Constraint Programming

Constraint Programming

- *Basic idea:* Programming with constraints, i.e. constraint solving embedded in a programming language
- *Constraints:* linear, non-linear, finite domain, Boolean, ...
- *Programming:* logic, functional, object-oriented, imperative, concurrent, ...
  mathematical programming vs. computer programming
- *Systems:* Prolog III/IV, CHIP, ECLIPSE, ILOG, CHOCO, Gecode, JaCoP ...

Finite Domain Constraints

Constraint satisfaction problem (CSP)

- n variables $x_1, \dots, x_n$
- For each variable x_j a *finite domain* D_j of possible values, often $D_j \subset \mathbb{N}$.
- m constraints $C_1, \dots, C_m$, where $C_i \subseteq D_{i_1} \times \dots \times D_{i_{k_i}}$ is a relation between k_i variables $x_{i_1}, \dots, x_{i_{k_i}}$. Write also $C_{i_1,\dots,i_{k_i}}$.
- A *solution* is an assignment of a value D_j to x_j, for each $j = 1, \dots, n$, such that all relations C_i are satisfied.

Coloring Problem

- Decide whether a map can be colored by 3 colors such that neighboring regions get different colors.
- For each region a variable $\mathbf{x_j}$ with domain $D_j = \{\text{red, green, blue}\}$.
- For each pair of variables x_i, x_j corresponding to two neighboring regions, a constraint $\mathbf{x_i \neq x_j}$.
- NP-complete problem.

Resolution by Backtracking

- Instantiate the variables in some order.
- As soon as all variables in a constraint are instantiated, determine its truth value.
- If the constraint is not satisfied, backtrack to the last variable whose domain contains unassigned values, otherwise continue instantiation.

Efficiency Problems

Mackworth 77

1. If the domain D_j of a variable x_j contains a value v that does not satisfy C_j, this will be the cause of repeated instantiation followed by immediate failure.
2. If we instantiate the variables in the order $x_1, x_2, \dots, x_n$, and for $x_i = v$ there is no value $w \in D_j$, for $j > i$, such that $C_{ij}(v, w)$ is satisfied, then backtracking will try all values for x_j, fail and try all values for x_{j-1} (and for each value of x_{j-1} again all values for x_j), and so on until it tries all combinations of values for $x_{i+1}, \dots, x_j$ before finally discovering that v is not a possible value for x_j.

   The identical failure process may be repeated for all other sets of values for $x_1, \dots, x_{i-1}$ with $x_i = v$.

Local Consistency

- Consider CSP with unary and binary constraints only.
- *Constraint graph G*
  - For each variable x_i a node i.
  - For each pair of variables x_i, x_j occurring in the same binary constraint, two arcs (i,j) and (j,i).
- The node i is *consistent* if $C_i(v)$, for all $v \in D_i$.
- The arc (i,j) is *consistent*, if for all $v \in D_i$ with $C_i(v)$ there exists $w \in D_j$ with $C_j(w)$ such that $C_{ij}(v,w)$.
- The graph is *node consistent* resp. *arc consistent* if all its nodes (resp. arcs) are consistent.

Arc Consistency

Algorithm AC-3 (Mackworth 77):

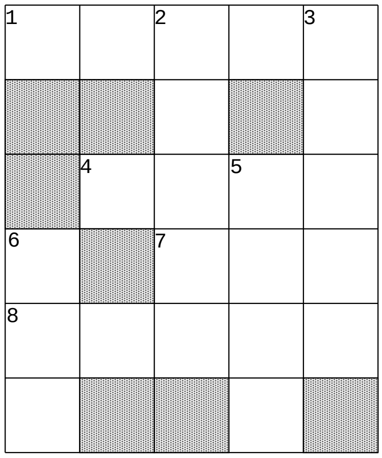
```
begin
   for i ← 1 until n do D_i ← {v ∈ D_i | C_i(v)};
   Q ← {(i,j) | (i,j) ∈ arcs(G), i ≠ j}
   while Q not empty do
      begin
         select and delete an arc (i,j) from Q;
         if REVISE(i,j) then
               Q ← Q ∪ {(k,i) | (k,i) ∈ arcs(G), k ≠ i, k ≠ j}
      end
end
```

Arc Consistency (2)

```
procedure REVISE(i,j):
begin
   DELETE ← false
   for each v ∈ D_i do
      if there is no w ∈ D_j such that C_ij(v,w) then
         begin
            delete v from D_i;
            DELETE ← true
         end;
   return DELETE
end
```

Crossword Puzzle

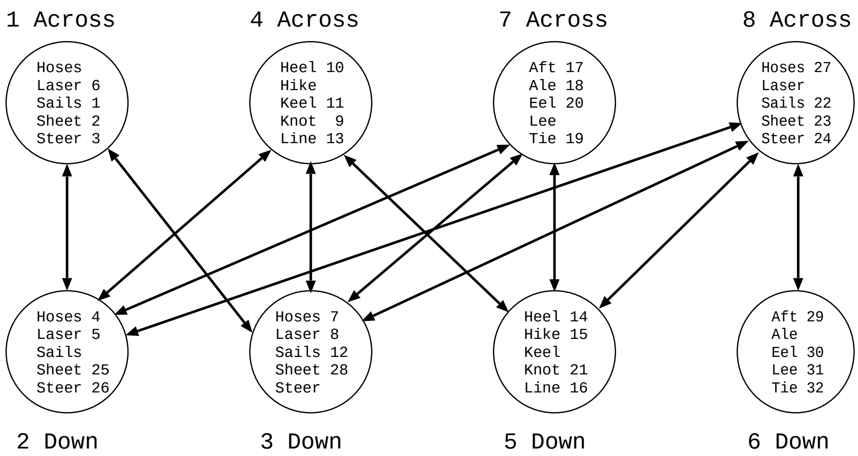
Dechter 92



Word List

Aft	Laser
Ale	Lee
Eel	Line
Heel	Sails
Hike	Sheet
Hoses	Steer
Keel	Tie
Knot	

Solution



Lookahead

Apply local consistency dynamically during search

- *Forward Checking:* After assigning to x the value v, eliminate for all uninstantiated variables y the values from D_y that are incompatible with v.
- *Partial Lookahead:* Establish arc consistency for all (y, y'), where y, y' have not been instantiated yet and y will be instantiated before y'.
- *Full Lookahead:* Establish arc consistency for all uninstantiated variables.

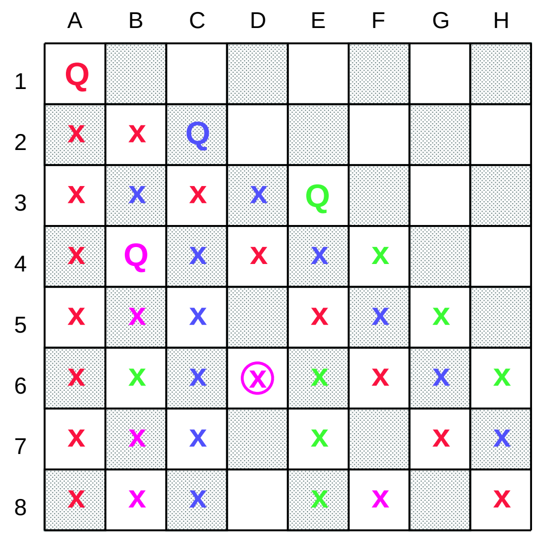
n-Queens Problem

Place n queens in an $n \times n$ chessboard such that no two queens threaten each other.

- *Variables* $x_i, i = 1, \dots, n$ with domain $D_i = \{1, \dots, n\}$ indicating the column of the queen in line i.
- *Constraints*
  - $x_i \neq x_j$, for $1 \leq i < j \leq n$ (vertical)
  - $x_i \neq x_j + (j - i)$, for $1 \leq i < j \leq n$ (diagonal 1)
  - $x_i \neq x_j - (j - i)$, for $1 \leq i < j \leq n$ (diagonal 2)

Forward Checking (2)

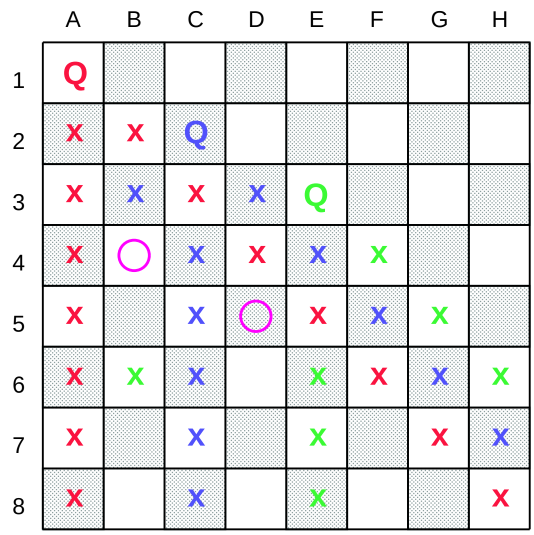
Forward Checking



1A
2C
3E
4BG
5B
6D
5D
4H
5B
6D
7F
6 (no more value)
5D
4 (no more value)
3F

Partial Lookahead (3)

Partial Lookahead

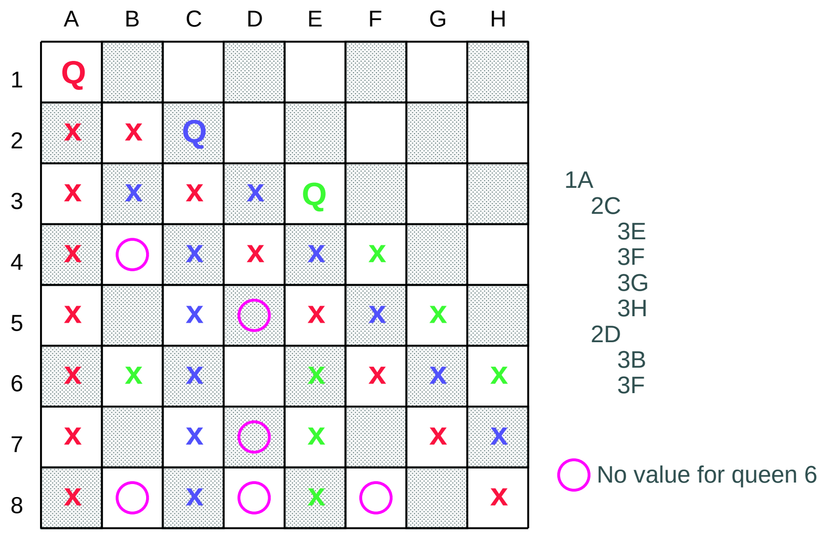


1A
2C
3E (delete 4B and 5D)
4GH
5B (no value left for 6)
3F (delete 6D and 6E)
4BH (failed, backtrack to 4)
3G (delete 5D and 7E)
4B

○ No value for queen 6

Full Lookahead (4)

Full Lookahead



Typical structure of a constraint program

- Declare the variables and their domains
- State the constraints
- Enumeration (`labeling`)

The constraint solver achieves only local consistency.

In order to get global consistency, the domains have to be enumerated.

Labeling

- Assigning to the variables their possible values and constructing the corresponding search tree.
- *Important questions*
  1. In which order should the variables be instantiated (variable selection) ?
  2. In which order should the values be assigned to a selected variable (value selection) ?
- Static vs. dynamic orderings
- *Heuristics*

Dynamic variable/value orderings

- Variable orderings
  - Choose the variable with the smallest domain *"first fail"*
  - Choose the variable with the smallest domain that occurs in most of the constraints *"most constrained"*
  - Choose the variable which has the smallest/largest lower/upper bound on its domain.

- Value orderings
  - Try first the minimal value in the current domain.
  - Try first the maximal value in the current domain.
  - Try first some value in the middle of the current domain.

Constraint programming systems

System	Avail.	Constraints	Language	Web site
B-prolog	comm.	FinDom	Prolog	www.probp.com
CHIP	comm.	FinDom, Boolean, Linear $\mathbb{Q}$ Hybrid	Prolog, C, C++	www.cosytec.com
Choco	free	FinDom	Java	choco.emn.fr
Eclipse	free non-profit	FinDom, Hybrid	Prolog	eclipseclp.org
Gecode	free	FinDom	C++	www.gecode.org
GNU Prolog	free	FinDom	Prolog	gnu-prolog.inria.fr
IF/Prolog	comm.	FinDom Boolean, Linear $\mathbb{R}$	Prolog	www.ifcomputer.com/IFProlog/
ILOG	comm.	FinDom, Hybrid	C++, Java	www-01.ibm.com/software/ integration/optimization/cplex-cp-optimizer/
JaCoP	free	FinDom	Java	jacop.osolpro.com
NCL	comm.	FinDom		www.enginest.com
Mozart	free	FinDom	Oz	www.mozart-oz.org
Prolog IV	comm.	FinDom, nonlinear intervals	Prolog	www.prologia.fr
Sicstus	comm.	FinDom, Boolean, linear $\mathbb{R}/\mathbb{Q}$	Prolog	www.sics.se/sicstus/