APLAI
Constraints on reals

Gerda Janssens
Departement computerwetenschappen
A01.26
http://www.cs.kuleuven.be/~gerda/APLAI

Constraint (Logic) Programming

1. Top-down search with passive constraints (Prolog)
2. Delaying automatically (arithmetic constraints) using the suspend library
3. Constraint propagation in ECLiPSe
   the symbolic domain library (`sd`)
   the interval constraints library (`ic`)
4. Top-down search witch active constraints, also variable and value ordering heuristics
5. Optimisation with active constraints
6. Constraints on reals (`locate` built-in)
7. Linear constraints over continuous and integer variables (`eplex` library)

6. Constraints on reals

1. Three classes of problems
2. Constraint propagation
3. Splitting one domain
4. Search
5. The built-in search procedure `locate`
6. Shaving using `squash`
7. Optimisation on continuous variables

No longer finite domains

- Implications?
- On search space?

- Discretising the domain, then again …
- But
  - domain size can become huge
  - May eliminate solutions

6.1 Three classes of problems

a) Continuous variables all dependent completely on finite domain variables
b) Finite search space, but with mathematical constraints which yield real number values
c) Infinite search space, possibly infinitely many solutions, which are then represented by means of formulas, `0.0 < X < 1.0`

a. Dependent continuous variables

- Cylindrical compost heap using a length of chicken wire.
- The volume of compost should be at least 2 cubic meters.
- The wire comes in lengths (L) of 2,3,4, and 5 meters.
- And width (W) 50,100, 200 centimeters
- W? L? Continuous vars?

With lib(ic) and lib(branch_and_bound)

```
compost_1(W,L,V) :-
   W :: [50, 100, 200],
   L :: 2..5,
   V $>= 2.0,
   V $= (W/100) * (L^2 /( 4 * pi)),        % height * area
    minimize(labeling([W,L]), V).
[eclipse 1]: compost_1(W, L, V).
Found a solution with cost
   2.5464790894703251__2.546479089470326
Found no solution with cost 2.5464790894703251 ..
   1.546479089470326

W = 200
L = 4
V = 2.5464790894703251__2.546479089470326
There are 6 delayed goals.
Yes (0.00s cpu)
% 4 =:= 3.9999999999999996__4.0000000000000009
```


Interval arithmetic for reals: bounded real

`V = 2.5464790894703251__2.546479089470326`

Either floating point numbers: finite precision; rounding errors…
Or intervals: the true value of the real is guaranteed to be in the interval;
 arithmetic deals with the bounds of the interval
 if interval is too wide: probably ill-conditioned problem or poorly computed

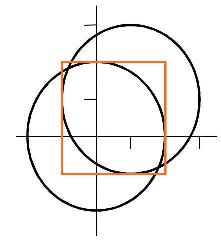
```
?- X is sqrt(2).
X = 1.4142135623730951
Yes (0.00s cpu)
?- X is sqrt(breal(2)).          % 2 converted to a bounded real:breal/1
X = 1.4142135623730949__1.4142135623730954
Yes (0.00s cpu)

?- X is float(1) / 10, Y is X + X + X + X + X + X + X + X + X + X.
X = 0.1
Y = 0.99999999999999989
Yes (0.00s cpu)
?- X is breal(1) / 10, Y is X + X + X + X + X + X + X + X + X + X.
X = 0.099999999999999992__0.1
Y = 0.99999999999999978__1.0000000000000007
Yes (0.00s cpu)
```

b. Finite search space and continuous variables

- Equation for a circle ray r, center point (x1,y1)
  (x – x1) ^2 + (y – y1)^2 = r ^2

```
circles(X,Y) :-
  4 $= X^2 + Y^2,            % center (0,0)
  4 $= (X-1)^2  +  (Y-1)^2,  % center (1,1)
  X $>= Y.             % to the right of the line X= Y
```


Intersection of 2 circles



```
circles(X,Y) :-
      4 $= X^2 + Y^2,
      4 $= (X-1)^2  +  (Y-1)^2,
      X $>= Y.

[eclipse 1]: circles(X, Y).
X = X{-1.0000000000000004 .. 2.0000000000000004}
Y = Y{-1.0000000000000004 .. 2.0000000000000004}
There are 13 delayed goals.
Yes (0.00s cpu)

% the two circles intersect at the points X and Y
% solution values of X and Y???
```


c. Infinitely many solutions

- Compost heap with L and W continuous variables

```
compost_2(W,L,V) :-
   W :: 50.0 .. 200.0,
   L :: 2.0 ..5.0 ,
   V $= 2.0,
   V $= (W/100) * (L^2 /( 4 * pi)).
   % minimize…labeling…. : problem!!!!

[eclipse 2]:  compost_2(W, L, V).
W = W{100.53096491487334 .. 200.0}
L = L{3.5449077018110313 .. 5.0}
V = 2.0__2.0
There are 11 delayed goals.
Yes (0.00s cpu)
```


6.2 Constraint propagation for real variables

- IC's general constraints (`$=/2`, `$=</2`, etc.) work for:
  real variables
  integer variables
  a mix of both
- Propagation is performed using
  safe arithmetic: rounding outward
- Integrality preserved where possible

Propagation behaviour (I)

- For integers, just like integer constraints:

```
?- [X, Y] :: 1..5, X $>= Y + 1.
X = X{2 .. 5}, Y = Y{1 .. 4}
Delayed goals:
        ic : (-(X{2 .. 5}) + Y{1 .. 4} =< -1)

?- [X, Y] :: 1..5, X $>= Y + 1, Y $>= 3.
X = X{[4, 5]}, Y = Y{[3, 4]}
Delayed goals:
        ic : (-(X{[4, 5]}) + Y{[3, 4]} =< -1)

?- [X, Y] :: 1..5, X $>= Y + 1, Y $>= 4.
X = 5, Y = 4
```


Propagation behaviour (II)

- For reals, uses safe arithmetic:

```
?- [X, Y] :: 1.0..5.0, X $>= Y + 1.
X = X{1.9999999999999998 .. 5.0}
Y = Y{1.0 .. 4.0000000000000009}
Delayed goals:
    ic : (-(X{1.9999999999999998 .. 5.0})
         + Y{1.0 .. 4.0000000000000009} =< -1)

?- [X, Y] :: 1.0..5.0, X $>= Y + 1, Y $>= 3.
X = X{3.9999999999999996 .. 5.0}
Y = Y{3.0 .. 4.0000000000000009}
Delayed goals:
    ic : (-(X{3.9999999999999996 .. 5.0})
         + Y{3.0 .. 4.0000000000000009} =< -1)
```


Propagation behaviour (III)

- Variables don’t usually end up ground:

```
?- [X, Y] :: 1.0..5.0, X $>= Y + 1, Y $>= 4.
X = X{4.9999999999999991 .. 5.0}
Y = Y{4.0 .. 4.0000000000000009}
Delayed goals:
    ic : (-(X{4.9999999999999991 .. 5.0})
         + Y{4.0 .. 4.0000000000000009} =< -1
Yes
```


Closer look at the intervals

- Bounds are ??
- In double precision (64 bits)   IEEE standard
  sign bit + 11 bits for exponent +
  52 bits for fraction from normalized mantissa ($1.b_{13}..b_{64}$)
- Upto 16 significant digits

```
[eclipse 4]:  X $= 1 + 2^(-52) .
X = 1.0000000000000002__1.0000 0000 0000 0002
[eclipse 5]: X $= 1 + 2^(-53) .
X = 1.0__1.0
[eclipse 4]:  X $= 1/3 .
X = 0.3333 3333 3333 3333 1__0.33333333333333337
```

Inconsistent pair of constraints : ic detects it by shaving

```
incons(X, Y, Diff) :-
  [X,Y] :: 0.0 .. 10.0,
  X $>= Y + Diff,        % X $>= X + 2*Diff
  Y $>= X + Diff.
?- incons(X, Y, 0.0001).
No (0.17s cpu)
```

Ic considers the individual constraints: first, X $>= Y + Diff

X = X{0.0099999999999997868 .. 10.0}

Y = Y{0.0 .. 9.99}

Then, Y $> X + Diff and Y = Y{0.019999999999999574 .. 9.99}

Next, X $>= Y + Diff and X = X{0.029999999999999361 .. 9.99}

Repeatedly shaving of the domain bounds of X/Y by the amount of DIff

Inconsistent pair of constraints : ic detects it by shaving

```
incons(X, Y, Diff) :-
  [X,Y] :: 0.0 .. 10.0,
  X $>= Y + Diff,
  Y $>= X + Diff.
?- incons(X, Y, 0.0001).
No (0.17s cpu)
?- incons(X, Y, 1e-5).
No (1.81s cpu)
?- incons(X, Y, 1e-6).
No (18.19s cpu)
?- incons(X, Y, 1e-7).
No (356.72s cpu)
```

```
?- incons(X, Y, 1e-8).
X = X{0.0 .. 10.0}
Y = Y{0.0 .. 10.0}
There are 2 delayed goals.
Yes (0.00s cpu)

ic : (Y{0.0 .. 10.0} -
  X{0.0 .. 10.0} =< -1e-8)
ic : (-(Y{0.0 .. 10.0}) +
  X{0.0 .. 10.0} =< -1e-8)

% propagation threshold
```


Propagation threshold

- A small floating-point number: bounds updates to non-integer variables are only performed if the change in the bounds exceeds this threshold
- The default threshold is 1e-8.
- Limiting the amount of propagation is important for efficiency.
- A higher threshold speeds up computations, but reduces precision and may in the extreme case prevent the system from being able to locate individual solutions.

6.3 Splitting one domain

```
?- circles(X, Y).        % both sols in the intervals
X = X{-1.0000000000000004 .. 2.0000000000000004}
Y = Y{-1.0000000000000004 .. 2.0000000000000004}
There are 13 delayed goals.
Yes (0.00s cpu)
?- circles(X, Y), (X $>= 1.5 ; X $< 1.5).
X = X{1.8228756488546369 .. 1.8228756603552694}
Y = Y{-0.82287567032498 .. -0.82287564484820042}
There are 12 delayed goals.
Yes (0.00s cpu, solution 1, maybe more)
No (0.03s cpu)  % for X< 1.5 no solution
```

Changed propagation threshold

```
?- circles(X, Y), (X $>= 1.5 ; X $< 1.5).
X = X{1.8228756488546369 .. 1.8228756603552694}
Y = Y{-0.82287567032498 .. -0.82287564484820042}

?- set_threshold(1e-12).
Yes (0.00s cpu)

?- circles(X, Y), (X $>= 1.5 ; X $=< 1.5).
X = X{1.8228756555318164 .. 1.8228756555339904}
Y = Y{-0.822875655533355 .. -0.82287565553152953}
There are 12 delayed goals.
Yes (0.00s cpu, solution 1, maybe more)
```


6.4 Search for continuous variables

- Domain of a variable is an interval
- At each search node: narrow one interval
- Complete search : union of subintervals covers the input interval
- When to stop: precision is reached, namely the maximum allowed width of any interval on completion of the search
- The search failure is sound (if all subnodes lead to a failure, this is a real failure)
- If search stops without failure, ???
  No guarantee (see incons/3 example)

Conditional solution for a problem on reals

- Consists of 2 components:
  - A real interval for each variable. Each interval is smaller than the given precision
  - A set of constraints in the form of delayed goals. These constraints are neither solved nor unsatisfiable when considered on the final real intervals.

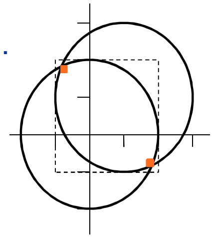
Solving real constraints

- IC provides two methods for solving real constraints
- `locate/2,3` good when there are a finite number of discrete solutions
  Works by splitting domains
- `squash/3` good for refining bounds on a continuous feasible region
  Works by trying to prove parts of domains infeasible, shaving

6.5 The built-in search predicate locate

- To search over continuous variables: splits the domains of specified variables until their sizes fall within the specified precision (e.g. `1e-5`).

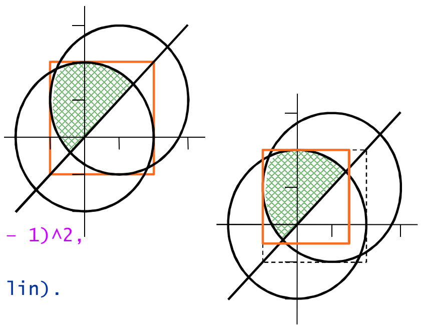
```
?- circles(X, Y), locate([X, Y], 1e-5).
X = X{1.8228756448482004 ..
      1.8228756603552694}
Y = Y{-0.82287566035526938 ..
      -0.82287564484820042}
There are 12 delayed goals.
Yes (0.00s cpu)
```



6.6 Shaving

- Other approach is needed when even for narrow intervals constraint propagation is unable to recognize infeasibility.
- Too many alternative conditional solutions are returned

Using squash



```
?- 4 $>= X^2 + Y^2,
   4 $>= (X - 1)^2 + (Y - 1)^2,
   Y $>= X,
   squash([X, Y], 1e-5, lin).

X = X{-1.000000000000002 .. 1.4142135999632603}
Y = Y{-0.41421359996326107 .. 2.0000000000000013}

There are 13 delayed goals.
Yes
```


Reimer problem

```
reimer(X,Y,Z) :-
      [X,Y,Z] :: -1.0 .. 1.0,
      X^2 -Y^2 + Z^2 $= 0.5,
      X^3 -Y^3 + Z^3 $= 0.5,
      X^4 -Y^4 + Z^4 $= 0.5.
?- reimer(X, Y, Z), locate([X, Y, Z], 1.0).
X = X{0.86122659100561871 .. 0.88401867762042985}
Y = Y{0.49164137443380324 .. 0.57684813462065521}
Z = Z{-0.3017325460737818 .. 0.10314973700795005}
There are 21 delayed goals.
Yes (0.02s cpu, solution 1, maybe more)
…
X = X{-1.0 .. -0.5}
Y = Y{-1.0 .. -0.10314973700795016}
Z = Z{0.10314973700795005 .. 1.0}
There are 21 delayed goals.
Yes (0.05s cpu, solution 6)
```

Reimer solutions (with locate)

Search precision	
1.0	6 solutions
0.01	8
1e-5	13
1e-8	11
1e-10	10
1e-12	11
1e-14	24

- Search precision=final interval width
- Wide interval: several sols in 1 interval
- Decreasing the interval width, increases number of solutions
- From, 1e-10, conditional solutions are detected to be infeasible.
- Many narrow intervals are returned in the vicinity of the actual solution.

Reimer problem: add shaving too ???

```
?- reimer(X, Y, Z), locate([X,Y,Z],1.0), squash([X,Y,Z],
   1e-15,lin).

X = X{0.876864256865334 .. 0.87686425686533553}
Y = Y{0.551378646680729 .. 0.55137864668073377}
Z = Z{-0.18742328309865935 .. -0.1874232830986538}
There are 21 delayed goals.
Yes (0.95s cpu, solution 1, maybe more)
…
X = X{-0.18742328309865894 .. -0.18742328309865275}
Y = Y{0.55137864668072978 .. 0.55137864668073366}
Z = Z{0.8768642568653342 .. 0.87686425686533553}
There are 21 delayed goals.
Yes (8.89s cpu, solution 4, maybe more)
No (8.91s cpu)

% at most 4!!! solutions for precisions up to 1e-10
% with 1e-14 again 16 conditional solutions left
```


Shaving is a special form of constraint propagation

- A variable is constrained to take a value – or to lie in a narrow interval – near its (upper or lower) bound, and the results of this constraint are propagated to the other variables to determine whether the problem is still feasible. If not, the domain of the original variable is reduced.
- Shaving is a (polynomial) alternative to the (exponential) interval splitting.

Mortgage problem (continuous vars)

```
% Loan (the loan)
% Payment: the fixed monthly amount paid off
% Interest: the fixed, but compounded, monthly
  interest rate
% Time : the time in months taken to pay off a loan
  (integer)

mortgage(Loan, _Payment,_Interest,0) :- Loan $= 0.
mortgage(Loan, Payment,Interest,Time):-
  Loan $> 0,
  Time $>= 1,
  NewLoan $= Loan *(1+Interest) - Payment,
  NewTime $= Time - 1,
  mortgage(NewLoan, Payment,Interest,NewTime).

?- mortgage(X,700,0.01,126).
X = 50019.564804291353__50019.56480429232
```

Mortgage queries: benefit of shaving

```
?- Pay :: 0.0 .. 200000.0, mortgage(200000, Pay, 0.01,
   360).
Pay = Pay{0.0 .. 200000.0}
There are 718 delayed goals.

?- Pay :: 0.0 .. 200000.0, mortgage(200000, Pay, 0.01,
   360), squash([Pay],1e-5,log).
Pay = Pay{2057.1348846043343 .. 2057.2332029240038}
There are 360 delayed goals.
Yes (9.94s cpu, solution 1, maybe more)
```


Mortgage queries: benefit of shaving

```
?- Interest :: 0.0 .. 1.0, mortgage(60000, 1500,
   Interest, 60).
Interest = Interest{0.0 .. 0.5}
There are 257 delayed goals.

?- Interest :: 0.0 .. 1.0,
   mortgage(60000, 1500, Interest, 60),
   squash([Interest], 1e-5, log).
Interest = Interest{0.0143890380859375 ..
   0.014396667480468751}
There are 237 delayed goals.
Yes (2156.83s cpu, solution 1, maybe more)
```


Summary

- Interval bounds: double precision
- Propagation threshold: which changes to the bounds are taken into account and trigger propagation 1e-8
- Precision (of search): the final width of the intervals 1e-5

  (as argument of locate and squash predicates)

6.7 Optimization on continuous variables

- Branch and bound relies on the fact that the search procedure instantiates the variable constrained to the cost function.
- Can no longer be guaranteed: variables have intervals smaller than the chosen precision of minimize/2.

Optimization example

```
cons(X,Y) :-
   [X,Y] :: -100.0.. 100.0,
   2*X + 2*Y^2 $< 10, X + 3*Y $=< 5, 2*X - Y $=< 10 .

?- cons(X, Y).
X = X{-100.0 .. 5.0}
Y = Y{-10.2469507659596 .. 10.2469507659596}
There are 5 delayed goals.

?- cons(X, Y), locate([X, Y], 0.01).
X = X{-0.0057595876495268421 .. 0.0022624905715885792}
Y = Y{-0.0027417303066677359 .. 0.0045673154896543151}
There are 2 delayed goals.
Yes (0.00s cpu, solution 1, maybe more)
X = X{0.0022624905715885792 .. 0.010284568792704}
Y = Y{-0.0027417303066677359 .. 0.0045673154896543151}
.... Many solutions
```


Optimization example

```
cons(X,Y) :-
   [X,Y] :: -100.0.. 100.0,
   2*X + 2*Y^2 $< 10, X + 3*Y $=< 5, 2*X - Y $=< 10 .

opt_1(Query, Expr, Min) :-
   OptExpr $= eval(Expr),
   minimize((Query, get_min(OptExpr,Min)), Min).

?- cons(X, Y), opt_1(locate([X, Y], 0.01), 6 - X, Min).
Found a solution with cost 5.9977375094284113 …
Found a solution with cost 1.9412953187340669
Found no solution with cost -1.0Inf .. 0.94129531873406691
X = X{3.9941057914745426 .. 4.0587046812659331}
Y = Y{-0.0012435228262297066 .. 0.0044940668232824887}
Min = 1.9412953187340669

% there is no solution with a value less than 0.941....
% but there is a gap between 0.941 and 1.941(best conditional
   solution) : due to the step of 1.0 in branch and bound
% we can only conclude that there is no minimum below 0.941
```


Tailoring the improvement in b&b

```
opt_2(Query, Expr, Improvement, Min) :-
   OptExpr $= eval(Expr),
   bb_min((Query, get_min(OptExpr,Min)), Min,
            bb_options{delta:Improvement}).

?- cons(X, Y), opt_2(locate([X, Y], 0.01), 6 - X, 0.5,
   Min).
Found a solution with cost 5.9977375094284113
…
Found a solution with cost 1.2349623277551629
Found no solution with cost -1.0Inf .. 0.7349623277551629
X = X{4.6891966915344225 .. 4.7650376722448371}
Y = Y{-0.0012435228262297066 .. 0.0044940668232824887}
Min = 1.2349623277551629
There are 4 delayed goals.
Yes (8.14s cpu)
```

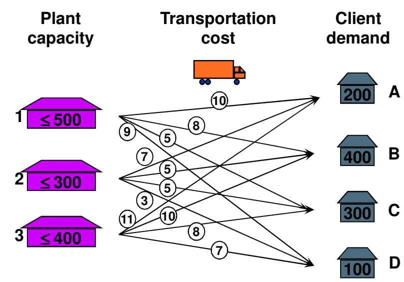

7. Linear constraints over continuous and integer variables

1. LP/MIP and the `eplex` library
2. Fundamentals
3. Solving satisfiability problems using `explex`
4. Repeated solver waking
5. The transportation problem
6. The linear facility location problem
7. The non-linear facility location problem

7.1 LP/MIP and lib(eplex)

- An interface between ECLiPSe and an external LP/MIP solver (with efficient methods from operation research)
  - COIN-OR Open Solver Interface: currently to COIN's CLP/CBC and SYMPHONY/CLP (Open Source)
  - XPRESS-MP, a product by Dash Associates
  - CPLEX, a product by ILOG SA /IBM
- Motivation to have this link:
  - Use state-of-the-art linear programming tools
  - Use ECLiPSe for modelling
  - Build hybrid solvers
- Implementation
  - Tight coupling, using subroutine libraries

Mathematical Programming



Objective Function

```
minimize
10 A1 + 7 A2 + 11 A3 +
 8 B1 + 5 B2 + 10 B3 +
 5 C1 + 5 C2 +  8 C3 +
 9 D1 + 3 D2 +  7 D3

subject to
 A1 + A2 + A3 = 200
 B1 + B2 + B3 = 400
 C1 + C2 + C3 = 300
 D1 + D2 + D3 = 100

A1 + B1 + C1 + D1 ≤ 500
A2 + B2 + C2 + D2 ≤ 300
A3 + B3 + C3 + D3 ≤ 400
```

Leads to a matrix:
•Each row is one constraint
•Each column is one variable

Constraints

LP/MIP – A brief overview

- Linear Programming:
  - Optimisation problem formulated using linear constraints and a linear objective function
  - An efficient algebraic method developed to solve such problems (Simplex, 1947)
  - Interior point (barrier) method
- Mixed Integer Programming:
  - LP problems where some of the variables are constrained to be integer
  - The problem of solving linear constraints over integer variables is in general NP-complete
  - No known 'optimal' method: generally use simplex/barrier with branch and bound search for optimal
- Quadratic Programming
  - Quadratic objective function, linear constraints
  - Barrier method

Possible Uses From Within ECLiPSe

- Black-box

  If problem is suitable for LP/QP/MIP as a whole

  But: Only one solution, non-incremental, rounding errors
- Incrementality and multiple instances

  Results explicitly retrieved

  Multiple independent subproblems
- Hybrid

  Solvers work on (possibly overlapping) subproblems

  Programmed cooperation strategy

  Data-driven triggering

General Usage

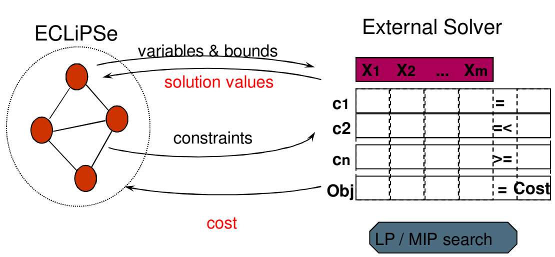
- Loading `:- lib(eplex).`
  Loads interface with default solver (OSI CLP/CBC on most machines)
  OSI: open solver interface
  CLP: linear solver `:- lib(eplex_osi_clpcbc).`
  CBC: mixed integer programming
- Differences between the underlying solvers are largely hidden
- More details:
  http://www.eclipseclp.org/doc/libman/libman052.html

Transportation Problem (black box solving)

```
:- lib(eplex).
main(Cost, Vars) :-
        Vars = [A1, A2, A3, B1, B2, B3, C1, C2, C3, D1, D2, D3],
        Vars :: 0.0..1.0Inf,

        A1 + A2 + A3 $= 200,
        B1 + B2 + B3 $= 400,
        C1 + C2 + C3 $= 300,
        D1 + D2 + D3 $= 100,

        A1 + B1 + C1 + D1 $=< 500,
        A2 + B2 + C2 + D2 $=< 300,
        A3 + B3 + C3 + D3 $=< 400,

        optimize(min(
            10*A1 + 7*A2 + 11*A3 +
             8*B1 + 5*B2 + 10*B3 +
             5*C1 + 5*C2 +  8*C3 +
             9*D1 + 3*D2 +  7*D3), Cost).
```


Overview: Eplex as black-box solver

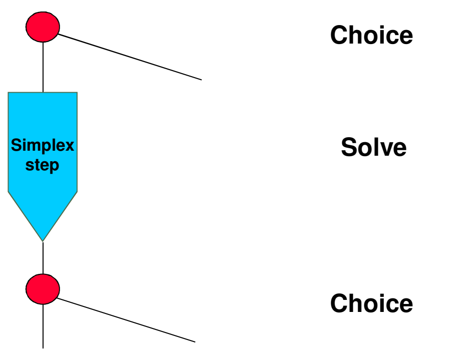
- CLP modelling / LP or MIP solving
  - 1. Do modelling and transformations in ECLiPSe
  - 2. Load model into external solver(CLP/CBC) and solve using LP or MIP solver
  - 3. Pass cost and solution values back to ECLiPSe



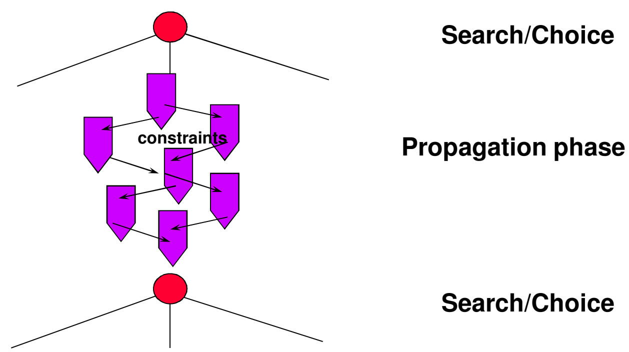
Problems with Integrating MP Solvers

- Solvable constraint class severely limited
  - Pure LP/QP/MIP/MIQP
- Numerical problems
  - The external solvers use floating-point numbers, subject to rounding errors
  - Other ECLiPSe solvers are usually precise (intervals or rational numbers)
  - Instantiating the ECLiPSe variables to inexact values may cause other implementations of the constraints to fail!
- Lack of incrementality
  - Cannot find alternative optima
  - Cannot add constraints or variables
- MIP search is a straightjacket
  - A complex search process inside the black box
  - Hard to control (via dozens of parameters)
  - Hard to add problem-specific heuristics
- Unsuitable for solver cooperation!

Control Flow in Classical MIP Solver

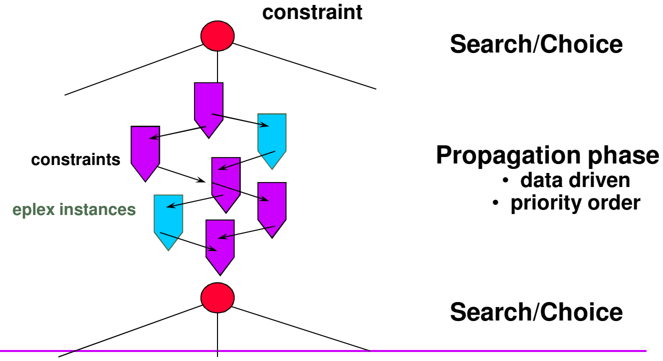


Control Flow with Constraint Propagation

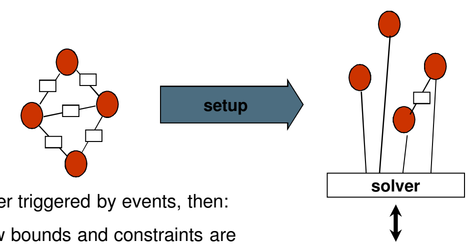


Control Flow with Constraint Propagation

Eplex instance demon can be integrated similar to a global constraint

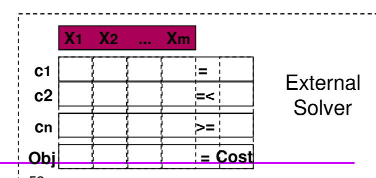


Eplex solver as compound constraint



Solver triggered by events, then:

- new bounds and constraints are sent to the external solver
- external solver is run
- cost bound (or failure) is exported
- solution values are exported and ECLiPSe variables annotated



7.2 Going from ic to eplex

```
:- lib(ic).
incons(W,X,Y,Z) :-
  W + X + Y + Z $>= 10,
  W + X + Y $= 5,
  Z $=< 4 .              % thus W + X + Y $>= 6

?- incons(W, X, Y, Z).
W = W{-1.0Inf .. 1.0Inf}
X = X{-1.0Inf .. 1.0Inf}
Y = Y{-1.0Inf .. 1.0Inf}
Z = Z{-1.0Inf .. 4.0}
There are 2 delayed goals.
Yes (0.00s cpu)
%  no detection of the inconsistency!!!
```


Using locate

```
:- lib(ic).
incons(W,X,Y,Z) :-
  W + X + Y + Z $>= 10,
  W + X + Y $= 5,
  Z $=< 4 .              % thus W + X + Y $>= 6

?- incons(W, X, Y, Z), locate([W,X,Y,Z],0.01).
W = W{27.896542342314088 .. 28.910297761270662}
X = X{28.288147181201772 .. 29.301656701329183}
Y = Y{-52.211954462599849 .. -51.184689523515857}
Z = Z{3.9072936443861837 .. 4.0}
There are 2 delayed goals.
% 28 + 29 - 51 >= 6 but also 28 + 29 - 52 = 5
% many conditional solutions
%  no detection of the inconsistency!!!
```


eplex library: opening the black box

```
:- lib(eplex).
?- eplex_solver_setup(min(0)),   % (1)
   incons(W, X, Y, Z),
   eplex_solve(_Opt).            % (2)
No (0.00s cpu)
```

- (1) Make an instance of the eplex library which is going to min(Cost)/max(Cost)
- (2) Launching the eplex solver: argument returns the optimal value of the cost function

Eplex supports

- Variable declarations and linear constraints
- No longer support for
  - Integer versions such as X #=< Y
  - strict inequalities and disequality, X $<Y, X $\= Y
  - Boolean constraints
  - reified constraints

Common Arithmetic Solver Interface

	`$::/2` `$=/2,` `=:=/2` `$>=/2,` `>=/2` `$=</2,` `=</2`	`$>/2,` `>/2` `$</2,` `</2`	`$\=/2` `=\=/2`	`::/2`	`#::/2` `#= /2` `#>=/2,` `#>/2` `#=</2,` `#</2`	`#\=/2`	`integers/1`	`reals/1`
suspend	✓	✓	✓	✓	✓	✓	✓	✓
ic	✓	✓	✓	✓	✓	✓	✓	✓
eplex	✓			✓			✓	✓
std arith	✓	✓	✓					

Solution for the problem when Z $=< 6

```
?- eplex_solver_setup(min(O)),
     incons6(W, X, Y, Z), eplex_solve(_).
W = W{-1.7976931348623157e+308 ..
  1.7976931348623157e+308 @ 5.0}
X = X{-1.7976931348623157e+308 ..
  1.7976931348623157e+308 @ 0.0}
Y = Y{-1.7976931348623157e+308 ..
  1.7976931348623157e+308 @ 0.0}
Z = Z{-1.7976931348623157e+308 .. 6.0 @ 6.0}
Yes (0.02s cpu)

% solution after @ is just one of the many
```


Solution in eplex solver

- Eplex does not provide any means of computing alternative solutions, for example by backtracking
- You can extract the current solution

```
?- eplex_solver_setup(min(O)), incons6(W, X, Y, Z),
   eplex_solve(_),
   eplex_var_get(W, typed_solution, W_Val).
W = W{-1.7976931348623157e+308 ..
   1.7976931348623157e+308 @ 5.0}
X = X{-1.7976931348623157e+308 ..
   1.7976931348623157e+308 @ 0.0}
Y = Y{-1.7976931348623157e+308 ..
   1.7976931348623157e+308 @ 0.0}
Z = Z{-1.7976931348623157e+308 .. 6.0 @ 6.0}
W_Val = 5.0
Yes (0.00s cpu)
```


Incrementality

- Solver setup
  `eplex_solver_setup(+Objective)`
- Run solver once
  `eplex_solve(-Cost)`
  Computes a solution with `Cost` but does not instantiate variables
- Access solution information
  Status, solutions, reduced costs, dual values, slack, statistics, etc.
  `eplex_get(+What, -Value)`
  `eplex_var_get(+Var, +What, -Value)`
- New constraints and variables can now be added and the problem re-solved!

LP example (without integers)

```
lptest(W,X) :-
   eplex_solver_setup(min(X)),
   [W,X] :: 0 .. 10,
   2*W + X $= 5,
   eplex_solve(_).

?- lptest(W, X), eplex_get(vars, Vars), eplex_get(typed_solution,
   Vals).
W = W{0.0 .. 10.0 @ 2.5}
X = X{0.0 .. 10.0 @ 0.0}
Vars = ''(X{0.0 .. 10.0 @ 0.0}, W{0.0 .. 10.0 @ 2.5})
Vals = ''(0.0, 2.5)
Yes (0.00s cpu)
?- lptest(W, X), eplex_get(vars, Vars), eplex_get(typed_solution,
   Vals), Vars = Vals.
W = 2.5
X = 0.0
Vars = ''(0.0, 2.5)
Vals = ''(0.0, 2.5)
Yes (0.00s cpu)
```


LP example (without integers)

```
return_solution :-
   eplex_get(vars,Vars),
   eplex_get(typed_solution, Vals),
   Var = Vals.

lptest(W,X) :-
   eplex_solver_setup(min(X)),
   [W,X] :: 0 .. 10,
   2*W + X $= 5,
   eplex_solve(_),
   return_solution.

?- lptest(W, X).
W = 2.5
X = 0.0
% Eplex warning: Imposing integer bounds on
  variable(s) [W, X] for eplex instance eplex does
  not impose integer type.
```


Opening the Black Box

- Code for optimize/2:

```
optimize(Objective, Cost) :-
  eplex_solver_setup(Objective),
  eplex_solve(Cost),
  eplex_get(vars, VarVector),
  eplex_get(typed_solution, SolutionVector),
  VarVector = SolutionVector.
```


Eplex also supports integrality constraints

- Mixed integer programming problems
- Integrality constraint needs to be specified by `integers(List)`
- In general MIP needs a form of search, as typically first a solution with reals is found and then MIP/ECLiPSe will have to search the ones with integers….

MIP version : W is an integer

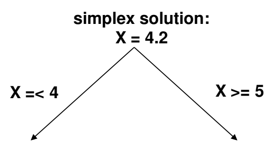
```
miptest(W,X) :- eplex_solver_setup(min(X)),
   [W,X] :: 0.0 .. 10.0, integers([W]),
   2*W + X $= 5 ,
   eplex_solve(_),
   return_solution.

?- miptest(W, X).
W = 2
X = 1.0
Yes (0.02s cpu)

% Minimising min(W+X)    W = 2 and X = 1.0   and minimum 3
% Minimising  min(W + sqrt(X)) ????
```


Internal (branch and bound) search method used for MIP

1. Solve the linear constraints in eplex, as if the variables were continuous
2. From the solution select an integer variable, W, whose eplex value val is non-integral. If no such variable exists, the problem is solved.
3. Let valint+ be the smallest integer larger than val and valint- the largest integer smaller than val. The valint+ = valint- + 1.
   Make a choicepoint: add either W >= valint+ or W =< valint-. Return to 1. On backtracking try the alternative.
4. To find the optimal solution, use the usual branch and bound.

Traditional MIP branching

- At each node:
  - Solve the continuous relaxation (ignoring integrality) with simplex
  - Select an integer variable with fractional simplex solution
  - Try two alternatives, with bounds forced to integers



- Eventually, all variables will be narrowed to integers (if a solution exists)

Repeated solver waking

- How we exploit interaction ic and eplex??
- CP: event driven (bounds) propagation
- LP/MIP: can detect inconsistency but must be invoked!!! (eplex_solve!!).
- When to re-invoke LP/MIP??
  - Addition of new linear constraints
  - new, tighter variable bounds that exclude the solution value (deviating_bounds).
  - Instantiation of variables to a value different from its solution value (deviating_inst).
- As trigger conditions for eplex_solver_setup (laziest: deviating_inst)
- Moreover, optimum cost is computed by eplex and can be used as lower bound on Cost variable

LP/MIP as a propagation constraint

- It reacts to e.g. bound changes
- Imposes a new bound on cost variable

- Cheap interval-propagation constraints are mixed with LP/MIP-solver constraints

Triggering the solver automatically

```
eplex_solver_setup(+Objective, ?Cost, +Options, +TriggerModes)
```

`Objective`

min(Expr) or max(Expr)

`Cost`

variable - it does not get instantiated, but only <u>bounded</u> by the solution cost.

Triggering the solver automatically

```
eplex_solver_setup(+Objective, ?Cost, +Options, +TriggerModes)
```

`TriggerModes`

- **`inst`** - if a variable was instantiated
- **`deviating_inst`** - if a variable was instantiated to a value that differs more than a tolerance from its LP-solution
- **`bounds`** - if a variable bound was changed !!!!
- **`deviating_bounds`** - if a variable bound was changed such that its LP-solution was excluded by more than a tolerance.
- **`new_constraint`** - when a new constraint appears
- **`<module>:<cond>`** – waking condition defined by other solver, e.g. ic:min
- **`trigger(Atom)`** - explicit triggering

Using trigger conditions

```
?- X+Y $>= 3, X-Y $= 0,
      eplex_solver_setup(min(X), X, [], [inst]).
X=X{1.4999999 .. 1.7976931348623157e+308 @ 1.5}
Y=Y{-1.7976931348623157e+308 .. 1.7976931348623157e+308 @ 1.5}
Delayed goals:
        lp_demon(…)
Yes.

?- X+Y $>= 3, X-Y $= 0,
   eplex_solver_setup(min(X), X, [], [inst]), X=2.0.
X =2.0
Y =Y{-1.7976931348623157e+308 .. 1.7976931348623157e+308 @ 2.0}
Delayed goals:
        lp_demon(…)
Yes.
```

Doing MIP in ECLiPSe (a better way)

```
:- lib(eplex), lib(branch_and_bound).
main :- <setup constraints>
        IntVars = <variables that should take integral values>,
        Objective = <objective function>,
        ...
        Objective $= CostVar,
        eplex_solver_setup(min(Objective), CostVar, [], [bounds]),
        ...
        minimize(mip_search(IntVars), CostVar).

mip_search(IntVars) :-
        ...
        % for each X violating the integrality condition
        eplex_var_get(X, solution, RelaxedSol),
        Split is floor(RelaxedSol),
        ( X $=< Split) ; X $>= Split+1 ),                  % choice
        ...
        % simplex re-solving triggered by bounds change
        % cost bound automatically applied
```

- No explicit simplex calls, no auxiliary parameters to search procedure
- Standard CLP search scheme - other choices/constraints can be added now

Example: interaction

```
nonlinear(W,X,Cost) :-
   Cost :: 0.0 .. inf,
   eplex_solver_setup(min(Cost), Cost, [], [bounds,new_constraint]),
   % Cost: optimal value of Arg1 of explex_solver_setup
   % wake up whenever a variable bound is tightened or
   %                  new linear constraint
   [W,X] :: 0 .. 10,
   integers([W]),
   2*W + X $= 5,
   add_linear_constraint(X,SqrtX),
   Cost $= W + SqrtX,
   minimize(nlsearch(X,SqrtX), Cost),
   return_solution,
   eplex_cleanup.             % just stop the linear solver

add_linear_constraint(X,SqrtX) :-    % linear approximation of sqrt
   eplex_var_get_bounds(X,Min,Max),
   SqrtMin is sqrt(Min), SqrtMax is sqrt(Max),
   SqrtX $>= SqrtMin,  SqrtX $=< SqrtMax,
   SqrtX* SqrtMin $=< X, SqrtX*SqrtMax $>= X.
```


Example Cont.

```
nlsearch(X,SqrtX) :-
   eplex_var_get(X,solution,Val),
   eplex_var_get(SqrtX,solution,SqrtVal),
   abs(sqrt(Val)- SqrtVal) =< 1e-5,!,
   return_solution.

nlsearch(X,SqrtX) :-
   eplex_var_get(X,solution,Val),
   (X $>= Val; X $=< Val),                  % make a choice!!!
   add_linear_constraint(X,SqrtX),
   nlsearch(X,SqrtX).

?- nonlinear(W, X, Cost).
Found a solution with cost 2.23606797749979
Found no solution with cost 1.5811387300841897 .. 1.2360679774997898
W = 0
X = 5.0
Cost = 2.23606797749979
Yes (0.17s cpu)
```


The transportation problem

- Well-known COPs
  - The transportation problem
  - The linear facility location problem
  - The non-linear facility location problem

Problem statement

- Warehouses have to serve customers with a given set of demands.

```
cust_count(4).                % 4 customers
warehouse_count(3).           % 3 warehouses
capacities([600,500,600]).
demands([400,200,200,300]).

% tc_ij  cost for transporting to customer c_i from
  warehouse w_j
transport_costs( []([](5,4,1),   %costs to deliver to c1
          [](3,3,2),
          [](4,2,6),
          [](2,4,4))).
```


Transport predicate

```
% supplies_ij is the amount of goods transported from w_j to c_i

transport(Supplies, Cost) :-
   eplex_solver_setup(min(Cost)),
   init_vars(Supplies),
   supply_cons(Supplies),
   cost_expr(Supplies,Cost),
   eplex_solve(Cost),
   return_solution.

init_vars(Supplies) :-
   cust_count(CustCt),
   warehouse_count(WCt),
   dim(Supplies,[CustCt,WCt]),
   ( foreachelem(S,Supplies)
   do
      0 $=< S
   ).
```


Supply constraints

```
supply_cons(Supplies) :-
   capacity_cons(Supplies), demand_cons(Supplies).

capacity_cons(Supplies):-    % capacity of a warehouse
   capacities(Capas),
   ( count(WHouse, 1,_),
     foreach(Cap, Capas),
     param(Supplies)
   do
     cust_count(CustCt),
     Cap $>= sum(Supplies[1..CustCt,WHouse])
   ).

demand_cons(Supplies):-      % demands by a customer
   demands(Demands),
   ( count(Cust, 1,_),
     foreach(Demand, Demands),
     param(Supplies)
   do
     warehouse_count(WCt),
     sum(Supplies[Cust,1..WCt]) $>= Demand
   ).
```


Cost constraints

```
cost_expr(Supplies,CostExpr):-
   transport_expr(Supplies, TranspExpr),
   CostExpr $= TranspExpr.

transport_expr(Supplies,sum(TranspExpr)):-
   transport_costs(TrCosts),
   ( foreachelem(TCost, TrCosts),
     foreachelem(Qty, Supplies),
     foreach(TExpr, TranspExpr)
   do
     TExpr = TCost*Qty
   ).

?- transport(S, C).
S = []([](0.0, 0.0, 400.0), [](0.0, 0.0, 200.0), [](0.0,
   199.99999999999997, 0.0), [](300.0, 0.0, 0.0))
C = 1800.0
Yes (0.00s cpu)
```

Facility location problem

- Warehouse: to be or not to be
- Each warehouse has setup costs
- Now a NP-hard problem

```
setup_costs([100,800,400]).
transport(Supplies, OpenWhs, Cost) :-
  eplex_solver_setup(min(Cost)),
  init_vars(Supplies, OpenWhs),
  supply_cons(Supplies, OpenWhs),
  cost_expr(Supplies, OpenWhs, Cost),
  eplex_solve(Cost),
  return_solution.
```


Adapted constraints

```
init_openwhs(OpenWhs):-
  warehouse_count(WCt),
  length(OpenWhs,WCt),
  OpenWhs :: 0.0.. 1.0,
  integers(OpenWhs).

capacity_cons(Supplies,OpenWhs):-
  capacities(Capas),
  ( count(WHouse, 1,_),
    foreach(OpenWh, OpenWhs),
    foreach(Cap, Capas),
    param(Supplies)
  do
    cust_count(CustCt),
    Cap*OpenWh $>= sum(Supplies[1..CustCt,WHouse])
  ).
%big M constraint   M*Bool $>= Expr
```


Adapted constraints (Cont)

```
cost_expr(Supplies,OpenWhs,CostExpr):-
  setup_expr(OpenWhs, SetupExpr),
  transport_expr(Supplies, TranspExpr),
  CostExpr $= SetupExpr + TranspExpr.

setup_expr(OpenWhs, sum(SetupExpr)):-
  setup_costs(SetupCosts),
      ( foreach(OpenWh, OpenWhs),
        foreach(SetupCost, SetupCosts),
        foreach(SExpr, SetupExpr)
      do
        SExpr = OpenWh * SetupCost
      ).

?- transport(S, W, C).
S = [](
[](0.0, 0.0, 400.0), [](0.0, 0.0, 200.0), [](200.0,
  0.0, 0.0), [](300.0, 0.0, 0.0))
W = [1, 0, 1]    C = 2700.0
Yes (0.02s cpu)
```


Non-linear facility location problem

- Now the capacity of each warehouse is a decision variable `Capacities`

`Cap*OpenWh $>= sum(Supplies[1..CustCt,WHouse])`

- Cost of setting up a warehouse: fixed cost + extra cost proportional to the square root of the capacity of the warehouse
- We combine again branch_and_bound with `eplex` just as we did for nonlinear

Non-linear facility predicate

```
nltransport(Capacities,OpenWhs, Cost) :-
  Cost :: 0.0 .. inf,
  eplex_solver_setup(min(Cost), Cost, [],
  [deviating_bounds,new_constraint]),
  % trigger when a bound of a variable becomes tightened
  % so as to exclude its linear optimal value from the
  % previous activation of the linear solver
  init_vars(Supplies,OpenWhs,Capacities,SqrtCaps ),
  supply_cons(Supplies,OpenWhs,Capacities),
  cost_expr(Supplies,OpenWhs,SqrtCaps,Cost),
  minimize(nlsearch(Capacities,SqrtCaps), Cost),
  return_solution,
  eplex_cleanup.
```


Init_capacities

```
init_vars(Supplies,OpenWhs,Capacities,SqrtCaps) :-
   init_capacities(Capacities,SqrtCaps),
…
init_capacities(Capacities,SqrtCaps):-
   warehouse_count(WCt),
   length(Capacities,WCt),
   length(SqrtCaps,WCt),
   total_demand(CapUPB),
   SqrtCapUPB is sqrt(CapUPB),
   Capacities :: 0 .. CapUPB,
   SqrtCaps :: 0.0 .. SqrtCapUPB,
   ( foreach(Cap,Capacities),
     foreach(SqrtCap,SqrtCaps)
   do
     add_linear_cons(Cap,SqrtCap)
   ).

total_demand(CapUPB):-
  demands(Demands),CapUPB is sum(Demands).
```


Capacity constraints

```
capacity_cons(Supplies,OpenWhs, Capacities):-
   total_demand(BigM),
   ( count(WHouse, 1,_),
     foreach(OpenWh, OpenWhs),
     foreach(Cap, Capacities),
     param(Supplies,BigM)
   do
     cust_count(CustCt),
     % sufficient capacity in the WHouse to meet
      % the supply commitment:     Cap*OpenWh not linear
     Cap         $>= sum(Supplies[1..CustCt,WHouse]),

      % non-zero supply is only possible from an open
   warehouse
     BigM*OpenWh $>= sum(Supplies[1..CustCt,WHouse])
   ).
```


Setup Constraint

```
setup_expr(OpenWhs,SqrtCaps, sum(SetupExpr)):-
   setup_costs(SetupCosts),
   ( foreach(OpenWh, OpenWhs),
     foreach(SqrtCap, SqrtCaps),
     foreach(SetupCost, SetupCosts),
     foreach(SExpr, SetupExpr)
        do
           SExpr = (OpenWh+SqrtCap*0.1) * SetupCost
                            %0.1 due to experiments..
        ).
```

Non-linear search

```
nlsearch(Capacities, SqrtCaps) :-
   ( too_different(Capacities, SqrtCaps, Cap, SqrtCap) ->
       makechoice(Cap),

       add_linear_cons(Cap, SqrtCap),
       nlsearch(Capacities, SqrtCaps)
   ;
       return_solution
   ).

too_different(Capacities, SqrtCaps, Cap, SqrtCap) :-
   get_pair(Capacities, SqrtCaps, Cap, SqrtCap),
   eplex_var_get(Cap, solution, CapVal),
   eplex_var_get(SqrtCap, solution, SqrtCapVal),
   abs(sqrt(CapVal)- SqrtCapVal) $>= 1e-5 .

get_pair([X|_],[Y|_],X,Y).
get_pair([_|TX],[_|TY],X,Y):- get_pair(TX,TY,X,Y).

makechoice(X):- eplex_var_get(X,solution,Val), (X $>= Val; X $=< Val).
```


Improved linear approximation

```
add_linear_cons(X,SqrtX) :-
   eplex_var_get_bounds(X,Min,Max),
   SqrtMin is sqrt(Min), SqrtMax is sqrt(Max),
   SqrtX $>= SqrtMin,  SqrtX $=< SqrtMax,
   SqrtX* SqrtMin $=< X, SqrtX*SqrtMax $>= X,
   (SqrtMax-SqrtMin) * (X - Min) $=< (SqrtX - SqrtMin)*
   (Max-Min).

?- nltransport(Caps, OpenWhs, Cost).
Found a solution with cost 3944.3710281197309
Found a solution with cost 3903.4026947632506
Found no solution with cost 3574.3828991755149 ..
   3902.4026947632506

Caps = [500.0, 0.0, 600.00000000000011]
OpenWhs = [1, 0, 1]
Cost = 3903.4026947632506
Yes (2.11s cpu)
```