get AC!

Is arc consistency enough?

By using AC we can remove many incompatible values

Do we get a solution?

Do we know that there exists a solution?

Unfortunately, the answer to both above questions is NO!

Example:

X ≠ Z
X ≠ Y
Y ≠ Z

CSP is arc consistent
but there is no solution

So what is the benefit of AC?

Sometimes we have a solution after AC

• any domain is empty → no solution exists
• all the domains are singleton → we have a solution

In general, AC prunes the search space.

Path consistency (PC)

How to strengthen the consistency level?

More constraints are assumed together!

Definition:

• The path (V_0, V_1, ..., V_m) is path consistent iff for every pair of values x_D_0, y_D_m satisfying all the binary constraints on V_0, V_m there exists an assignment of variables V_1, ..., V_{m-1} such that all the binary constraints between the neighbouring variables V_i, V_{i+1} are satisfied.

• CSP is path consistent iff every path is consistent.

Some notes:

• Only the constraints between the neighboring variables must be satisfied

• It is enough to explore paths of length 2 (Montanary, 1974)

Relation between PC and AC

Does PC subsume AC (i.e. if CSP is PC, is it AC as well)?

• The arc (i, j) is consistent (AC) if the path (i, j, j) is consistent (PC)

• Thus PC implies AC

Is PC stronger than AC (Is there any CSP that is AC but it is not PC)?

Example:

X in {1,2}, Y in {1,2}, Z in {1,2}, X≠Z, X≥Y, Y≥Z

• It is AC, but not PC (X=1, Z=2 cannot be extended to X, Y, Z)

AC removes incompatible values from the domains, what will be done in PC?

• PC removes pairs of values

• PC makes constraints explicit (A≤B, B≤C ⇒ A≤C)

• A unary constraint = a variable's domain
Path revision

Constraints represented extensionally via matrixes. Path consistency is realized via matrix operations.

Example:
- A, B, C in \{1, 2, 3\}, B > 1
- A < C, A = B, B > C - 2

Algorithm PC-1

How to make the path (i, k, j) consistent?
- \( R_{ij} \leftarrow R_{ij} \land (R_{ik} \land R_{kk} \land R_{kj}) \)

How to make a CSP path consistent?
Repeated revisions of paths (of length 2) while any domain changes.

Algorithm PC-2

Paths in one direction only (attention: this is not DPC!)
After every revision, the affected paths are re-revised.

Algorithm PC-2

Algorithm PC-2

How to improve PC-1?
- Just a few "bits"
  - It is not necessary to keep all copies of \( Y^k \)
  - Some operations produce no modification (\( Y_{ii} = Y_{ii} \)
  - Half of the operations can be removed (\( Y_{ji} = Y_{ij} \))
- The grand problem
  - After domain change all the paths are re-revised
  - It is enough to revise just the influenced paths

Algorithm of path revision

**Algorithm PC-3 (Mohr, Henderson 1986)**
- Based on computing supports for a value (like AC-4)
- If the pair (a, b) at the arc (i, j) is not supported by another variable, then a is removed from D_i and b is removed from D_j.
- This algorithm is not sound!

**Algorithm PC-4 (Han, Lee 1988)**
- Correction of the PC-3 algorithm
- Based on computing supports of pairs (b, c) at arc (i, j)

**Algorithm PC-5 (Singh 1995)**
- Uses the ideas behind AC-6
- Only one support is kept and a new support is looked for when the current support is lost
Drawbacks of PC

- **memory consumption**
  - because PC eliminates pairs of values, we need to keep all the compatible pairs extensionally, e.g. using (0,1)-matrix

- **bad ratio strength/efficiency**
  - PC removes more (or same) inconsistencies than AC, but the strength/efficiency ratio is much worse than for AC

- **modifies the constraint network**
  - PC adds redundant arcs (constraints) and thus it changes connectivity of the constraint network
  - this complicates using heuristics derived from the structure of the constraint network (like density, graph width etc.)

- **PC is still not a complete technique**
  - A, B, C, D in {1,2,3}
  - A ≠ B, A ≠ C, A ≠ D, B ≠ C, B ≠ D, C ≠ D
  - is PC but has no solution

Half way between AC and PC

Can we make a consistency algorithm:

- **stronger than AC**
- **without drawbacks of PC** (memory consumption, changing the constraint network)?

 Restricted path consistency (Berlandier 1995)

- based on AC-4 (uses the support sets)
- as soon as a value has only one support in another variable, PC is evoked for this pair of values

k-consistency

Is there a common formalism for AC and PC?

- AC: a value is extended to another variable
- PC: a pair of values is extended to another variable
- ... we can continue

**Definition:**

CSP is **k-consistent** iff any consistent assignment of (k-1) different variables can be extended to a consistent assignment of one additional variable.

![4-consistent graph](image)

Strong k-consistency

**Definition:**

CSP is strongly k-consistent iff it is j-consistent for every j ≤ k.

Visibly: strongly k-consistency ⇒ k-consistency

Moreover: strongly k-consistency ⇒ j-consistency ∀ j ≤ k

In general: ¬ k-consistency ⇒ strongly k-consistency

- NC = strong 1-consistency = 1-consistency
- AC = (strong) 2-consistency
- PC = (strong) 3-consistency
- sometimes we call NC+AC+PC together strong path consistency

What k-consistency is enough?

Assume that the number of vertices is n. What level of consistency do we need to find out the solution?

- **Strong n-consistency for graphs with n vertices!**
  - n-consistency is not enough - see the previous example
  - strong k-consistency where k=n is not enough as well

![Graph with n vertices](image)

And what about this graph?

(3)AC is enough! Because this a tree.

Think globally

CSP describes the problem locally:

- the constraints restrict small sets of variables
- heterogeneous real-life constraints
- missing global view
- weaker domain filtering

Global constraints

- global reasoning over a local sub-problem
- using semantic information to improve efficiency

**Example:**

- local (arc) consistency deduces no pruning
- but some values can be removed

![Global constraints](image)
a set of binary inequality constraints among all variables
\[ X_1 \neq X_2, X_1 \neq X_3, \ldots, X_{k-1} \neq X_k \]
all different\((X_1,\ldots, X_k) = \{(d_1,\ldots, d_k) \mid \forall i \ d_i \in D_i \land \forall i,j \ i \neq j \rightarrow d_i \neq d_j\}\)
better pruning based on matching theory over bipartite graphs

Initialisation:
1) compute maximum matching
2) remove all edges that do not belong to any maximum matching

Propagation of deletions \((X_1 \neq a)\):
1) remove discharged edges
2) compute new maximum matching
3) remove all edges that do not belong to any maximum matching

So far we have two separate methods:
- depth-first search
  - complete (finds a solution or proves its non-existence)
  - too slow (exponential)
- consistency techniques
  - usually incomplete (inconsistent values stay in domains)
  - pretty fast (polynomial)

Share advantages of both approaches - combine them!
- label the variables step by step (backtracking)
- maintain consistency after assigning a value

Do not forget about traditional solving techniques!
- Linear equality solvers, simplex …
- such techniques can be integrated to global constraints!
There is also local search.

"Maintain" consistency among the already labelled variables.
"Look back" = look to already labelled variables
What's result of consistency maintenance among labelled variables?
- a conflict (and/or its source - a violated constraint)

Backward consistency checks

Forward checking
- It is better to prevent failures than to detect them only!
- Consistency techniques can remove incompatible values for future (=not yet labelled) variables.
- Forward checking ensures consistency between the currently labelled variable and the variables connected to it via constraints.

\[ Q \leftarrow (V_i, V_j) \text{ in arcs}(G), i > cv \]  
\[ \text{consistent} \leftarrow \text{true} \]  
\[ \text{while} \ \text{consistent} \land \text{Q non empty do} \]  
\[ \text{select and delete any arc } (V_i, V_j) \text{ from } Q \]  
\[ \text{consistent} \leftarrow \text{not REVISE}(V_i, V_j) \]  
\[ \text{end while} \]  
\[ \text{return consistent} \]  
end AC-BT

Backjumping & comp. uses information about the violated constraints.
Partial Look Ahead

We can extend the consistency checks to more future variables! The value assigned to the current variable can be propagated to all future variables.

Partial lookahead consistency checks

procedure DAC-LA(G,cv)
  for i=cv+1 to n do
    for each arc (Vi,Vj) in arcs(G) such that i>j & j≥cv do
      if REVISE(Vi,Vj) then
        if empty Di then return fail
      end for
    end for
  end for
  return true
end DAC-LA

Procedure 

Partial look ahead

Knowing more about far future is an advantage! Instead of DAC we can use a full AC (e.g. AC-3).

Full look ahead consistency checks

procedure AC3-LA(G,cv)
  Q ← { (Vi,Vcv) in arcs(G), i>cv} % start with arcs going to cv
  consistent ← true
  while consistent & Q non empty do
    select and delete any arc (Vk,Vm) from Q
    if REVISE(Vk,Vm) then
      Q ← Q ∪ { (Vi,Vk) | (Vi,Vk) in arcs(G), i≠k, i≠m, i>cv }
      consistent ← not empty Dk
    end if
  end while
  return consistent
end AC3-LA

Notes:
- The arcs going to the current variable are checked exactly once.
- The arcs to past variables are not checked at all.
- It is possible to use other than AC-3 algorithms (e.g. AC-4)

Comparison

4 queens

Backtracking is not very good
19 attempts

Forward checking is better
3 attempts

And the winner is Look Ahead
2 attempts

Consistency and Search

Consistency techniques are (usually) incomplete.
% We need a search algorithm to resolve the rest!

Labeling
- depth-first search
  - assign a value to the variable
  - propagate = make the problem locally consistent
  - backtrack upon failure
- X in 1..3 = X=1 ∨ X=2 ∨ X=3 (enumeration)

In general, search algorithm resolves remaining disjunctions!
- X=1 ∨ X=1 (step labeling)
- X=2 ∨ X=3 (bisection)
- X≤Y ∨ Y≤X (variable ordering)

Variable ordering

Variable ordering in labelling influence significantly efficiency of solvers (e.g. in a tree-structured CSP).

What variable ordering should be chosen in general?

FAIL-FIRST principle

- select the variable whose instantiation will lead to a failure

It is better to tackle failures earlier, they can be become even harder
- prefer the variables with smaller domain (dynamic order)
  - a smaller number of choices ~ lower probability of success
  - the dynamic order is appropriate only when new information appears during solving (e.g., in look ahead algorithms)

Solve the hard cases first, they may become even harder later.
- prefer the most constrained variables
  - it is more complicated to label such variables (it is possible to assume complexity of satisfaction of the constraints)
  - this heuristic is used when there is an equal size of the domains
- prefer the variables with more constraints to past variables
  - a static heuristic that is useful for look-back techniques
**Definition:**
CSP is solved using backtrack-free search if for some order of variables we can find a value for each variable compatible with the values of already assigned variables.

How to find out a sufficient consistency level for a given graph?

Some observations:
- A variable must be compatible with all the “former” variables.
- Let m be the maximum of backward edges for all the vertices.
- If m is the maximum, strong (m+1)-consistency is enough.
- The order minimizing m is looked for.

Value ordering

Order of values in labelling influence significantly efficiency (if we choose the right value each time, no backtrack is necessary).

Some strategies:
- Prefer the values belonging to the solution.
- Prefer the values with more supports.
- Prefer the value leading to less domain reduction.
- Prefer the value simplifying the problem.

Golomb ruler

A base model:
Variables $X_1, \ldots, X_M$ with the domain $0..M*M$.
- $X_1 = 0$ (ruler start).
- No permutations of variables.
- $D_{ij} = X_j - X_i$ (difference variables).
- All different ($D_{ij}, D_{i,j+1}, \ldots, D_{j,M-i+1}$).

Model extensions:
- Symmetry breaking.
- Better bounds (implied constraints) for $D_{ij}$.

What is the effect of different constraint models?

What is the effect of different search strategies?
Conclusions

It is not necessary to program all the presented techniques from scratch!

Use existing constraint solvers (packages)!
- provide implementation of data structures for modeling variables' domains and constraints
- provide a basic consistency framework (AC-3)
- provide filtering algorithms for many constraints (including global constraints)
- provide basic search strategies
- usually extensible (new filtering algorithms, new search strategies)

Examples of constraint solvers:
- Prolog: CHIP, ECLiPSe, SICStus Prolog, Prolog IV, GNU Prolog, IF/Prolog
- C/C++: CHIP++, ILOG Solver
- Java: JCL, JCL, Koalog
- Mozart

Resources

Books
- R. Dechter: Constraint Processing, Morgan Kaufmann, 2003

Journal

On-line materials
- On-line Guide to Constraint Programming (tutorial)
  http://kti.mff.cuni.cz/~bartak/constraints/
- Constraints Archive (archive and links)
  http://4c.ucc.ie/web/archive/index.jsp
- Constraint Programming online (community web)
  http://www.cp-online.org/

Summary

Constraints
- arbitrary relations over the problem variables
- express partial local information in a declarative way

Basic constraint satisfaction framework:
- local consistency: connecting filtering algorithms for individual constraints
- depth-first search resolves remaining disjunctions
- local search can also be used

Problem solving using constraints:
- declarative modeling of problems as a CSP
- dedicated algorithms can be encoded in constraints
- special search strategies

It is easy to state combinatorial problems in terms of a CSP
... but it is more complicated to design solvable models.

We still did not reach the Holy Grail of computer programming (the user states the problem, the computer solves it) but CP is close.