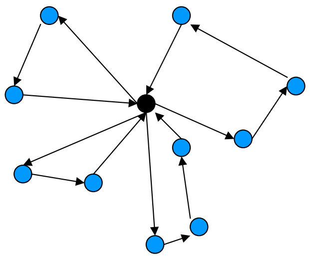
ROUTING: problem definition

- Routing concerns the problem of finding a set of routes to be covered by a set of vehicles visiting a set of cities/customers once starting and ending at one (*n*) depot(s).

- Constraints
  - temporal restrictions:
    - time windows
    - maximal duration of a travel
  - vehicle capacity
  - customer demands
- Optimization Criteria
    - number of vehicles
    - travel cost
    - travel time



ROUTING: problem definition

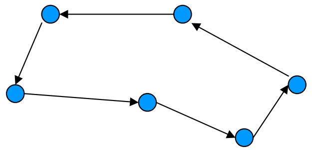
- Routing has been solved within OR community by using
  - Branch & Bound approaches
  - Dynamic Programming
  - Local Search techniques
  - Branch & Cut
  - Column generation

- Routing has been solved within CP community by using
  - Branch & Bound approaches
  - Local Search techniques embedded in CP

- Basic component: Travelling Salesman Problem (TSP) and its time constrained variant.

TSP: problem definition

- TSP is the problem of finding a minimum cost tour covering a set of nodes once.



- No subtours are allowed

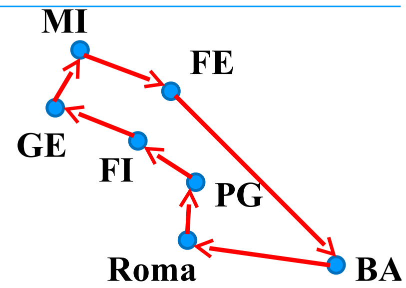
- TSPTW: Time windows are associated to each node. Early arrival is allowed. Late arrival is not permitted

- Even finding a Hamiltonian Circuit (no costs) is NP-complete (if the graph is not complete)

Compute TSP: CLP models

Direct

- *Variables = positions*
- *Values = cities*
- *Constraints:*
  - *alldifferent(Variables)*

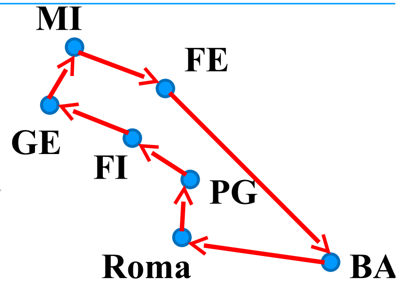


GE	MI	FE	BA	Roma	PG	FI
1	2	3	4	5	6	7

Compute TSP: CLP models

Circuit

- Variables = cities
- Values = successor city
- Constraints:
  - alldifferent(Variables)
  - Variable[i] ≠ i
  - circuit(Variables)

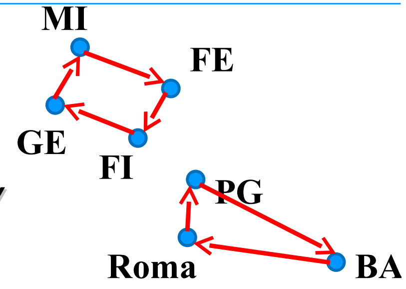


Roma	BA	GE	MI	FE	FI	PG
BA	FE	FI	GE	MI	PG	Roma

Compute TSP: CLP models

Circuit

- Variables = cities
- Values = successor city
- Constraints:
  - alldifferent(Variables)
  - Variable[i] ≠ i
  - ~~circuit(Variables)~~



Roma	FI	GE	MI	FE	BA	PG
BA	FE	FI	GE	MI	PG	Roma

Circuit in ECLiPSe

- Il vincolo `circuit` non è presente nella libreria `fd` di ECLiPSe
- Nelle ultime versioni, è presente nella libreria `ic`
- Ricorda: non è possibile mescolare vincoli della libreria `fd` e della libreria `ic`, perché usano diverse rappresentazioni dei domini

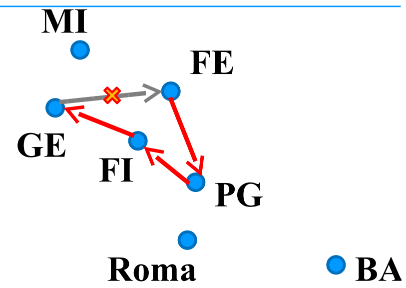
`fd`	`ic`
`fd_global`	`ic_global`
`fd_global_gac`	`ic_global_gac`
`cumulative`	`ic_cumulative`
`edge_finder`	`ic_edge_finder`
`edge_finder3`	`ic_edge_finder3`

circuit constraint

- Può essere implementato semplicemente

`circuit(L) :-`

- cerca valore ground
- segui path fino a una variabile
- elimina dal dominio di questa (ultima) variabile la prima città
- (sospenditi se non hai finito)



	PG	GE			FI	
BA	FE	FI	GE	MI	PG	Roma

Esercizio

- *Dato il seguente CSP:*

```
X1::2..5, X2 :: 3..5, X5::1..4,
circuit([X1,X2,4,1,X5]),
alldifferent([X1,X2,4,1,X5]).
```

- *Si mostri la propagazione effettuata*

TSP: CP model

- Variable associated to each node. The domain of each variable contains possible next nodes to be visited
- N nodes ⟹ N + 1 variables $\mathtt{Next}_i$ (duplicate the depot)
- For all **i**: $\mathtt{Next}_i \neq \mathtt{i}$
- $\mathtt{circuit([Next_0,\ldots Next_n])}$
- $\mathtt{alldifferent([Next_0,\ldots Next_n])}$
- Cost $\mathtt{c_{ij}}$ if $\mathtt{Next_i = j}$
- In some models, we can find the redundant variables **Prev** indicating a node predecessor.

Circuit in CHR

- Per ogni nodo del grafo, impongo un vincolo CHR

  circ(Id,Next)

- In cui
  - **Id** è l'Identificatore del nodo
  - **Next** è una variabile con dominio che rappresenta la città che verrà visitata dopo **Id**

```
circuit(L):-
  length(L,N),

  circuit_loop(L,1,N).

circuit_loop([],_,_).
circuit_loop([H|T],I,N):-
  circ(I,H),
  I1 is I+1,
  circuit_loop(T,I1,N).
```


Circuit in CHR

- Impongo anche un vincolo CHR

  maxlength(N)

  perché mi serve sapere qual è la lunghezza totale del tour
- Ho raggiunto la lunghezza massima?
  - Se sì, impongo che devo tornare all'inizio
  - Se no impongo che non devo tornare all'inizio

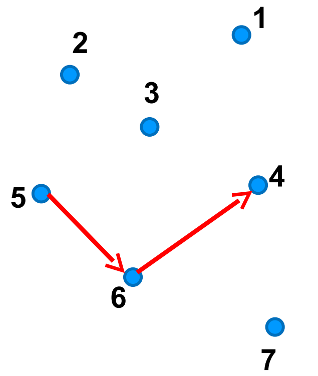
```
circuit(L):-
  length(L,N),
  maxlength(N),
  circuit_loop(L,1,N).

circuit_loop([],_,_).
circuit_loop([H|T],I,N):-
  circ(I,H),
  I1 is I+1,
  circuit_loop(T,I1,N).
```

Circuit in CHR

- I vincoli CHR

`path(Inizio,Fine,Lung)`

- Indicano che è stato creato un percorso dal nodo **`Inizio`** al nodo **`Fine`** di lunghezza **`Lung`** in cui tutte le variabili *Next* sono ground, eccetto eventualmente l'ultima



path(5,$Next_4$,4)

Circuit in CHR

```
initial @ circ(I,Next) ==> Next #\=I, path(I,Next,2).

maxlength @ maxlength(N) \ path(Id,Next,N) <=> true.
```


Circuit in CHR

```
initial @ circ(I,Next) ==> Next #\=I, path(I,Next,2).

maxlength @ maxlength(N) \ path(Id,Next,N) <=> true.

addfirst @ circ(I,Next) \ path(Next,Last,Lpath) <=>
                Lpath1 is Lpath+1,
                Last #\= I,
                path(I,Last,Lpath1).
```

La regola CHR si attiva solo quando i vincoli nel CHR store sono più specifici della testa della regola

Quindi non può unificare due variabili **`Next`** *diverse: se sono già uguali, la regola si attiva, altrimenti no*

In questo caso, siccome **`path`** *viene imposto sempre con il primo argomento ground, la regola addfirst si attiva se c'è nel constraint store* **`circ(I,Next)`** *con* **`Next`** *ground*

Circuit in CHR

```
initial @ circ(I,Next) ==> Next #\=I, path(I,Next,2).

maxlength @ maxlength(N) \ path(Id,Next,N) <=> true.

addfirst @ circ(I,Next) \ path(Next,Last,Lpath) <=>
                Lpath1 is Lpath+1,
                Last #\= I,
                path(I,Last,Lpath1).

addlast @ circ(Last,Next) \ path(First,Last,Lpath) <=>
                Lpath1 is Lpath+1,
                Next #\= First,
                path(First,Next,Lpath1).
```

TSP: code

```
tsp(Data,Next,Costs):-
        remove_arcs_to_self(Next),
        circuit(Next),
        alldifferent(Next),
        create_objective(Next,Costs,Z),
        minimize(labeling(Next),Z).
```

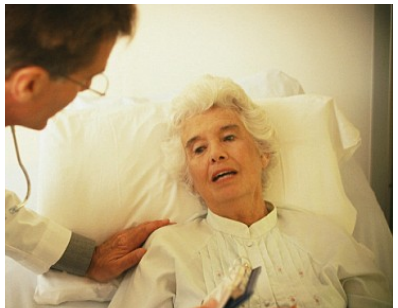
- `circuit`: symbolic constraint that ensures that no subtour is present in the solution.
- `create_objective`: creates `Costs` variables, imposes an `element` constraint between the set of `Next` variables and `Costs` variables, and creates a variable `Z` representing the objective function summing costs corresponding to assignments

TSP: results

- Pure CP implementations: still far from the state of the art OR approaches.
- Integration of OR techniques in CP: better results
  - local search
  - optimal solution of relaxations
    - Lagrangean relaxation
    - Assignment Problem
    - Minimum Spanning Arborescence
  - search strategies based on these relaxations
    - subtour elimination
- Addition of Time Windows in OR approaches requires to re-write major code parts while in CP comes for free.

Home Health Care

- *A patient in a hospital bed is expensive*
- *Patients want to stay at home, if possible.*
  - *Better rehabilitation rates*
- *Some patients (especially elderly) need to be serviced only for few hours a day*

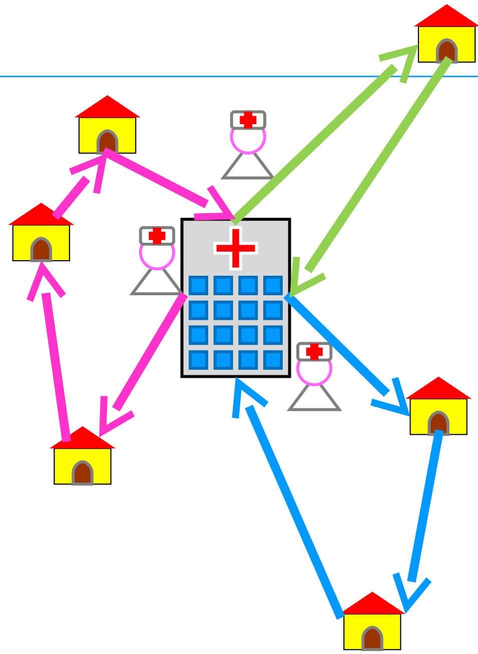


- *Send nurses to patients' homes*

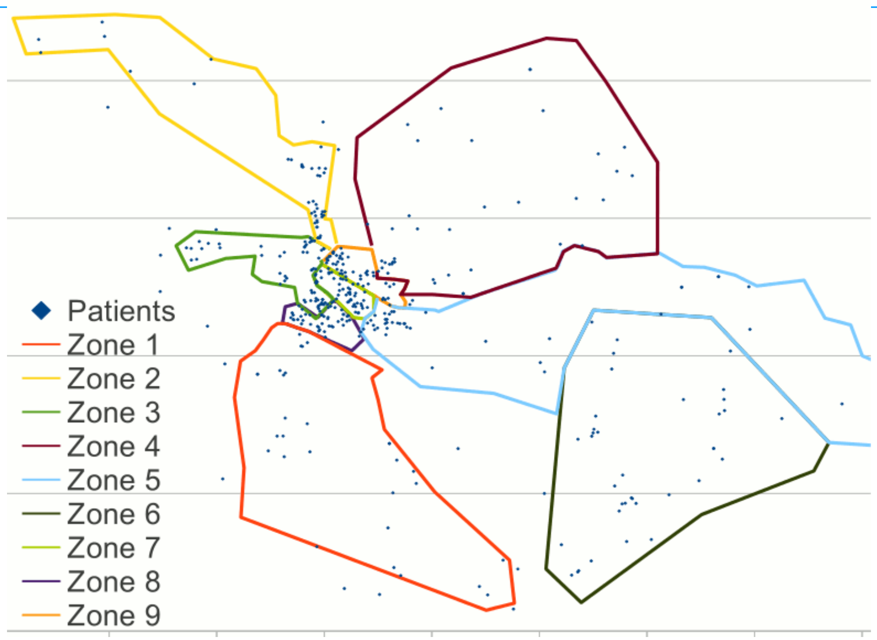
Home Health Care (HHC)

Infermiere

- *Ogni infermiera parte dall'ospedale, visita un certo numero di pazienti e torna all'ospedale*
- *Ad ogni paziente deve essere effettuato uno o più servizi (di durata diversa)*
- *Il tempo di viaggio + tempo di servizio non deve superare le 7 ore al giorno*
- *E` meglio se uno stesso paziente viene visitato sempre dalla stessa infermiera nell'arco della settimana*
- *Si deve cercare di far sì che le infermiere lavorino circa lo stesso numero di ore*



Current solution



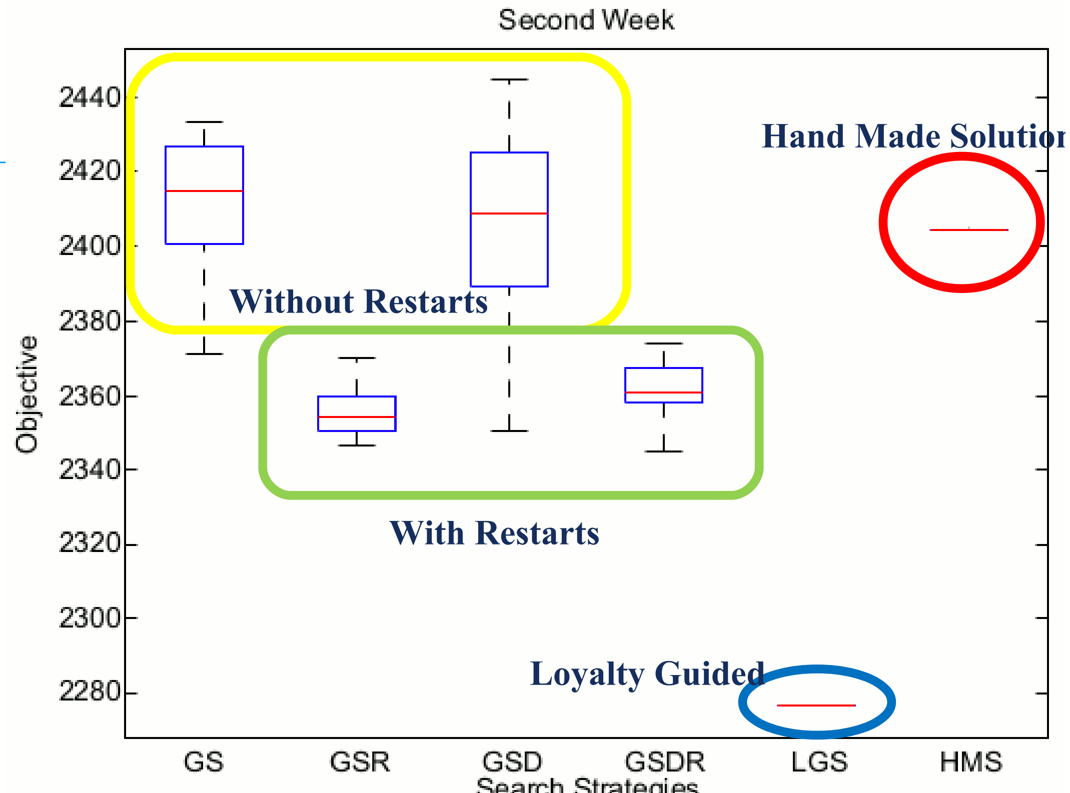
Objectives

- ***Loyalty**: A patient that is visited every day by a different nurse is not satisfied*
- ***Fairness**: nurses complain about disparities. The total WorkLoad (Service Time + Travel Time) should be fair among the nurses*

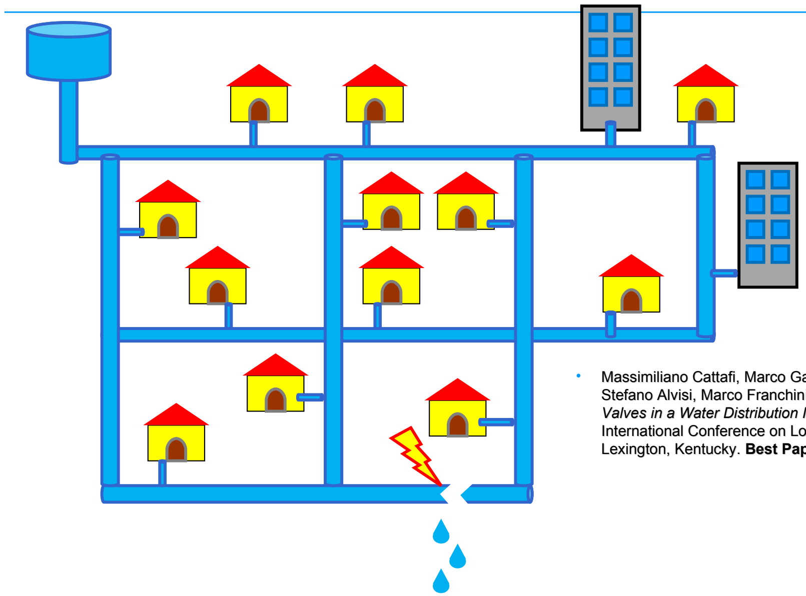
$$\textit{minimize } \max_{nurse} WeekWL_{nurse}$$

Loyalty Guided Search

- *Start with service with longest duration*
- *Given a service s:*
  - *split its domain into two parts:*
    1. *Nurses that are already visiting that patient (in other days)*
    2. *Nurses that do not visit the patient*
  - *Try the set 1 (and on backtracking the set 2)*
  - *Inside one set, try to assign it first to the less occupied nurse*

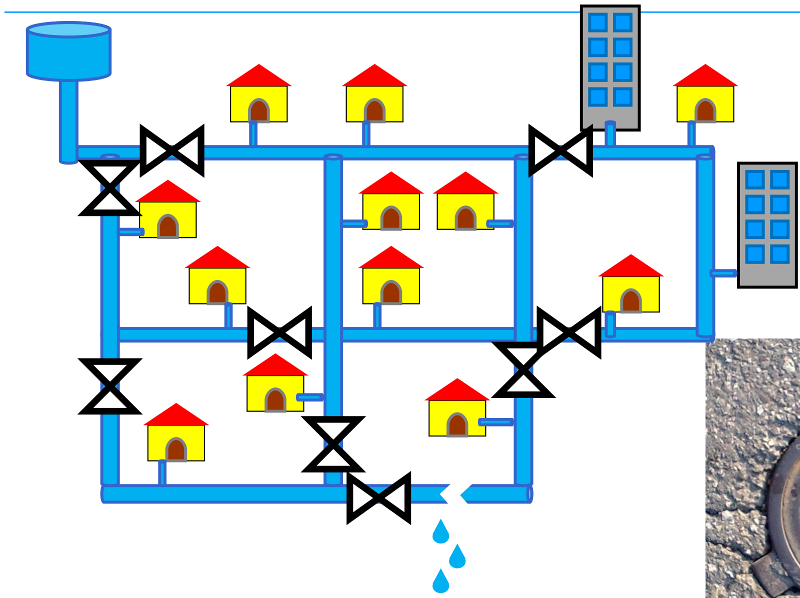


Hydroinformatics

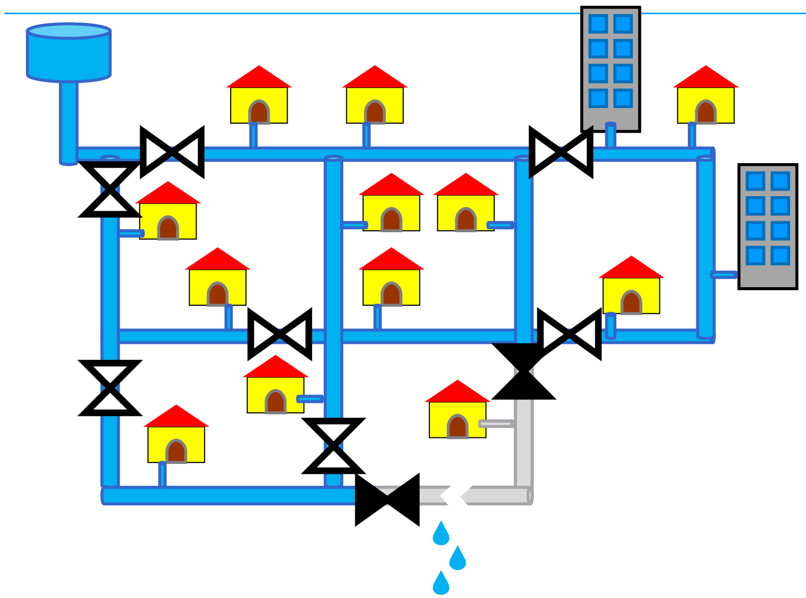


- Massimiliano Cattafi, Marco Gavanelli, Maddalena Nonato, Stefano Alvisi, Marco Franchini, *Optimal Placement of Valves in a Water Distribution Network with CLP(FD)*, International Conference on Logic Programming 2011, Lexington, Kentucky. **Best Paper Award**

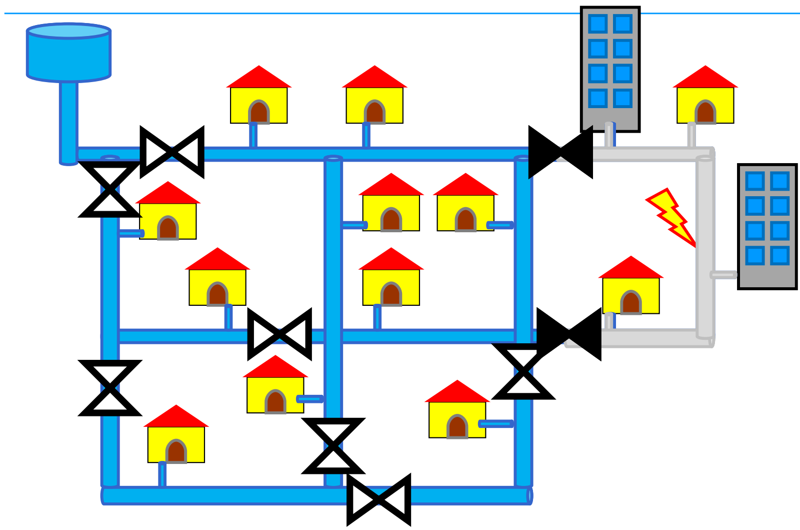
Problem description



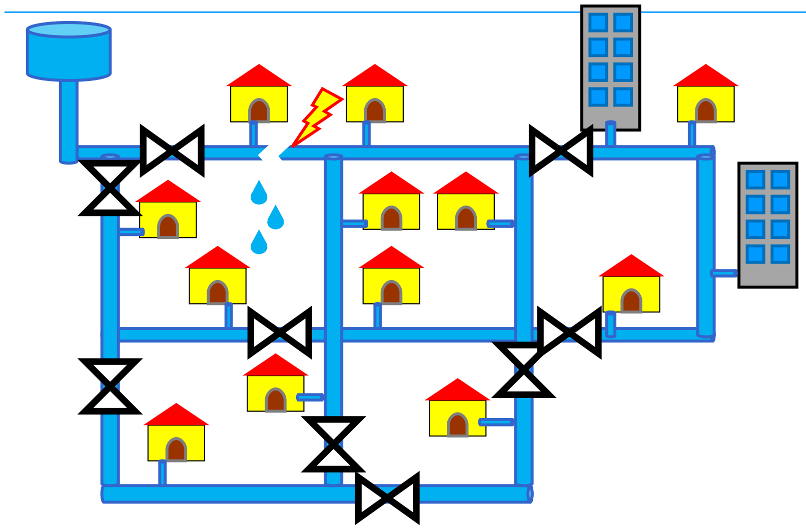
Problem description



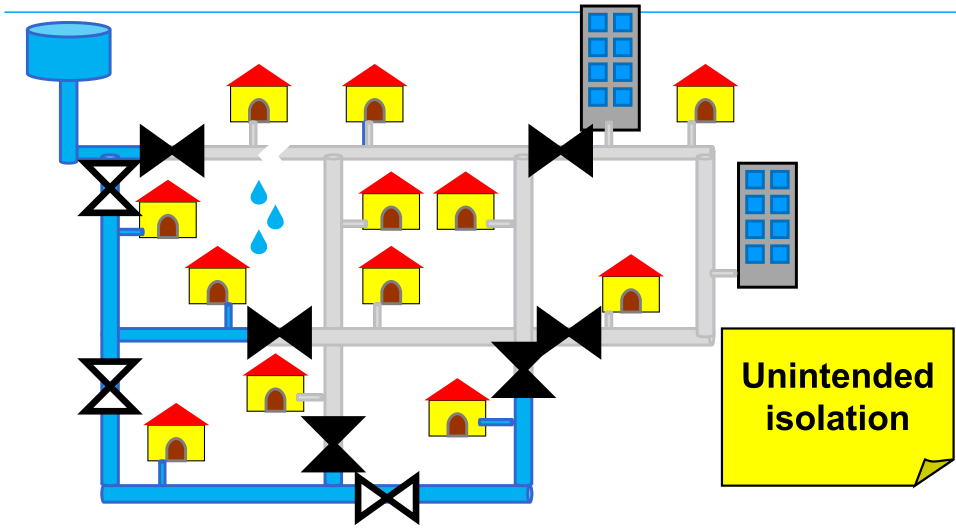
Problem description



Problem description



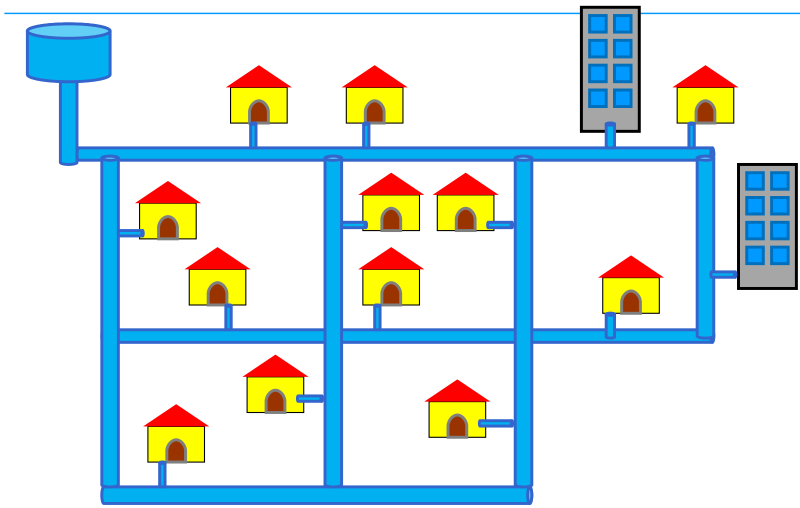
Problem description



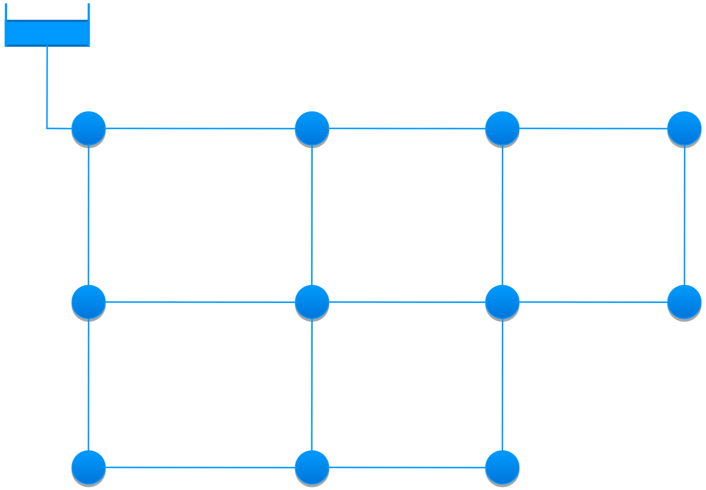
Problem

- *is there a way to place valves such that users' disruption is minimal? (in some sense ...)*
- *place two valves in each pipe, so you can isolate each pipe individually*
- *ok, but valves have a cost*
  - *cost of the valve*
  - *installation cost*
  - *reliability: near a valve the pipe is more fragile, breaks easily*

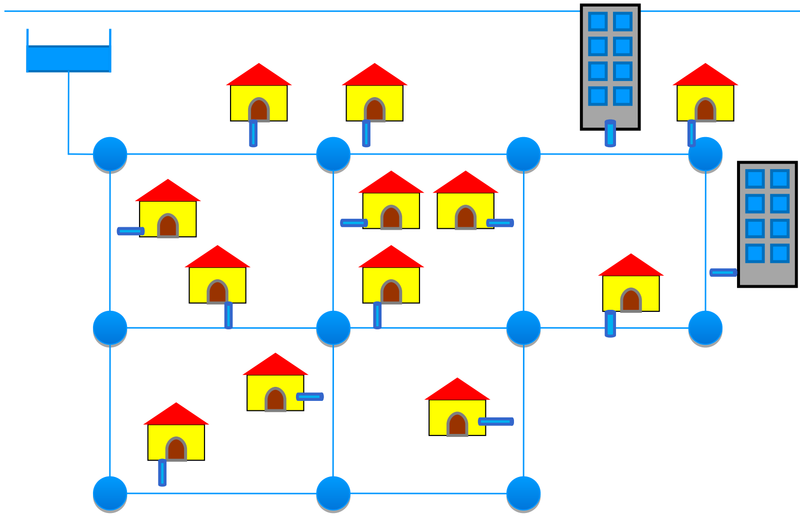
More formally



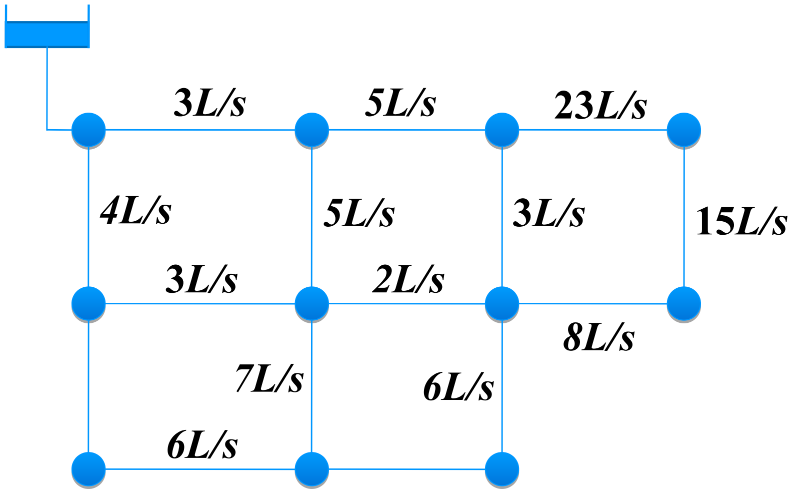
More formally



More formally

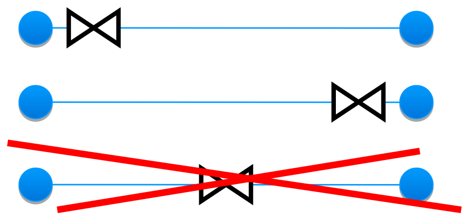


More formally



Other constraints

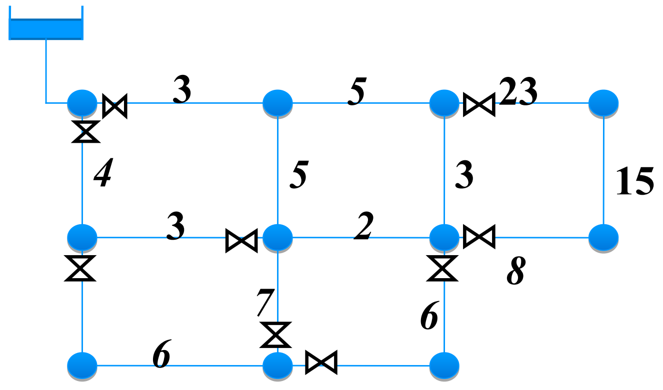
- *Valves can be placed only near one of the endpoints in an edge*



so, there are at most two valves in one edge

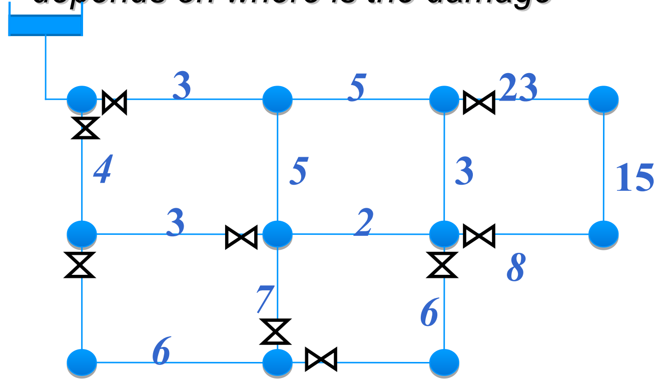
- *There must always be a way to isolate any pipe*
- *The number of available valves is given*

Assignment cost



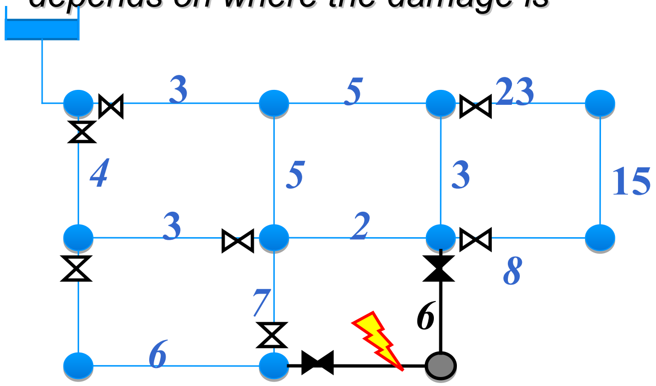
Cost of an assignment

- Undelivered demand in case of damage
- depends on where is the damage



Cost of an assignment

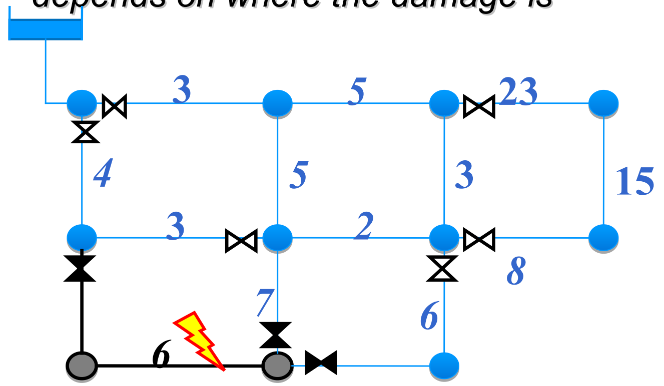
- Undelivered demand in case of damage
- depends on where the damage is



case 1: 6

Cost of an assignment

- Undelivered demand in case of damage
- depends on where the damage is

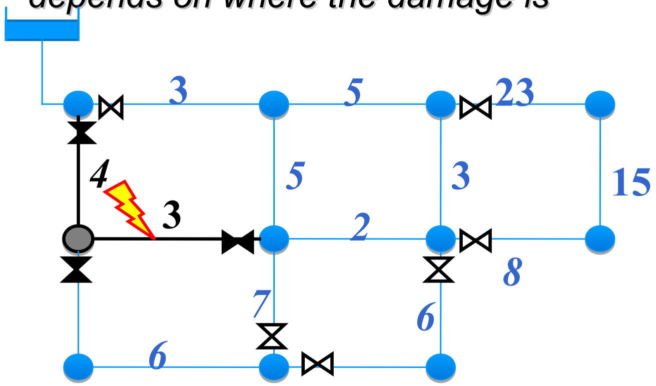


case 1: 6

case 2: 6

Cost of an assignment

- Undelivered demand in case of damage
- depends on where the damage is



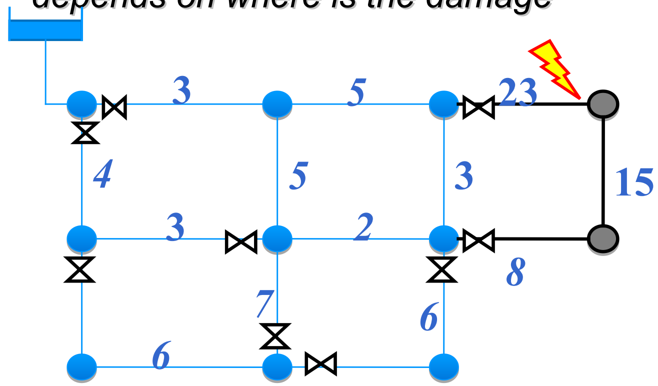
case 1: 6

case 2: 6

case 3: 7

Cost of an assignment

- Undelivered demand in case of damage
- depends on where is the damage



case 1: 6

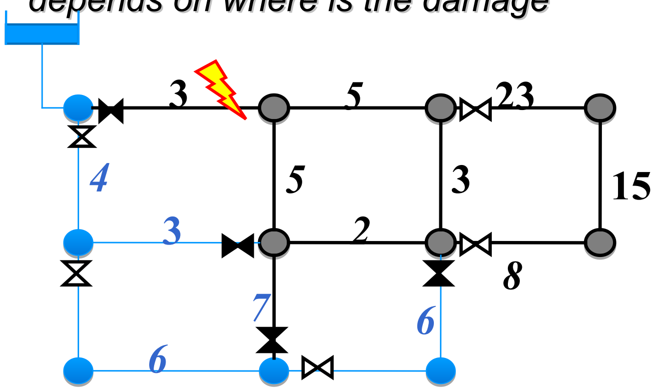
case 2: 6

case 3: 7

case 4: 46

Cost of an assignment

- Undelivered demand in case of damage
- depends on where is the damage



case 1: 6

case 2: 6

case 3: 7

case 4: 46

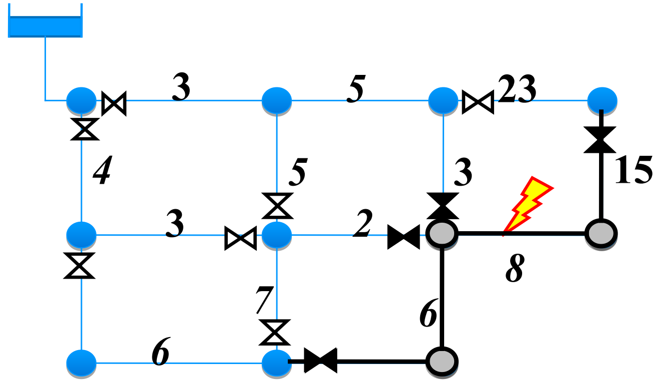
case 4: 64

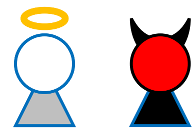
worst case: 64

Game model

1. player 1 places all available valves (say, n)
2. player 2 damages one pipe
3. player 1 closes the valves and fixes the broken pipe

The undelivered demand is a cost for player 1, and reward for player 2





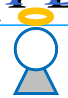
undelivered demand

15 +
8 +
6 =

29

Implementation in CLP(FD)

1. Player 1 places all available valves



Problem: generates a huge number of moves!

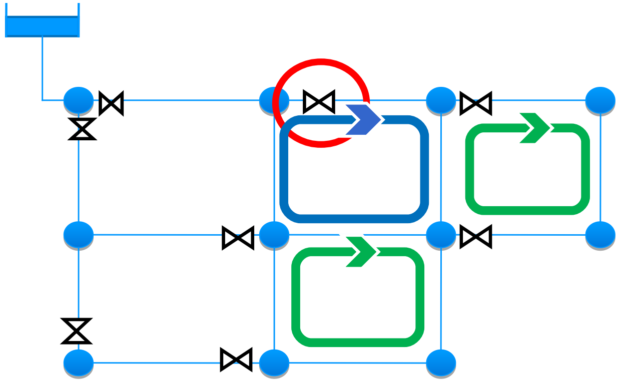
Solution: implement in CLP(FD)

- *Variables*: one variable for each possible position of a valve (2 per pipe)
- *Domains*: 0-1 (0=no valve, 1=valve)
- *Constraints*: Number of valves = Nv

Other constraints:

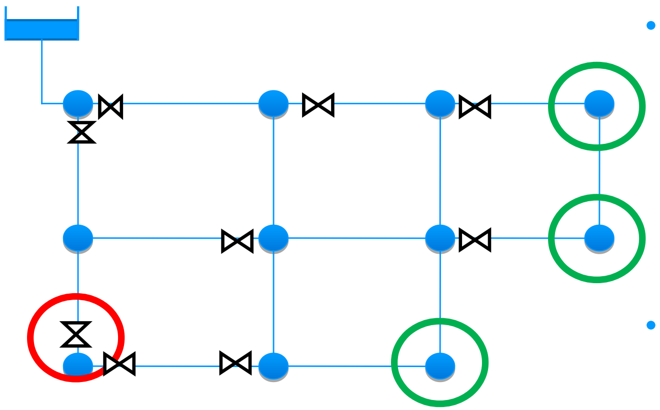
- prune clearly suboptimal solutions (useless valves)
- remove symmetric solutions

useless valves



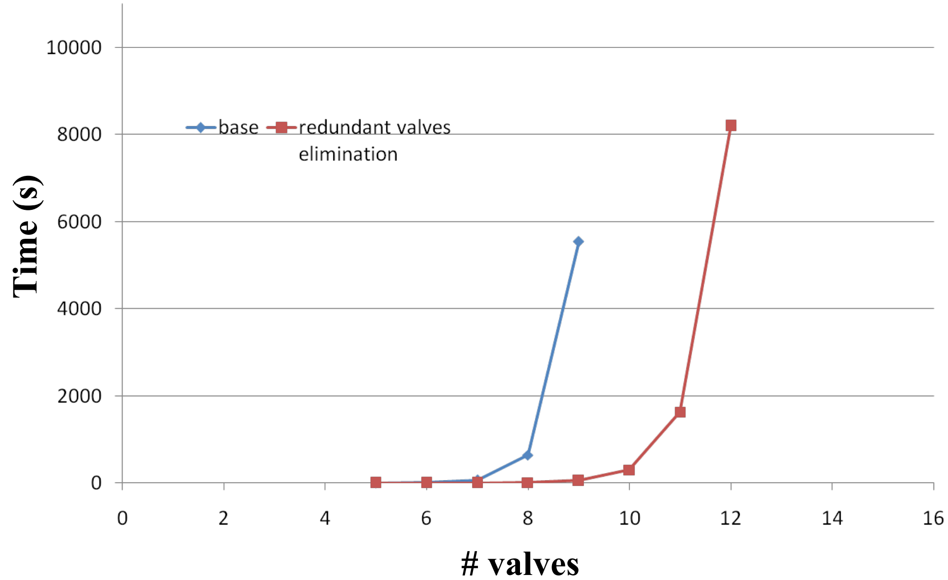
- There cannot be exactly one valve in any closed circuit
- Two or more valves -> ok
  - they can define a sector
- Zero valves -> ok
  - the circuit is included in some sector

symmetries



- Nodes with exactly two incident edges have a symmetry
  - because weights are on edges, not on nodes
- Remove immediately one of the positions near such nodes

Results: computation time

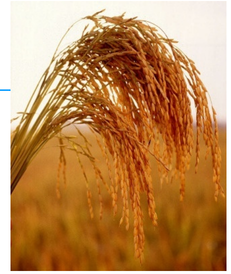


Computational Sustainability

- New research field
- Many environmental problems are combinatorial in nature
  - Ecologic Corridors: Three national parks should be connected, buying land from owners. Which lands should be bought to minimize cost? [Gomes et al]

www.computational-sustainability.org

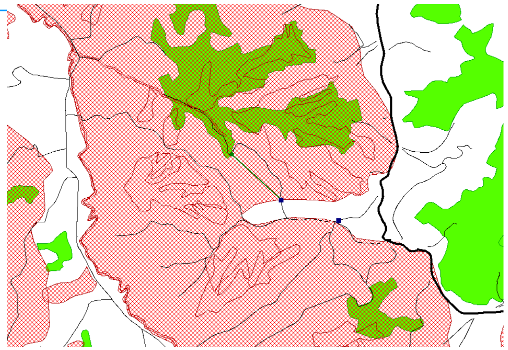
Biomass plant location



- Biomass power plants can use different types of fuel:
  - Hay
  - Wood chops, …
- Carbon neutral: biomass comes from plants that converted CO_2 into O_2
- But: we have to transport biomass to the plant, and transport produces CO_2!
- Is it worth the while?

Biomass plant location

- Green = Forest
- Red = impossible location
- Black = roads



- Find the best location for a biomass plant, maximizing the difference between produced energy and energy used to build the plant/transportation