# Chapter 16: More Global Constraints (Car Sequencing)

Helmut Simonis

Cork Constraint Computation Centre
Computer Science Department
University College Cork
Ireland

ECLiPSe ELearning Overview





## Licence

# Outline







# What we want to introduce

- Car sequencing problem
- `gcc` global cardinality constraint
- `sequence` constraint
- Search does not always have to be based on original problem variables
- Can be useful to consider additional variables which allow more clever search

# Problem Definition

## Car Sequencing

We have to schedule a number of cars for production on an assembly line. Each car is of a certain type, and we know how many cars of each type we have to produce. Car types differ in the options they require, i.e. sun-roof, air conditioning. For each option, we have capacity limits on the assembly line, expressed as $k$ cars out of $n$ consecutive cars on the line may have some option. Find an assignment which produces the correct number of cars of each type, while satisfying the capacity constraints.





# Example (DSV88)

- 100 cars
- 18 types
- 5 options
  - Option 1: 1 out of 2
  - Option 2: 2 out of 3
  - Option 3: 1 out of 3
  - Option 4: 2 out of 5
  - Option 5: 1 out of 5

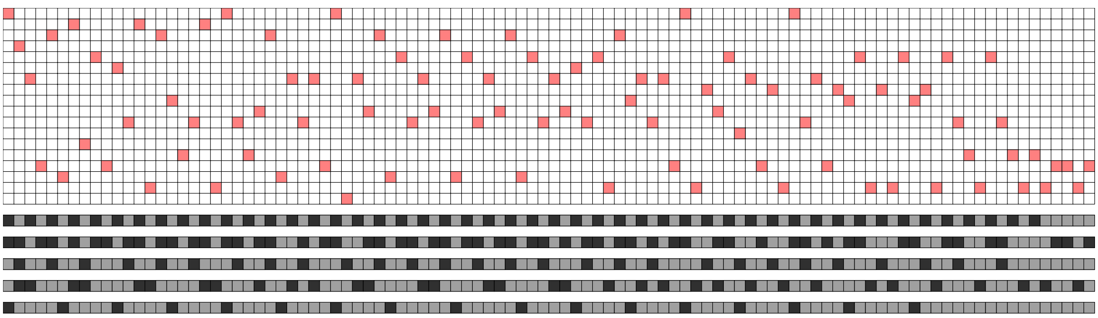
# Car Types

| Type | Cars Required | Option 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 1 | 5 | 1 | 1 | 0 | 0 | 1 |
| 2 | 3 | 1 | 1 | 0 | 1 | 0 |
| 3 | 7 | 1 | 1 | 1 | 0 | 0 |
| 4 | 1 | 0 | 1 | 1 | 1 | 0 |
| 5 | 10 | 1 | 1 | 0 | 0 | 0 |
| 6 | 2 | 1 | 0 | 0 | 0 | 1 |
| 7 | 11 | 1 | 0 | 0 | 1 | 0 |
| 8 | 5 | 1 | 0 | 1 | 0 | 0 |
| 9 | 4 | 0 | 1 | 0 | 0 | 1 |
| 10 | 6 | 0 | 1 | 0 | 1 | 0 |
| 11 | 12 | 0 | 1 | 1 | 0 | 0 |
| 12 | 1 | 0 | 0 | 1 | 0 | 1 |
| 13 | 1 | 0 | 0 | 1 | 1 | 0 |
| 14 | 5 | 1 | 0 | 0 | 0 | 0 |
| 15 | 9 | 0 | 1 | 0 | 0 | 0 |
| 16 | 5 | 0 | 0 | 0 | 0 | 1 |
| 17 | 12 | 0 | 0 | 0 | 1 | 0 |
| 18 | 1 | 0 | 0 | 1 | 0 | 0 |





# Solution



# Modelling Alternatives

- Assign start time (sequence number) to each car
  - 100 variables, each with 100 values
  - Handling of car types implicit
  - Symmetry breaking for cars of same type (inequalities)?
  - Capacity constraints?
- Assign car type to each slot on assembly line
  - 100 variables, 18 values
  - How to control number of cars of each type?
  - How to express capacity constraints?





# Model

- 100 variables ranging over car types
- `gcc` constraint to control number of items with same type
- $5 \times 100$ 0/1 variables indicating use of option for each slot
- `element` constraints to map car types to options used
- `sequence` constraints to enforce limits on each option

## Reminder: `gcc(Pattern, Variables)`

- `gcc` *global cardinality constraint*
- `Pattern` is list of terms `gcc(Low, High, Value)`
- The overall number of variables taking value `Value` is between `Low` and `High`
- Generalization of `alldifferent`
- Domain consistent version in ECLiPSe





## `gcc` Example

```
X1 :: [2,4], X2 :: [1,3,4], X3 :: [1,2,3,4],
X4 :: [3,4,5], X5 :: [3,4,5],
gcc([gcc(1,1,1),gcc(2,3,2),gcc(1,3,3),
     gcc(0,4,4),gcc(1,3,5)],
    [X1,X2,X3,X4,X5]),
```

X1 = ?, X2 = ?, X3 = ?, X4 = ?, X5 = ?

# gcc Reasoning

```
X1 :: [2,4], X2 :: [1,3,4], X3 :: [1,2,3,4],
X4 :: [3,4,5], X5 :: [3,4,5],
gcc([gcc(1,1,1),gcc(2,3,2),gcc(1,3,3),
     gcc(0,4,4),gcc(1,3,5)],
    [X1,X2,X3,X4,X5]),
```

X1 = ?2, X2 = ?, X3 = ?2, X4 = ?, X5 = ?





# gcc Next Step

X1 :: [2,~~4~~], X2 :: [1,3,4], X3 :: [~~1~~,2,~~3~~,~~4~~],<br>
X4 :: [3,4,5], X5 :: [3,4,5],<br>
gcc([gcc(1,1,1),gcc(2,3,2),gcc(1,3,3),<br>
     gcc(0,4,4),gcc(1,3,5)],<br>
    [X1,X2,X3,X4,X5]),

X1 = 2, X2 = ?1, X3 = 2, X4 = ?, X5 = ?

## gcc Continued

X1 :: [2,~~4~~], X2 :: [1,~~3~~,~~4~~], X3 :: [~~1~~,2,~~3~~,~~4~~],
X4 :: [3,4~~4~~,5], X5 :: [3,4~~4~~,5],
gcc([gcc(1,1,1),gcc(2,3,2),gcc(1,3,3),
gcc(0,4,4),gcc(1,3,5)],
[X1,X2,X3,X4,X5]),

X1 = 2, X2 = 1, X3 = 2, X4 = ?, X5 = ?





## gcc Made Domain Consistent

X1 :: [2,~~4~~], X2 :: [1,~~3~~,~~4~~], X3 :: [~~1~~,2,~~3~~,~~4~~],
X4 :: [3,~~4~~,5], X5 :: [3,~~4~~,5],
gcc([gcc(1,1,1),gcc(2,3,2),gcc(1,3,3),
gcc(0,4,4),gcc(1,3,5)],
[X1,X2,X3,X4,X5]),

X1 = 2, X2 = 1, X3 = 2, X4 $\in \{3, 5\}$, X5 $\in \{3, 5\}$

# How does the constraint solver do that?

Explained in optional material at end

Domain Consistent `gcc`





# Reminder: `element(X,List,Y)`

- `List` is a list of integers
- The $X^{th}$ element of `List` is $Y$
- The index starts from 1
- Typical uses:
  - Projection
  - Cost

# Element Examples

`Prime` is 1 iff $X \in 1..10$ is a prime number

```
X :: 1..10,
element(X,[1,1,1,0,1,0,1,0,0,0],Prime),
```

`Cost` is the cost corresponding to the assignment of $Y$

```
Y :: 1..10,
element(Y,[5,3,34,0,3,1,12,12,1,3],Cost)
```





# `sequence_total(Min,Max,Low,High,K,Vars)`

- Variables `Vars` have 0/1 domain
- Between `Min` and `Max` variables have value 1
- For every sub-sequence of length $K$, between `Low` and `High` variables have value 1

# sequence_total Example

```
[X1,X2,X3,X4,X5,X6,X7,X8,X9,X10] :: 0..1,
sequence_total(2,3,1,2,3,
               [X1,X2,X3,X4,X5,X6,X7,X8,X9,X10]),
```

X1 = 0, X4 = 0, X7 = 0, X10 = 0





# Example, cont'd

$$x_1, \overbrace{x_2, x_3, x_4}^{1..2}, \overbrace{x_5, x_6, x_7}^{1..2}, \overbrace{x_8, x_9, x_{10}}^{1..2}$$

$$\underbrace{0, \overbrace{\overbrace{x_2, x_3, x_4}^{1..2}, \overbrace{x_5, x_6, x_7}^{1..2}, \overbrace{x_8, x_9, x_{10}}^{1..2}}^{3..6}}_{2..3}$$

# Mathematical Equivalent

$$\text{Vars} = [x_1, x_2, ...x_N]$$

$$\text{Min} \leq \sum_{1 \leq i \leq N} x_i \leq \text{Max}$$

$$1 \leq s \leq N - k + 1 : \quad \text{Low} \leq \sum_{s \leq j \leq s+k-1} x_j \leq \text{High}$$

Cork Constraint Computation Centre





# Mathematical Equivalent

- Pruning very different when using finite domain inequalities
- Currently no domain consistent implementation of `sequence_total`
- Weaker version `sequence` (no global counters) domain consistent
- Currently using decomposition:
  - `sequence_total` = `sequence` + `gcc` + more

Cork Constraint Computation Centre

# Main Program

```
:-module(car).
:-export(top/0).
:-lib(ic).
:-lib(ic_global_gac).

top:-
    problem(Problem),
    model(Problem,L),
    writeln(L).
```





# Structure Definitions

```
:-local struct(problem(cars,
                       models,
                       required,
                       using_options,
                       value_order)).
:-local struct(option(k,
                      n,
                      index_set,
                      total_use)).
```

# Model (Part 1)

```
model(problem{cars:NrCars,
              models:NrModels,
              required:Required,
              using_options:List,
              value_order:Ordered},L):-
    length(L,NrCars),
    L :: 1..NrModels,
    (foreach(Cnt,Required),
     count(J,1,_),
     foreach(gcc(Cnt,Cnt,J),Card) do
        true
    ),
    gcc(Card,L),
    ...
```





# Model (Part 2)

```
    ...
    (foreach(option{k:K,
                    n:N,
                    index_set:IndexSet,
                    total_use:Total},List),
     param(L,NrCars) do
        (foreach(X,L),
         foreach(B,Binary),
         param(IndexSet) do
            element(X,IndexSet,B)
        ),
        sequence_total(Total,Total,0,K,N,Binary)
    ),
    search(L,0,input_order,ordered(Ordered),
```

# Data

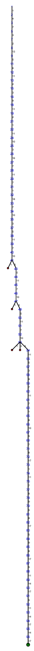
```
problem(100,18,
  [5,3,7,1,10,2,11,5,4,6,12,1,1,5,9,5,12,1],
  [option(1,2,[1,2,3,5,6,7,8,14],
     [1,1,1,0,1,1,1,1,0,0,0,0,0,1,0,0,0,0],48),
   option(2,3,[1,2,3,4,5,9,10,11,15],
     [1,1,1,1,1,0,0,0,1,1,1,0,0,0,1,0,0,0],57),
   option(1,3,[3,4,8,11,12,13,18],
     [0,0,1,1,0,0,0,1,0,0,1,1,1,0,0,0,0,1],28),
   option(2,5,[2,4,7,10,13,17],
     [0,1,0,1,0,0,1,0,0,1,0,0,1,0,0,0,1,0],34),
   option(1,5,[1,6,9,12,16],
     [1,0,0,0,0,1,0,0,1,0,0,1,0,0,0,1,0,0],17)],
  [1,3,2,4,6,8,7,12,13,5,9,11,10,14,16,18,17,15]).
```





# Data Generation

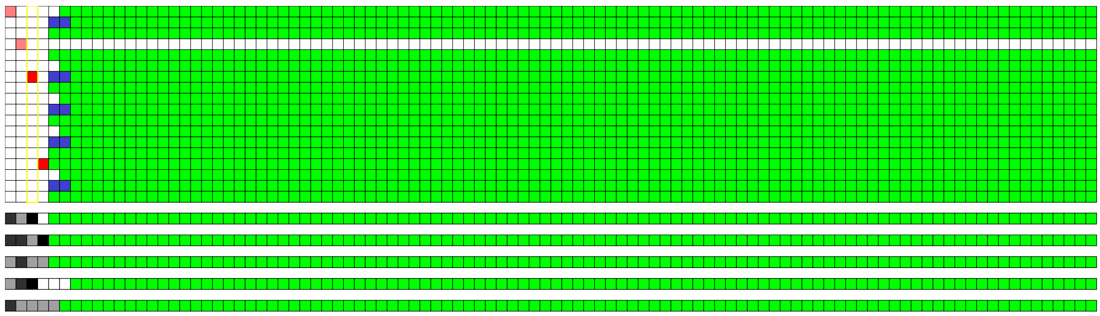
- Data not really stored as facts
- Generated from text data files in different format
- Benchmark set from CSPLIB (`http://www.csplib.org`)



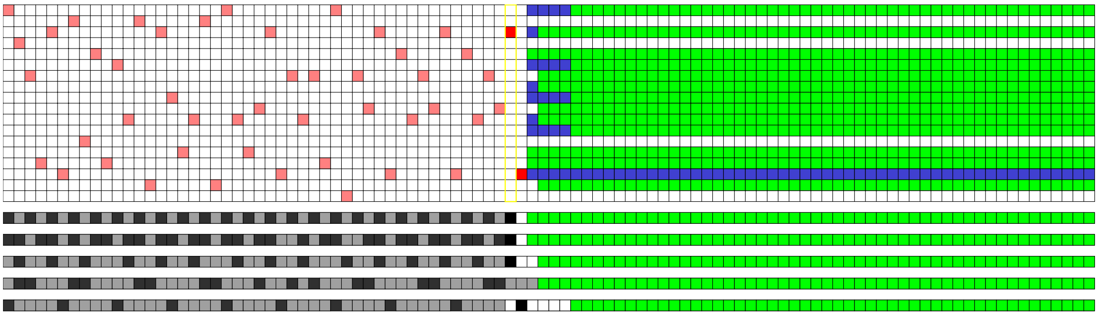




## Assignment Step 4



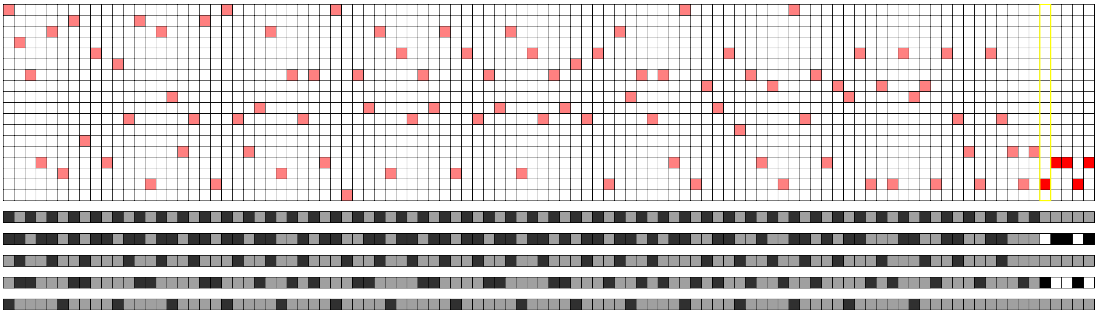
# Assignment Step 40







# Assignment Step 83



# Another Example (PR97)

- 100 cars
- 22 types
- 5 options
  - Option 1: 1 out of 2
  - Option 2: 2 out of 3
  - Option 3: 1 out of 3
  - Option 4: 2 out of 5
  - Option 5: 1 out of 5

Cork Constraint Computation Centre



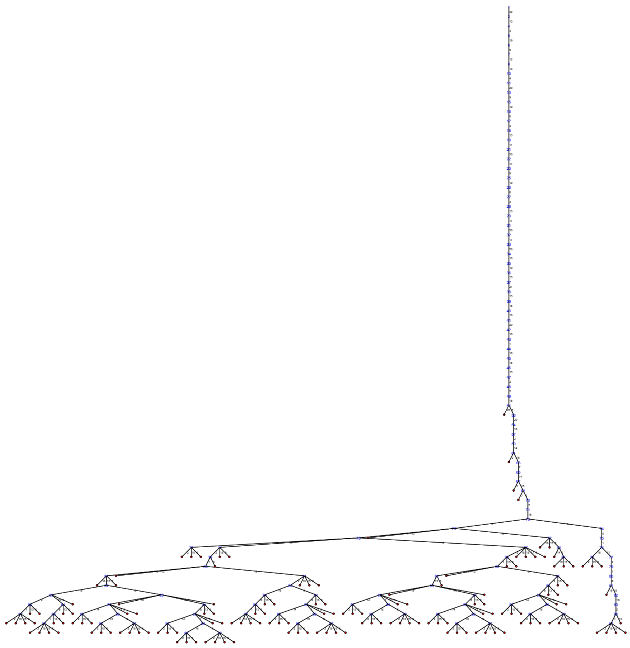


# Second Example: Car Types

| Type | Cars Required | Option 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| 1 | 6 | 1 | 0 | 0 | 1 | 0 |
| 2 | 10 | 1 | 1 | 1 | 0 | 0 |
| 3 | 2 | 1 | 1 | 0 | 0 | 1 |
| 4 | 2 | 0 | 1 | 1 | 0 | 0 |
| 5 | 8 | 0 | 0 | 0 | 1 | 0 |
| 6 | 15 | 0 | 1 | 0 | 0 | 0 |
| 7 | 1 | 0 | 1 | 1 | 1 | 0 |
| 8 | 5 | 0 | 0 | 1 | 1 | 0 |
| 9 | 2 | 1 | 0 | 1 | 1 | 0 |
| 10 | 3 | 0 | 0 | 1 | 0 | 0 |
| 11 | 2 | 1 | 0 | 1 | 0 | 0 |
| 12 | 1 | 1 | 1 | 1 | 0 | 1 |
| 13 | 8 | 0 | 1 | 0 | 1 | 0 |
| 14 | 3 | 1 | 0 | 0 | 1 | 1 |
| 15 | 10 | 1 | 0 | 0 | 0 | 0 |
| 16 | 4 | 0 | 1 | 0 | 0 | 1 |
| 17 | 4 | 0 | 0 | 0 | 0 | 1 |
| 18 | 2 | 1 | 0 | 0 | 0 | 1 |
| 19 | 4 | 1 | 1 | 0 | 0 | 0 |
| 20 | 6 | 1 | 1 | 0 | 1 | 0 |
| 21 | 1 | 1 | 0 | 1 | 0 | 1 |
| 22 | 1 | 1 | 1 | 1 | 1 | 1 |

Cork Constraint Computation Centre

# Search (Stopped After 1000 Nodes)







# Observation

- This does not look good
- Typical `thrashing` behaviour
- We made a wrong choice at some point
- ... but did not detect it
- Many additional choices are made before failure is detected
- We have to explore the complete tree under the wrong choice
- This is far too expensive

## Change of Search Strategy

- Do not label car slot variables
- Decide instead if slot should use an option or not
- This restricts the car models which can be placed in this slot
- Start with the most restricted option
- When all options are assigned, the car type is fixed
- Potential problem: We now have 500 instead of 100 decision variables
- Naive searchspace $2^{500} = 3.2e150$ instead of $22^{100} = 1.7e134$

Cork Constraint Computation Centre





## Second Modification

- Instead of assigning values left to right
- Start assigning in middle of board
- And alternate around middle until you reach edges
- Idea: Slots at edges are less constrained, i.e. easier to assign
- Save those slots until the end
- We already encountered this idea for the N-Queens problem

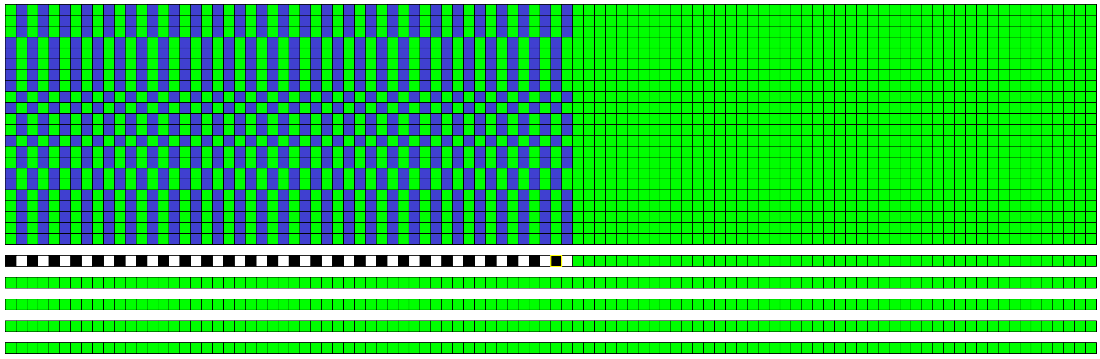
Cork Constraint Computation Centre

# Modified Search

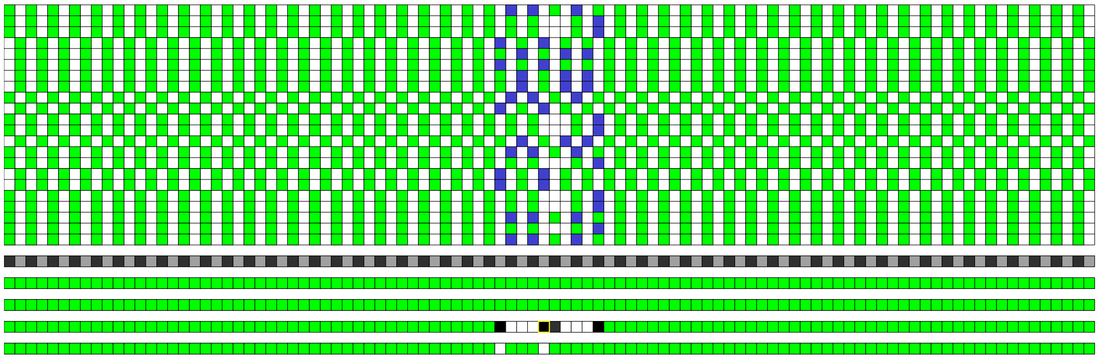




# Assignment Step 2



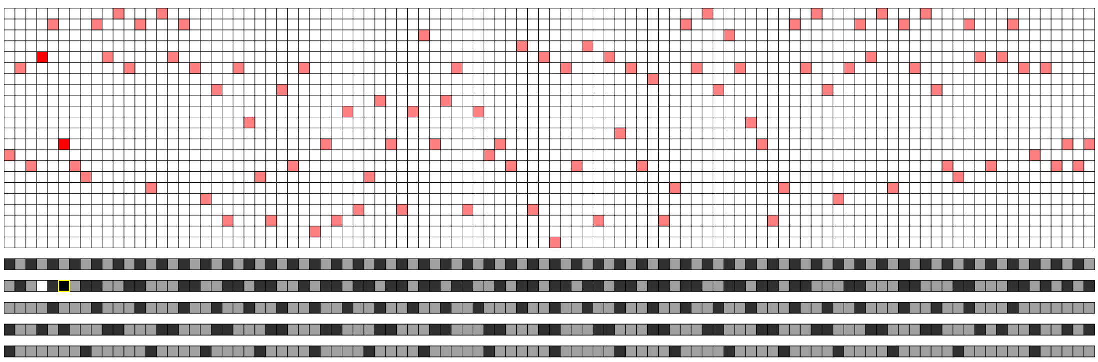
# Assignment Step 28







# Assignment Step 119



# Observations

- Important to start in middle
- Making hard choices first
- Concentrate on difficult to satisfy sub-problem
- Number of choices is much smaller than number of variables
- Some assignments lead to a lot of propagation





# Conclusions

- Introduced global constraint `sequence`
- Reuse `gcc` and `element`
- Search on auxiliary variables can work well
- Raw search space measures are unreliable
- Modelling idea
  - Decide what to make in a given time slot
  - ... and not when to schedule some given activity

# Making gcc Domain Consistent

```
X1 :: [2,4], X2 :: [1,3,4], X3 :: [1,2,3,4],
X4 :: [3,4,5], X5 :: [3,4,5],
gcc([gcc(1,1,1),gcc(2,3,2),gcc(1,3,3),
      gcc(0,4,4),gcc(1,3,5)],
     [X1,X2,X3,X4,X5]),
```

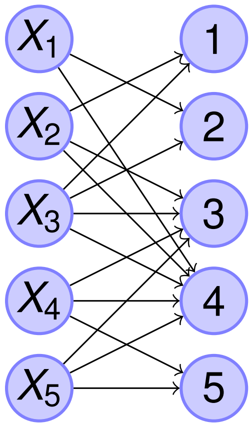




# Method: Max Flow Model

- Express constraint as max-flow problem
- Any flow solution corresponds to a valid assignment
- Only work with one flow solution
- Obtain all others by considering
  - *residual graph* and
  - *strongly connected components*
- Classical method, faster methods exist
- Use of max flow based propagators for many constraints

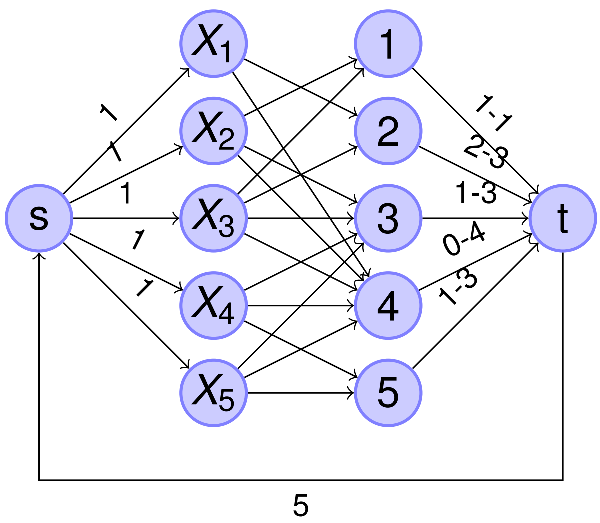
## Start with Value Graph



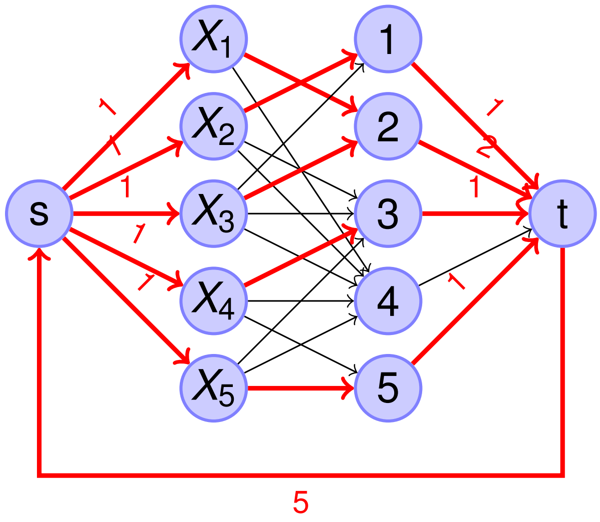





## Convert to Flow Problem



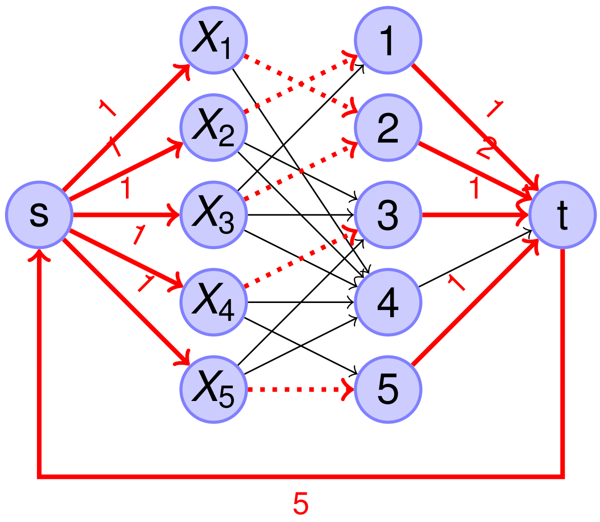
# Find Maximal Flow



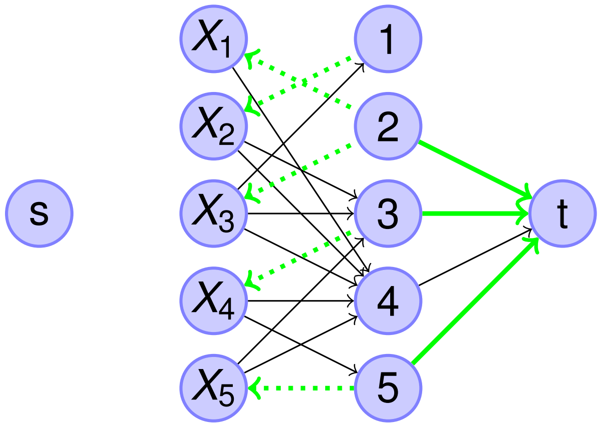





# Mark Value Edges in Flow



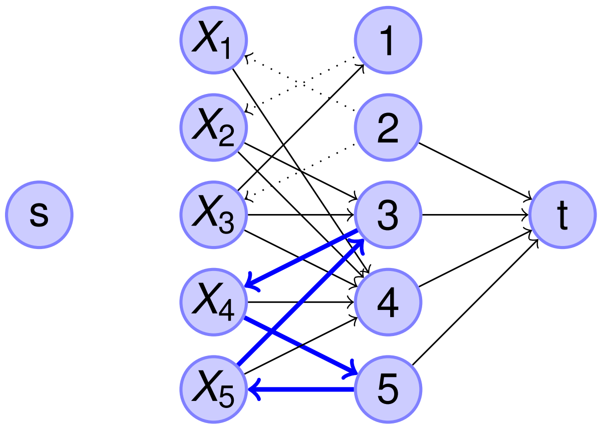
# Residual Graph



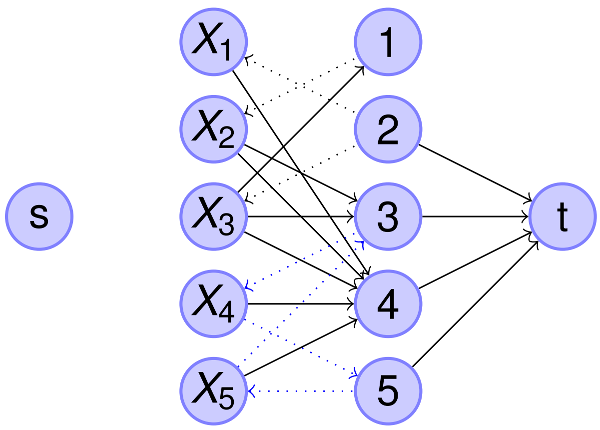





# Find Strongly Connected Components



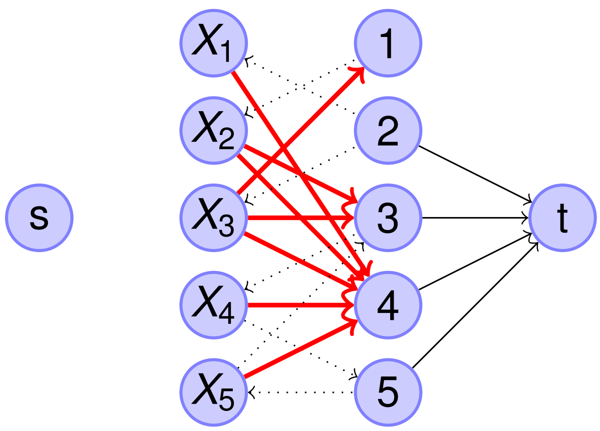
# Mark Edges



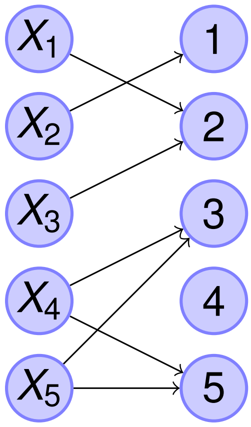





# Remove Unmarked Edges



## Constraint is Domain Consistent








## More Information

Mehmet Dincbas, Helmut Simonis, and Pascal Van Hentenryck.
Solving the car-sequencing problem in constraint logic programming.
In *ECAI*, pages 290–295, 1988.

Jean-Charles Regin and Jean-Francois Puget.
A filtering algorithm for global sequencing constraints.
In Gert Smolka, editor, *CP*, volume 1330 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 1997.

## More Information


- Christine Solnon, Van Dat Cung, Alain Nguyen, and Christian Artigues.
  The car sequencing problem: overview of state-of-the-art methods and industrial case-study of the ROADEF 2005 challenge problem.
  *European Journal of Operational Research*, Volume 191:912–927, December 2008.
- Willem Jan van Hoeve, Gilles Pesant, Louis-Martin Rousseau, and Ashish Sabharwal.
  Revisiting the sequence constraint.
  In Frederic Benhamou, editor, *CP*, volume 4204 of *Lecture Notes in Computer Science*, pages 620–634. Springer, 2006.






## More Information


- Michael J. Maher, Nina Narodytska, Claude-Guy Quimper, and Toby Walsh.
  Flow-based propagators for the sequence and related global constraints.
  In Peter J. Stuckey, editor,*Principles and Practice of Constraint Programming, 14th International Conference, CP 2008, Sydney, Australia, September 14-18, 2008. Proceedings*, pages 159–174.