APLAI
Constraints on reals: Summary

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

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

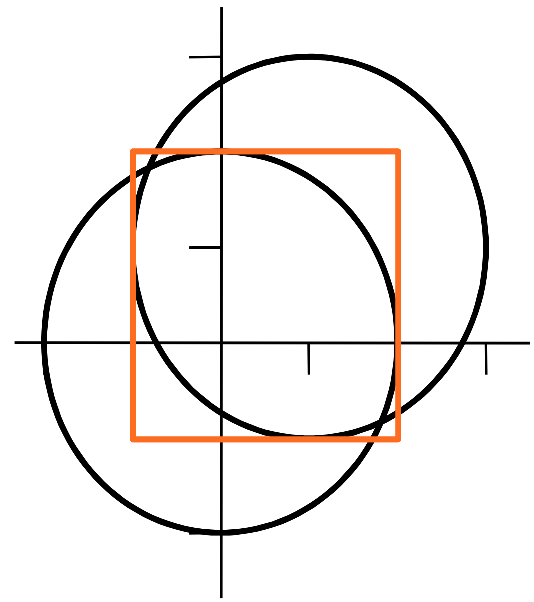
```
?- X is sqrt(2).
X = 1.4142135623730951
Yes (0.00s cpu)
?- X is sqrt(breal(2)).          % 2 converted to a bounded real:breal/1
X = 1.4142135623730949__1.4142135623730954
Yes (0.00s cpu)

?- X is float(1) / 10, Y is X + X + X + X + X + X + X + X + X + X.
X = 0.1
Y = 0.9999999999999989
Yes (0.00s cpu)
?- X is breal(1) / 10, Y is X + X + X + X + X + X + X + X + X + X.
X = 0.09999999999999992__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

- 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

- 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__1.0000000000000002
[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, 0.01).
No
```

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
```

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

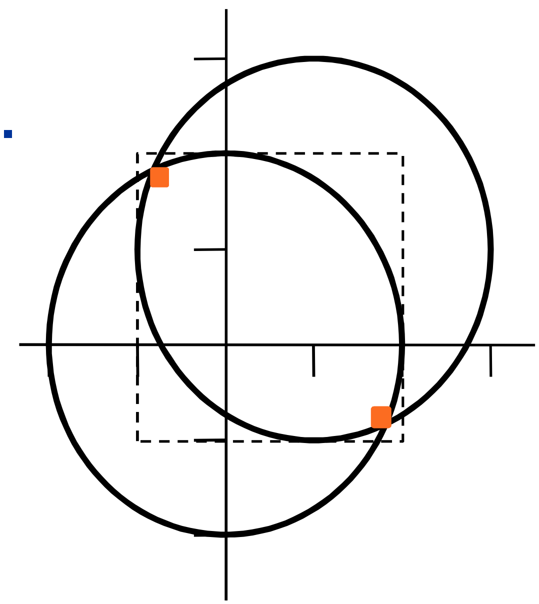
  Works by splitting domains
- `squash/3` good for refining bounds on a continuous feasible region

  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`).

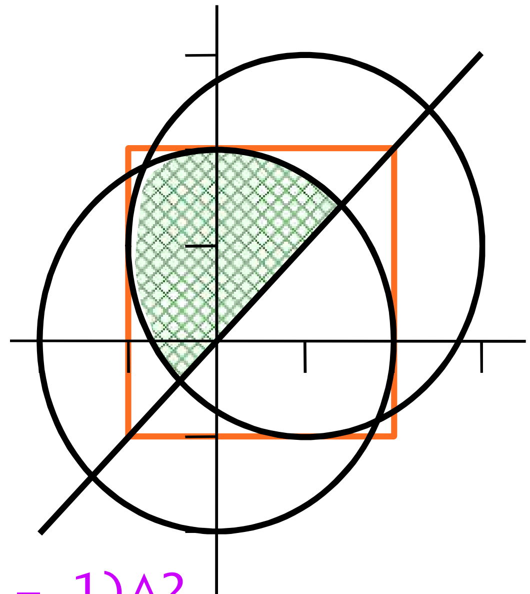
```
?- circles(X, Y), locate([X, Y], 1e-5).
X = X{1.8228756448482004 ..
       1.8228756603552694}
Y = Y{-0.82287566035526938 ..
       -0.82287564484820042}
There are 12 delayed goals.
Yes (0.00s cpu)
```

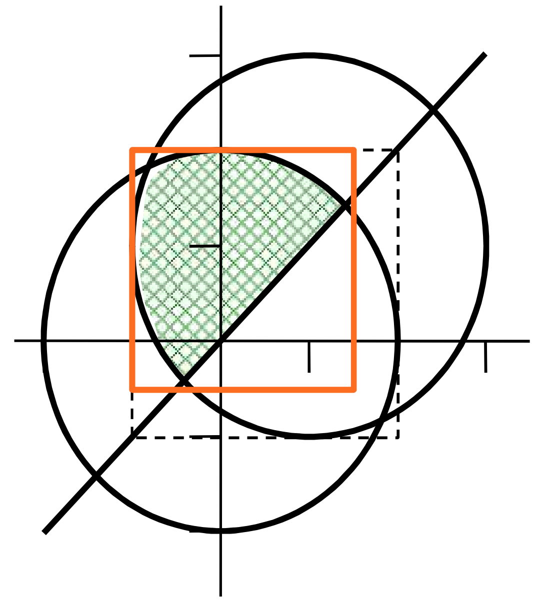


6.6 Shaving

- 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{-1.0000000000000002 .. 1.4142135999632603}
Y = Y{-0.41421359996326107 .. 2.0000000000000013}

There are 13 delayed goals.
Yes
```

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.

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)

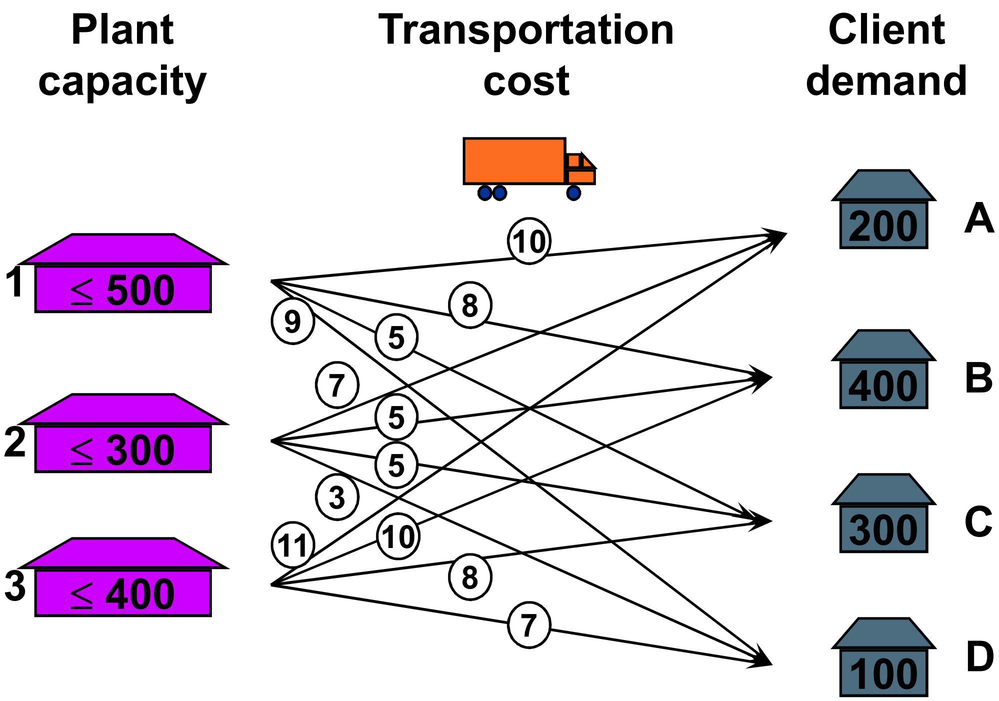
7. Linear constraints over continuous and integer variables

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

Linear programming and ECLiPSe: eplex library

- quite different to other constraint libraries, requiring explicit initialization and solving calls, and returning its solutions in a different way.
- use the linear solver as a constraint propagator
- triggers are used to control when the propagator is invoked.
- solving a non-linear optimization problem by repeated addition of linear constraints

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
```

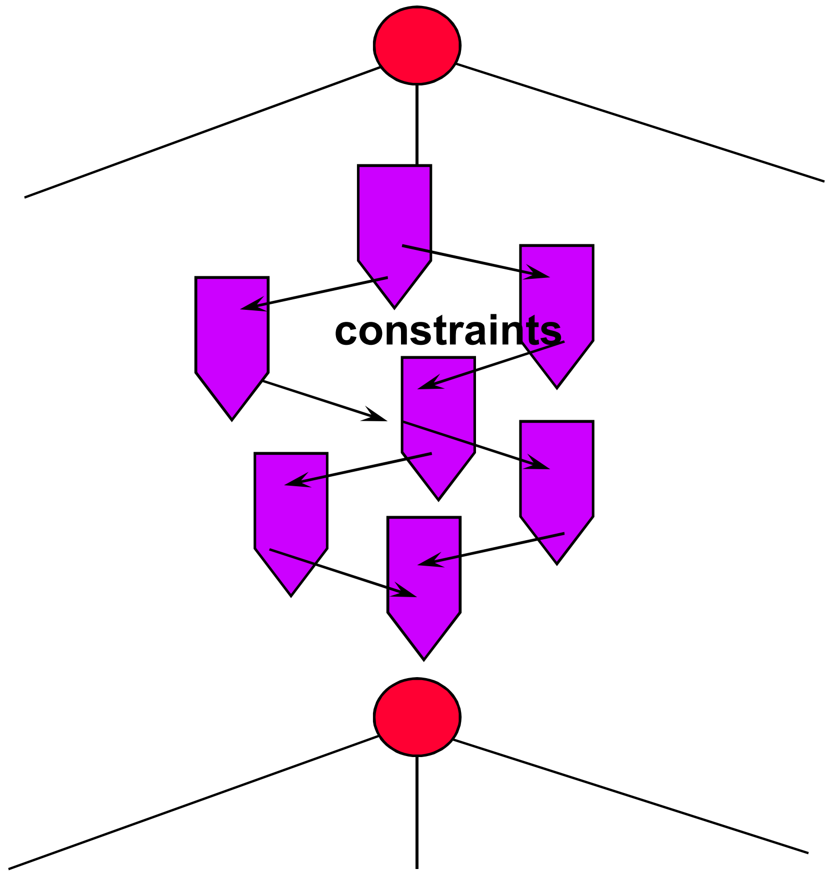
Leads to a matrix:
- **Each row is one constraint**
- **Each column is one variable**

Constraints

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).
```

Control Flow with Constraint Propagation



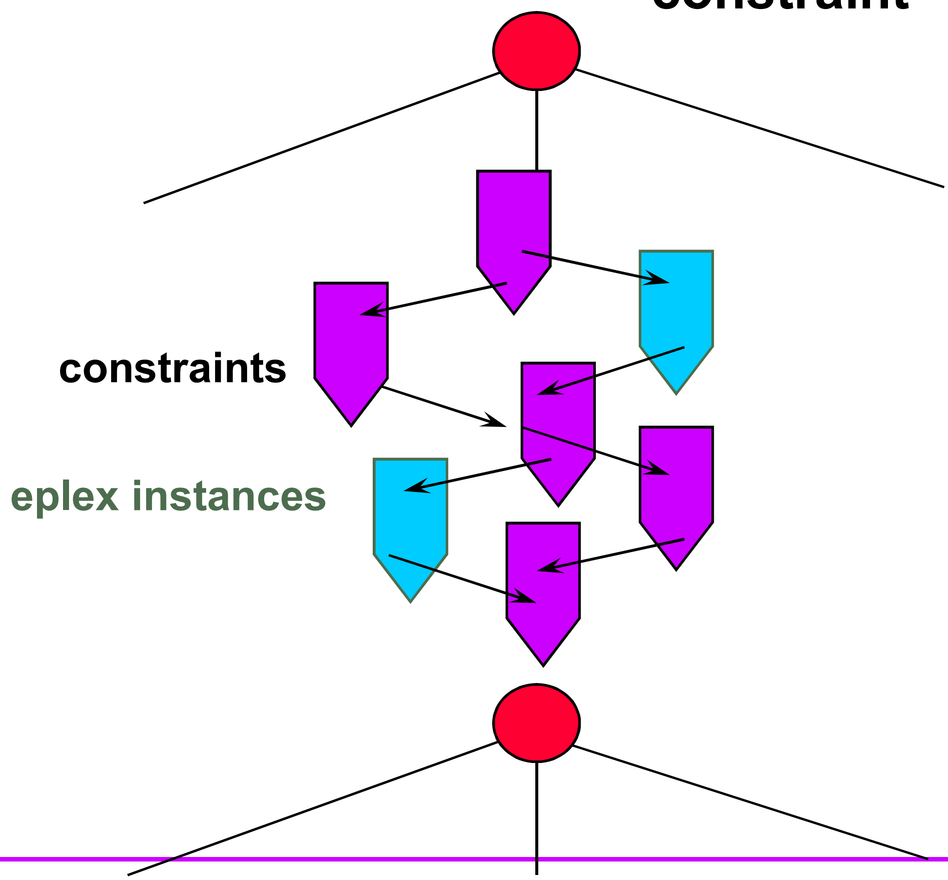
Search/Choice

Propagation phase

Search/Choice

Control Flow with Constraint Propagation

Eplex instance demon can be integrated similar to a global constraint

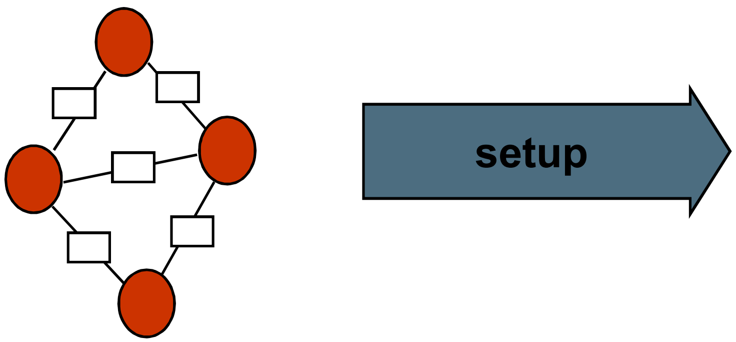


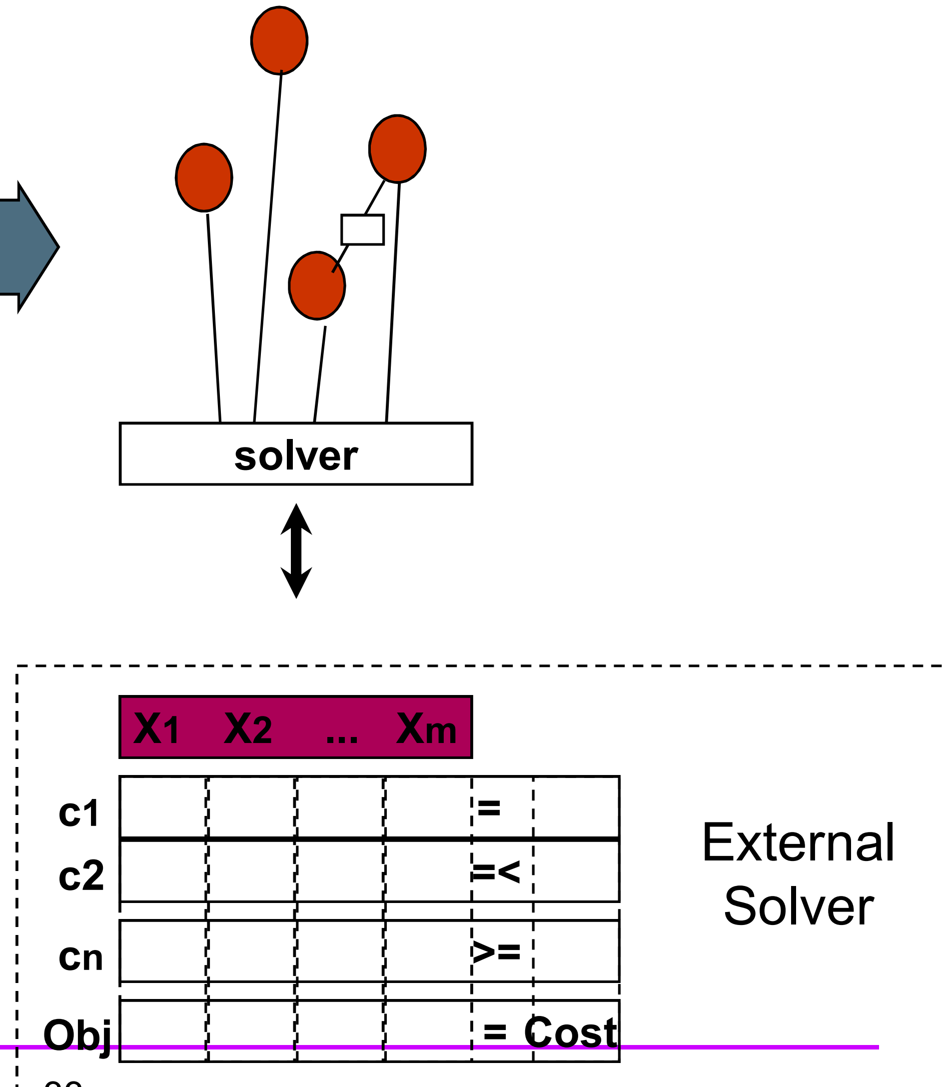
Search/Choice

Propagation phase
- **data driven**
- **priority order**

Search/Choice

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

Opening the Black Box

- Code for optimize/2:

```
:- lib(eplex).
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

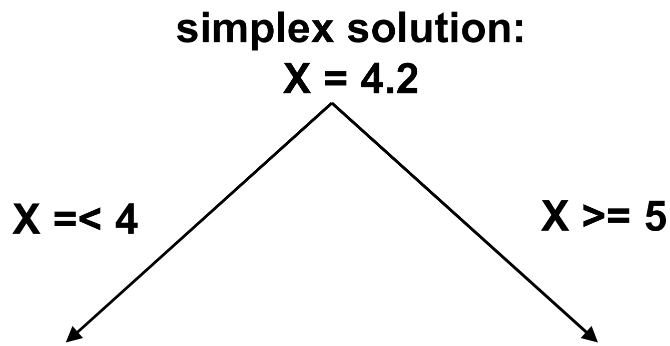
- Mixed integer programming problems
- Integrality constraint needs to be specified by `integers(List)`
- In general MIP needs a form of search, as typically first a solution with reals is found and then MIP/ECLiPSe will have to search the ones with integers….

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)) ????
```

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 bounded 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 .. 1.797693134862315 7e+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.
```

The transportation problem

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