Chapter 3: Finite Constraint Domains

Where we meet the simplest and yet most difficult constraints, and some clever and not so clever ways to solve them

Finite Constraint Domains

- Constraint Satisfaction Problems
- A Backtracking Solver
- Node and Arc Consistency
- Bounds Consistency
- Generalized Consistency
- Optimization for Arithmetic CSPs

Finite Constraint Domains

- An important class of constraint domains
- Use to model constraint problems involving choice: e.g. scheduling, routing and timetabling
- The greatest industrial impact of constraint programming has been on these problems

Constraint Satisfaction Problems

- A **constraint satisfaction problem** (**CSP**) consists of:
  - a constraint C over variables $x1,..., xn$
  - a domain D which maps each variable xi to a set of possible values $D(xi)$
- It is understood as the constraint

$$C \wedge x1 \in D(x1) \wedge \cdots \wedge xn \in D(xn)$$

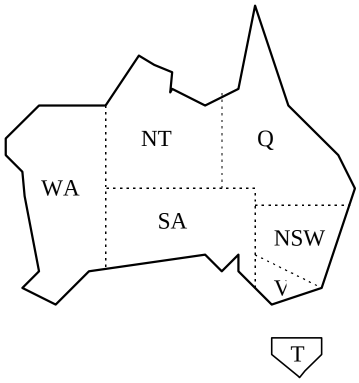
Map Colouring

A classic CSP is the problem of coloring a map so that no adjacent regions have the same color

Can the map of Australia be colored with 3 colors ?

$$WA \neq NT \wedge WA \neq SA \wedge NT \neq SA \wedge$$
$$NT \neq Q \wedge SA \neq Q \wedge SA \neq NSW \wedge$$
$$SA \neq V \wedge Q \neq NSW \wedge NSW \neq V$$

$$D(WA) = D(NT) = D(SA) = D(Q) =$$
$$D(NSW) = D(V) = D(T) =$$
$$\{red, yellow, blue\}$$

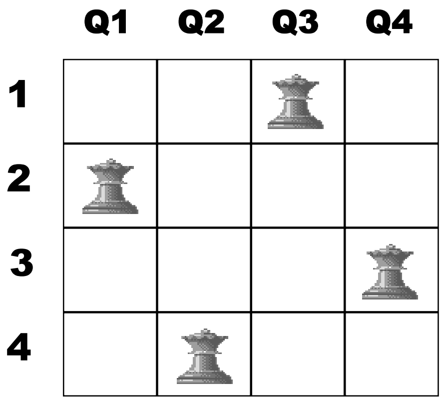


4-Queens

Place 4 queens on a 4 x 4 chessboard so that none can take another.

Four variables Q1, Q2, Q3, Q4 representing the row of the queen in each column. Domain of each variable is {1,2,3,4}

One solution! -->



4-Queens

The constraints:

Not on the same row

$$Q1 \neq Q2 \wedge Q1 \neq Q3 \wedge Q1 \neq Q4 \wedge$$
$$Q2 \neq Q3 \wedge Q2 \neq Q4 \wedge Q3 \neq Q4 \wedge$$

Not diagonally up

$$Q1 \neq Q2+1 \wedge Q1 \neq Q3+2 \wedge Q1 \neq Q4+3 \wedge$$
$$Q2 \neq Q3+1 \wedge Q2 \neq Q4+2 \wedge Q3 \neq Q4+1 \wedge$$

Not diagonally down

$$Q1 \neq Q2-1 \wedge Q1 \neq Q3-2 \wedge Q1 \neq Q4-3 \wedge$$
$$Q2 \neq Q3-1 \wedge Q2 \neq Q4-2 \wedge Q3 \neq Q4-1$$

Smugglers Knapsack

Smuggler with knapsack with capacity 9, who needs to choose items to smuggle to make profit at least 30

object	*profit*	*size*
whiskey	15	4
perfume	10	3
cigarretes	7	2

$$4W + 3P + 2C \leq 9 \wedge 15W + 10P + 7C \geq 30$$

What should be the domains of the variables?

Simple Backtracking Solver

- The simplest way to solve CSPs is to enumerate the possible solutions
- The **backtracking solver**:
  - enumerates values for one variable at a time
  - checks that no prim. constraint is false at each stage
- Assume *satisfiable(c)* returns *false* when primitive constraint *c* with no variables is unsatisfiable

Partial Satisfiable

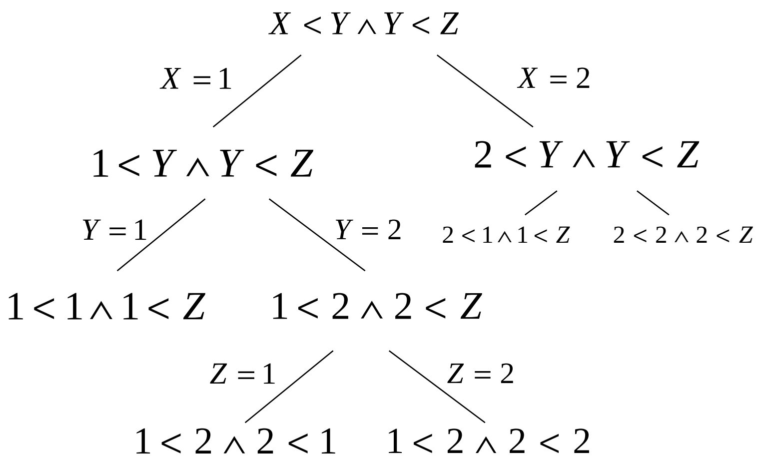
- Check whether a constraint is unsatisfiable because of a prim. constraint with no vars
- partial_satisfiable(*C*)
  - **for** each primitive constraint *c* in *C*
    - **if** *vars(c)* is empty
      - **if** *satisfiable(c) = false* **return** *false*
  - **return** *true*

Backtrack Solve

- back_solve(C,D)
  - **if** *vars(C)* is empty **return** partial_satisfiable*(C)*
  - choose *x* in *vars(C)*
  - **for** each value *d* in *D(x)*
    - let *C1* be *C* with *x* replaced by *d*
    - **if** partial_satisfiable(*C1*) **then**
      - **if** back_solve(*C1*,*D*) **then return** *true*
  - **return** *false*

Backtracking Solve

$X < Y \wedge Y < Z \qquad D(X) = D(Y) = D(Z) = \{1,2\}$



Node and Arc Consistency

- **basic idea**: find an equivalent CSP to the original one with smaller domains of vars
- **key:** examine 1 prim.constraint *c* at a time
- **node consistency**: (*vars(c)={x}*) remove any values from domain of *x* that falsify *c*
- **arc consistency**: (*vars(c)={x,y}*) remove any values from *D(x)* for which there is no value in *D(y)* that satisfies *c* and vice versa

Node consistency

- Primitive constraint *c* is **node consistent** with domain *D* if |*vars(c)*| *!=1* or
  - if *vars(c) = {x}* then for each *d* in *D(x)*
  - *x* assigned *d* is a solution of *c*
- A CSP is node consistent if each prim. constraint in it is node consistent

Node Consistency Examples

Example CSP is not node consistent (see *Z*)

$$X < Y \wedge Y < Z \wedge Z \leq 2$$
$$D(X) = D(Y) = D(Z) = \{1,2,3,4\}$$

This CSP is node consistent

$$X < Y \wedge Y < Z \wedge Z \leq 2$$
$$D(X) = D(Y) = \{1,2,3,4\},\ D(Z) = \{1,2\}$$

The map coloring and 4-queens CSPs are node consistent. Why?

Achieving Node Consistency

- node_consistent(*C*,*D*)
  - **for** each prim. constraint *c* in *C*
    - *D* := node_consistent_primitive(*c*, *D*)
  - **return** *D*
- node_consistent_primitive(*c*, *D*)
  - **if** |*vars(c)*| =*1* **then**
    - let *{x}* = *vars(c)*

      $D(x) := \{d \in D(x) | \{x \mapsto d\} \text{ is a solution of } c\}$
  - **return** *D*

Arc Consistency

- A primitive constraint c is **arc consistent** with domain D if $|vars\{c\}| \, != 2$ or
  - $vars(c) = \{x,y\}$ and for each d in $D(x)$ there exists e in $D(y)$ such that

    $\{x \mapsto d, y \mapsto e\}$ is a solution of c
  - and similarly for y
- A CSP is arc consistent if each prim. constraint in it is arc consistent

Arc Consistency Examples

This CSP is node consistent but not arc consistent

$$X < Y \wedge Y < Z \wedge Z \leq 2$$

$$D(X) = D(Y) = \{1,2,3,4\}, \; D(Z) = \{1,2\}$$

For example the value 4 for X and $X < Y$.

The following equivalent CSP is arc consistent

$$X < Y \wedge Y < Z \wedge Z \leq 2$$

$$D(X) = D(Y) = D(Z) = \emptyset$$

The map coloring and 4-queens CSPs are also arc consistent.

Achieving Arc Consistency

- arc_consistent_primitive(c, D)
  - **if** $|vars(c)| = 2$ **then**

    $D(x) := \{d \in D(x) \mid \text{exists } e \in D(y),$
    $\{x \mapsto d, y \mapsto e\} \text{ is a soln of } c\}$

    $D(y) := \{e \in D(y) \mid \text{exists } d \in D(x),$
    $\{x \mapsto d, y \mapsto e\} \text{ is a soln of } c\}$
  - **return** D
- removes values which are not arc consistent with c

Achieving Arc Consistency

- arc_consistent(C,D)
  - **repeat**
    - W := D
    - **for** each prim. constraint c in C
      - D := arc_consistent_primitive(c,D)
  - **until** W = D
  - **return** D
- A very naive version (there are much better)

Using Node and Arc Cons.

- We can build constraint solvers using the consistency methods
- Two important kinds of domain
  - **false domain**: some variable has empty domain
  - **valuation domain**: each variable has a singleton domain
- extend *satisfiable* to CSP with val. domain

Node and Arc Cons. Solver

- *D* := node_consistent(*C*,*D*)
- *D* := arc_consistent(*C*,*D*)
- **if** *D* is a false domain
  - **return** *false*
- **if** *D* is a valuation domain
  - **return** *satisfiable*(*C*,*D*)
- **return** *unknown*

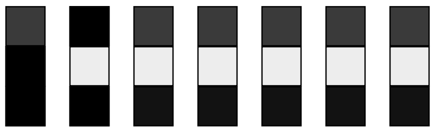
Node and Arc Solver Example

Colouring Australia: with constraints

$\boxed{WA = red} \wedge \boxed{NT = yellow}$

WA NT SA Q NSW V T

Node consistency



$WA \neq NT$ $WA \neq SA$ $NT \neq SA$

$NT \neq Q$ $SA \neq Q$ $SA \neq NSW$

$SA \neq V$ $Q \neq NSW$ $NSW \neq V$

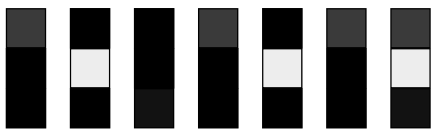
Node and Arc Solver Example

Colouring Australia: with constraints

$\boxed{WA = red} \wedge \boxed{NT = yellow}$

WA NT SA Q NSW V T

Arc consistency



Answer:

unknown

$\boxed{WA \neq NT}$ $\boxed{WA \neq SA}$ $\boxed{NT \neq SA}$

$\boxed{NT \neq Q}$ $\boxed{SA \neq Q}$ $\boxed{SA \neq NSW}$

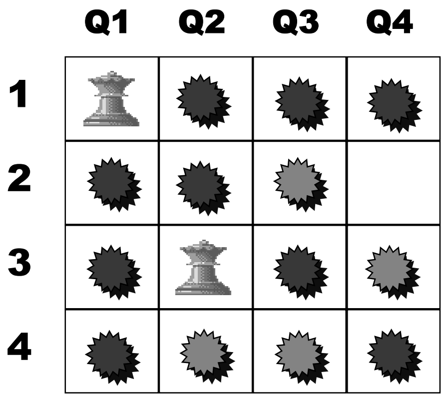
$\boxed{SA \neq V}$ $\boxed{Q \neq NSW}$ $\boxed{NSW \neq V}$

Backtracking Cons. Solver

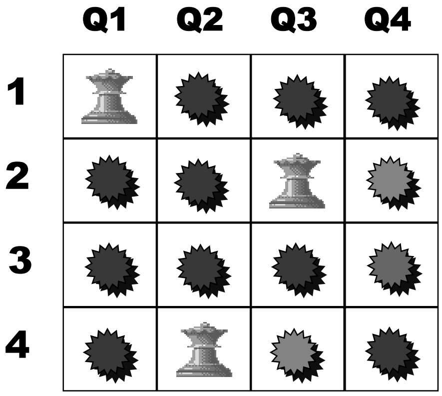
- We can combine consistency with the backtracking solver
- Apply node and arc consistency before starting the backtracking solver and after each variable is given a value

Back. Cons Solver Example

No value can be assigned to Q3 in this case!

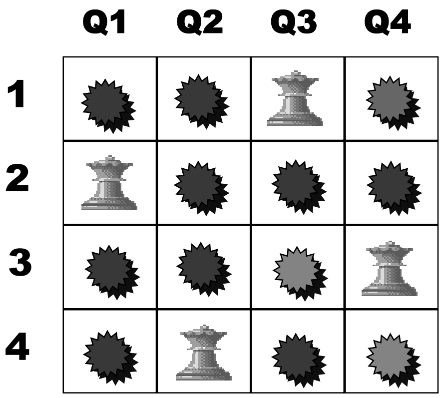


Back. Cons Solver Example



We cannot find any possible value for Q4 in this case!

Back. Cons Solver Example

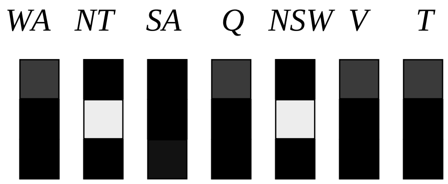


Node and Arc Solver Example

Colouring Australia: with constraints

$\boxed{WA = red} \wedge \boxed{NT = yellow}$

Backtracking enumeration



Select a variable with domain of more than 1, T

Add constraint $T = red$ Apply consistency

Answer: *true*

Bounds Consistency

- What about prim. constraints with more than 2 variables?
- **hyper-arc consistency**: extending arc consistency to arbitrary number of variables
- Unfortunately determining hyper-arc consistency is NP-hard (so its probably exponential)
- What is the solution?

Bounds Consistency

- **arithmetic CSP:** constraints are integer
- **range:** [*l..u*] represents the set of integers *{l, l+1, ..., u}*
- **idea** use real number consistency and only examine the endpoints (upper and lower bounds) of the domain of each variable
- Define *min(D,x)* as minimum element in domain of *x*, similarly for *max(D,x)*

Bounds Consistency

- A prim. constraint *c* is **bounds consistent** with domain *D* if for each var *x* in *vars(c)*
  - exist real numbers *d1, ..., dk* for remaining vars *x1, ..., xk* such that
    $$\{x \mapsto \min(D,x), x1 \mapsto d1, \ldots xk \mapsto dk\}$$
  - is a solution of *c*
  - and similarly for $\{x \mapsto \max(D,x)\}$
- An arithmetic CSP is bounds consistent if all its primitive constraints are

Bounds Consistency Examples

$$X = 3Y + 5Z$$

$$D(X) = [2..7], D(Y) = [0..2], D(Z) = [-1..2]$$

Not bounds consistent, consider *Z=2*, then *X-3Y=10*

But the domain below is bounds consistent

$$D(X) = [2..7], D(Y) = [0..2], D(Z) = [0..1]$$

Compare with the hyper-arc consistent domain

$$D(X) = \{3,5,6\}, D(Y) = \{0,1,2\}, D(Z) = \{0,1\}$$

Achieving Bounds Consistency

- Given a current domain *D* we wish to modify the endpoints of domains so the result is bounds consistent
- **propagation rules** do this

Achieving Bounds Consistency

Consider the primitive constraint $X = Y + Z$ which is equivalent to the three forms

$$X = Y + Z \quad Y = X - Z \quad Z = X - Y$$

Reasoning about minimum and maximum values:

$$X \geq \min(D,Y) + \min(D,Z) \quad X \leq \max(D,Y) + \max(D,Z)$$
$$Y \geq \min(D,X) - \max(D,Z) \quad Y \leq \max(D,X) - \min(D,Z)$$
$$Z \geq \min(D,X) - \max(D,Y) \quad Z \leq \max(D,X) - \min(D,Y)$$

Propagation rules for the constraint $X = Y + Z$

Achieving Bounds Consistency

$$X = Y + Z$$
$$D(X) = [4..8], D(Y) = [0..3], D(Z) = [2..2]$$

The propagation rules determine that:

$$(0+2=)\ 2 \leq X \leq 5\ (=3+2)$$
$$(4-2=)\ 2 \leq Y \leq 6\ (=8-2)$$
$$(4-3=)\ 1 \leq Z \leq 8\ (=8-0)$$

Hence the domains can be reduced to

$$D(X) = [4..5], D(Y) = [2..3], D(Z) = [2..2]$$

More propagation rules

$$4W + 3P + 2C \le 9$$

$$W \le \frac{9}{4} - \frac{3}{4}\min(D,P) - \frac{2}{4}\min(D,C)$$

$$P \le \frac{9}{3} - \frac{4}{3}\min(D,W) - \frac{2}{3}\min(D,C)$$

$$C \le \frac{9}{2} - \frac{4}{2}\min(D,W) - \frac{3}{2}\min(D,P)$$

Given initial domain:

$$D(W) = [0..9], D(P) = [0..9], D(C) = [0..9]$$

We determine that $W \le \frac{9}{4},\ P \le \frac{9}{3},\ C \le \frac{9}{2}$

new domain: $D(W) = [0..2], D(P) = [0..3], D(C) = [0..4]$

Disequations $Y \neq Z$

Disequations give weak propagation rules, only when one side takes a fixed value that equals the minimum or maximum of the other is there propagation

$D(Y) = [2..4], D(Z) = [2..3]$ no propagation

$D(Y) = [2..4], D(Z) = [3..3]$ no propagation

$D(Y) = [2..4], D(Z) = [2..2]$ prop $D(Y) = [3..4], D(Z) = [2..2]$

Multiplication $X = Y \times Z$

If all variables are positive its simple enough

$X \geq \min(D,Y) \times \min(D,Z) \quad X \leq \max(D,Y) \times \max(D,Z)$

$Y \geq \min(D,X) / \max(D,Z) \quad Y \leq \max(D,X) / \min(D,Z)$

$Z \geq \min(D,X) / \max(D,Y) \quad Z \leq \max(D,X) / \min(D,Y)$

Example: $D(X) = [4..8], D(Y) = [1..2], D(Z) = [1..3]$

becomes: $D(X) = [4..6], D(Y) = [2..2], D(Z) = [2..3]$

But what if variables can be 0 or negative?

Multiplication $X = Y \times Z$

Calculate X bounds by examining extreme values

$$X \geq \text{minimum}\{\min(D,Y) \times \min(D,Z), \min(D,Y) \times \max(D,Z) \\ \max(D,Y) \times \min(D,Z), \max(D,Y) \times \max(D,Z)\}$$

Similarly for upper bound on X using maximum

BUT this does not work for Y and Z? As long as *min(D,Z) <0* and *max(D,Z)>0* there is no bounds restriction on Y

$$X = Y \times Z \quad \{X \mapsto 4, Y \mapsto d, Z \mapsto 4/d\}$$

Recall we are using **real** numbers (e.g. 4/*d*)

Multiplication $X = Y \times Z$

We can wait until the range of Z is non-negative or non-positive and then use rules like

$$Y \geq \text{minimum}\{\min(D,X)/\min(D,Z), \min(D,X)/\max(D,Z) \\ \max(D,X)/\min(D,Z), \max(D,X)/\max(D,Z)\}$$

division by 0:

$$\frac{+\,ve}{0} = +\infty \qquad \frac{-\,ve}{0} = -\infty \qquad \frac{0}{0} = -\infty$$

Bounds Consistency Algm

- Repeatedly apply the propagation rules for each primitive constraint until there is no change in the domain
- We do not need to examine a primitive constraint until the domains of the variables involve are modified

Bounds consistency solver

- *D* := bounds_consistent(*C*,*D*)
- **if** *D* is a false domain
  - **return** *false*
- **if** *D* is a valuation domain
  - **return** *satisfiable*(*C*,*D*)
- **return** *unknown*

Back. Bounds Cons. Solver

- Apply bounds consistency before starting the backtracking solver and after each variable is given a value

Back. Bounds Solver Example

Smugglers knapsack problem (whiskey available)

capacity *profit*

$$4W + 3P + 2C \leq 9 \quad \wedge \quad 15W + 10P + 7C \geq 30$$

Current domain:

$$D(W) = [0..0], D(P) = [1..1], D(C) = [3..3]$$

Initial bounds consistency

W = 0

P = 1

(0,1,3)

Solution Found: return *true*

Back. Bounds Solver Example

Smugglers knapsack problem (whiskey available)

capacity *profit*

$$4W + 3P + 2C \leq 9 \quad \wedge \quad 15W + 10P + 7C \geq 30$$

Initial bounds consistency

W = 0 *W = 1* *W = 2*

P = 1 *P = 2* *P = 3* **(1,1,1)** **(2,0,0)**

(0,1,3) *false* *false* No more solutions

Generalized Consistency

- Can use any consistency method with any other communicating through the domain,
  - node consistency : prim constraints with 1 var
  - arc consistency: prim constraints with 2 vars
  - bounds consistency: other prim. constraints
- Sometimes we can get more information by using complex constraints and special consistency methods

Alldifferent

- *alldifferent({V1,...,Vn})* holds when each variable *V1,..,Vn* takes a different value
  - *alldifferent({X, Y, Z})* is equivalent to
    $X \neq Y \wedge X \neq Z \wedge Y \neq Z$
- Arc consistent with domain
  $D(X) = \{1,2\}, D(Y) = \{1,2\}, D(Z) = \{1,2\}$
- BUT there is no solution! specialized consistency for *alldifferent* can find it

Alldifferent Consistency

- let *c* be of the form *alldifferent(V)*
- **while** exists *v* in *V* where *D(v) = {d}*
  - *V := V - {v}*
  - **for** each *v'* in *V*
    - *D(v') := D(v') - {d}*
- *DV* := union of all *D(v)* for *v* in *V*
- **if** |*DV*| < |*V*| **then return** false domain
- **return** *D*

Alldifferent Examples

$$alldifferent(\{X, Y, Z\})$$

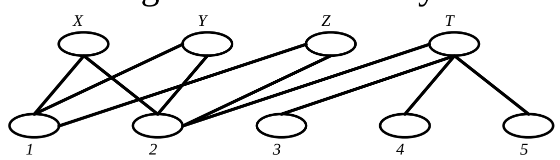
$$D(X) = \{1,2\}, D(Y) = \{1,2\}, D(Z) = \{1,2\}$$

DV = {1,2}, V={X,Y,Z} hence detect unsatisfiability

$$alldifferent(\{X, Y, Z, T\})$$

$$D(X) = \{1,2\}, D(Y) = \{1,2\}, D(Z) = \{1,2\}, D(T) = \{2,3,4,5\}$$

DV = {1,2,3,4,5}, V={X,Y,Z,T} don't detect unsat.
Maximal matching based consistency could



Other Complex Constraints

$cumulative([S_1,\ldots,S_n],[D_1,\ldots,D_n],[R_1,\ldots,R_n],L)$

- schedule *n* tasks with start times *Si* and durations *Di* needing resources *Ri* where *L* resources are available at each moment

$element(I,[V_1,\ldots,V_n],X)$

- array access if *I = i*, then *X = Vi* and if *X != Vi* then *I != i*

Optimization for CSPs

- Because domains are finite can use a solver to build a straightforward optimizer
- retry_int_opt(*C, D, f, best*)
  - *D2* := int_solv(*C*,*D*)
  - **if** *D2* is a false domain **then return** *best*
  - let *sol* be the solution corresponding to *D2*
  - **return** retry_int_opt(*C /\ f < sol(f), D, f, sol*)

Backtracking Optimization

- Since the solver may use backtrack search anyway combine it with the optimization
- At each step in backtracking search, if *best* is the best solution so far add the constraint $f < best(f)$

Back. Optimization Example

Smugglers knapsack problem (whiskey available)

capacity *profit*

$$4W + 3P + 2C \le 9 \quad \wedge \quad 15W + 10P + 7C \ge 30$$

Initial bounds consistency

W = 0 *W = 1* *W = 2*

P = 1 *P = 2* *P = 3* **(1,1,1)** *false*

(0,1,3) *false* *false* **Return last sol (1,1,1)**

Branch and Bound Opt.

- The previous methods,unlike simplex dont use the objective function to direct search
- **branch and bound** optimization for (C,f)
  - use simplex to find a real optimal,
  - if solution is integer stop
  - otherwise choose a var x with non-integer opt value d and examine the problems
    $(C \wedge x \leq \lfloor d \rfloor, f) \quad (C \wedge x \geq \lceil d \rceil, f)$
  - use the current best solution to constrain prob.

Branch and Bound Example

Smugglers knapsack problem

$W \leq 2$ $\quad$ $W \geq 3$

$P \leq 0$ $\quad$ $P \geq 1$ $\quad$ **false**

$C \leq 0$ $\quad$ $C \geq 1$ $\quad$ $W \leq 1$ $\quad$ $W \geq 2$

Solution (2,0,0) = 30 $\quad$ $P \leq 1$ $\quad$ $P \geq 2$ $\quad$ **false**

$W \leq 1$ $\quad$ $W \geq 2$ $\quad$ **Solution (1,1,1) = 32**

$C \leq 2$ $\quad$ $C \geq 3$ $\quad$ **false** $\quad$ **Worse than best sol**

false $\quad$ $W \leq 0$ $\quad$ $W \geq 1$

false $\quad$ **false**

Finite Constraint Domains Summary

- CSPs form an important class of problems
- Solving of CSPs is essentially based on backtracking search
- Reduce the search useing consistency methods
  - node, arc, bound, generalized
- Optimization is based on repeated solving or using a real optimizer to guide the search