Constraint Programming Tutorial

John Hooker
Carnegie Mellon University

Institute of Computing
State University of Campinas, Brazil
September-October 2012

A First Glimpse at Constraint Programming

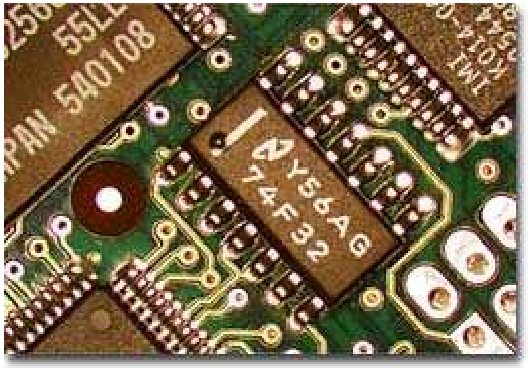
Applications, Early Successes
Advantages and Disadvantages
Software
Tutorial Outline and Calendar
References

What is constraint programming?

- An alternative to optimization methods in operations research.
- Developed in the computer science and artificial intelligence communities.
  - Over the last 20+ years.
- Particularly successful in scheduling and logistics.

Early commercial successes

- Circuit design (Siemens)



- Container port scheduling (Hong Kong and Singapore)

- Real-time control (Siemens, Xerox)

Applications

- Job shop scheduling
- Assembly line smoothing and balancing
- Cellular frequency assignment
- Nurse scheduling
- Shift planning
- Maintenance planning
- Airline crew rostering and scheduling
- Airport gate allocation and stand planning

capitol technologies, inc.
FACTORY AUTOMATION SYSTEMS
automation@capitoltech.com
www.capitoltech.com
3615 W. Voorde Dr.
South Bend, IN 46628

Applications

- Production scheduling
  - chemicals
  - aviation
  - oil refining
  - steel
  - lumber
  - photographic plates
  - tires
- Transport scheduling (food, nuclear fuel)
- Warehouse management
- Course timetabling

Advantages of CP

- Good at scheduling, logistics
  - …where other optimization methods may fail.
- Adding messy constraints makes the problem easier.
  - The more constraints, the better.
- More powerful modeling language.
  - Simpler models (due to global constraints).
  - Constraints convey problem structure to the solver.

Disdvantages of CP

- Less effective for continuous optimization.
    - Relies on interval propagation
- Less robust
    - May blow up past a certain problem size,
    - Lacks relaxation technology
- Software is less highly engineered
    - Younger field

Comparison with Mathematical Programming

MP	CP
Numerical calculation	Logic processing
Relaxation	Inference (filtering, constraint propagation)
Atomistic modeling (linear inequalities)	High-level modeling (global constraints)
Branching	Branching
Independence of model and algorithm	Constraint-based processing

Complementary Strengths

- CP can be profitably combined with other optimization methods.
  - Integer programming, global optimization
  - Combine complementary strengths

Software for CP

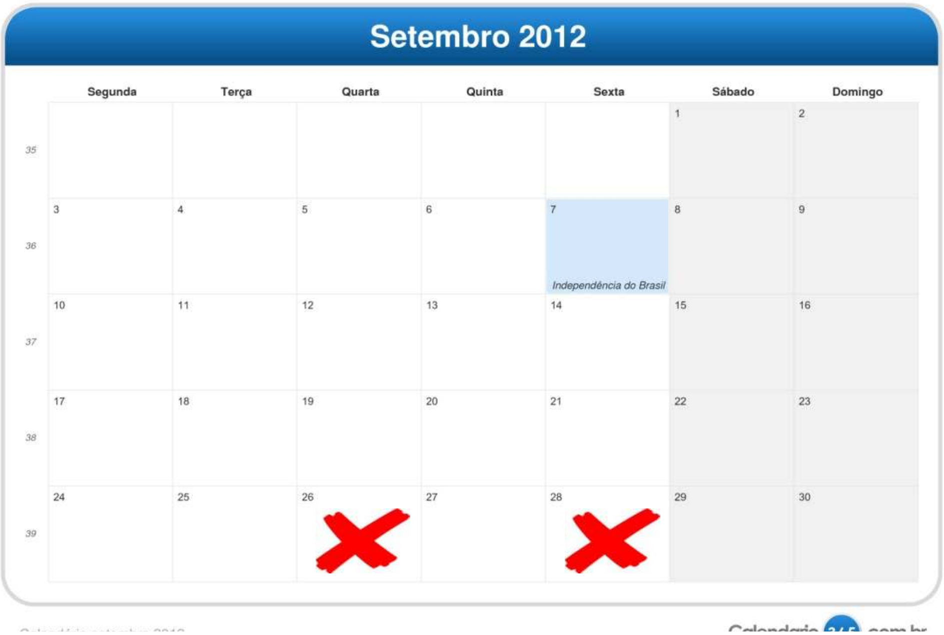
- ECLiPSe (NICTA), open source
  - Early CP (and hybrid) solver, still maintained
- CHIP (Cosytec), commercial
  - State-of-the-art solver
- OPL CP Optimizer (IBM), commercial (free academic download)
  - State-of-the-art solver, originally developed by ILOG
- Gecode (Schulte & Tack), free download
  - State-of-the-art toolkit for building CP solvers
- Frontline MIP/CP solver (Frontline Systems), commercial
  - Add-in for Excel spreadsheets
- G12 (NICTA), under development
  - Major CP and hybrid system
- Google OR-tools (Google), open source
  - Includes CP solver

Tentative Outline

- A First Glimpse at CP
- Basic Ideas of CP
- CP Modeling
- Consistency and Backtracking
- Review of Network Flow Theory
- The Alldiff, Cardinality and Nvalues Constraints
- The Sequence Constraint
- The Regular Constraint
- Disjunctive and Cumulative Scheduling
- Propositional Satisfiability (SAT)
- Symmetry
- Advanced Modeling
- CP/OR Integration

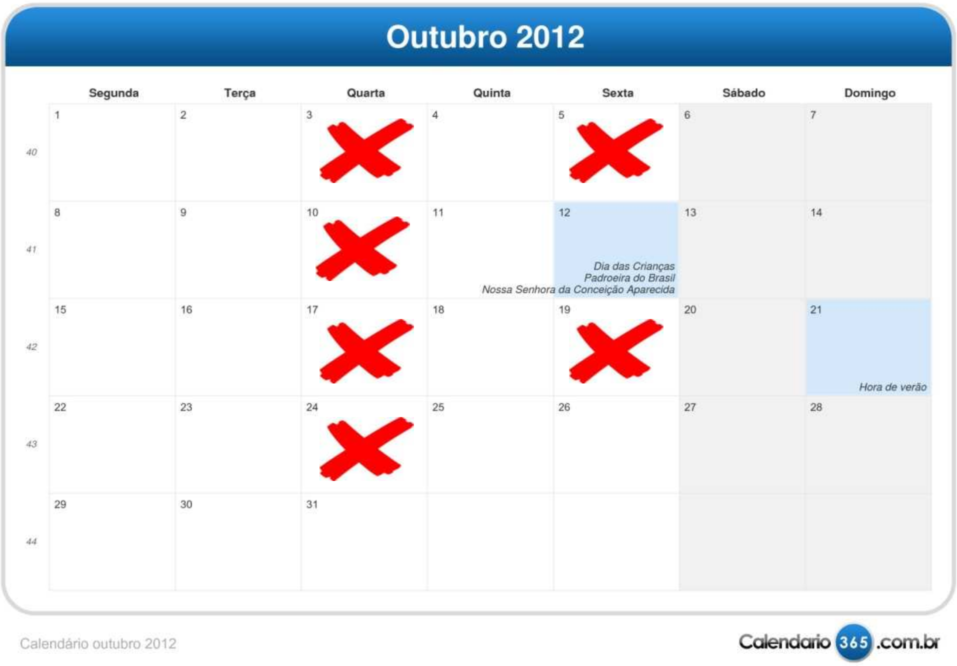
Calendar

- Quarta-feira: 6 - 8 pm
- Sexta-feira: 10am - 12



Setembro 2012

	Segunda	Terça	Quarta	Quinta	Sexta	Sábado	Domingo
35						1	2
36	3	4	5	6	7 Independência do Brasil	8	9
37	10	11	12	13	14	15	16
38	17	18	19	20	21	22	23
39	24	25	26 ✗	27	28 ✗	29	30



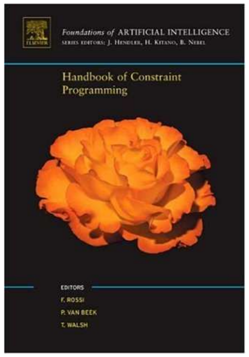
Outubro 2012

	Segunda	Terça	Quarta	Quinta	Sexta	Sábado	Domingo
40	1	2	3 ✗	4	5 ✗	6	7
41	8	9	10 ✗	11	12 Dia das Crianças Padroeira do Brasil Nossa Senhora da Conceição Aparecida	13	14
42	15	16	17 ✗	18	19 ✗	20	21 Hora de verão
43	22	23	24 ✗	25	26	27	28
44	29	30	31				

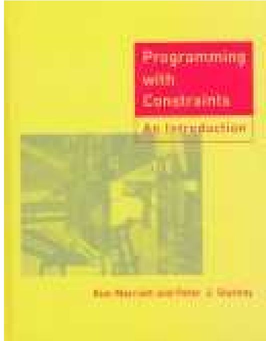
Calendário outubro 2012

Calendario 365 .com.br

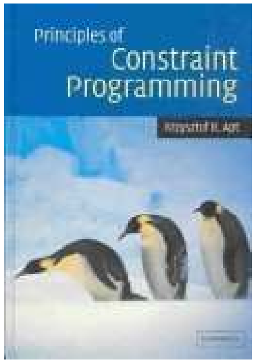
References


Foundations of ARTIFICIAL INTELLIGENCE
Handbook of Constraint Programming
EDITORS
F. ROSSI
P. VAN BEEK
T. WALSH

Handbook of Constraint Programming, F. Rossi, P. van Beek, T. Walsh, eds.

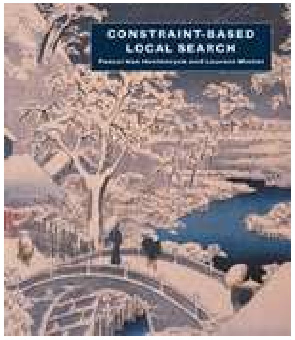


Programming with *Constraints*, K. Marriott, P. J. Stuckey

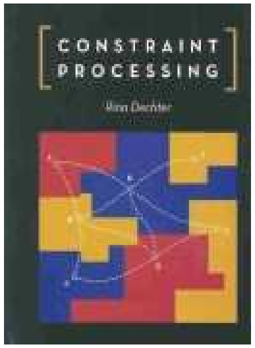


Principles of Constraint Programming, K. Apt

Constraint-Based Local Search, P. Van Hentenryck and L. Michel



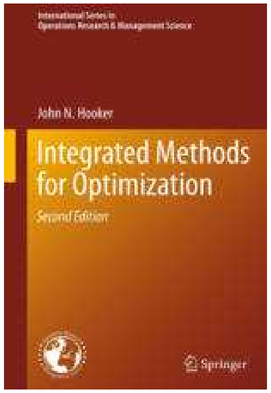
Constraint Processing, R. Dechter



References

This tutorial is based partly on:

- J. N. Hooker, *Integrated Methods for Optimization*, 2nd ed., Springer (2012). Contains references and many exercises.


International Series in
Operations Research & Management Science
John N. Hooker
Integrated Methods for Optimization
Second Edition
Springer

References

Online resources:

- Introductory material on CP in Portuguese (thesis by T. Serra)
- 2011 CP Summer School (slides only)
- **2009 CPAIOR Tutorial in CP (slides and videos)**
- 2008 CP Summer School (slides only)
- **2007 CP Summer School (slides and videos)**
- Association for Constraint Programming
- **These slides (updated the day after each class).**
  - Google “John Hooker” to find website.

Basic Ideas of CP

Procedural and declarative models
Filtering and propagation
Global constraints

Basic Ideas of CP

- It is both **procedural** and **declarative**.
  - procedural = write a computer program
  - declarative = state constraints on the solution

Basic Ideas of CP

- It is both **procedural** and **declarative**.
  - procedural = write a computer program
  - declarative = state constraints on the solution

- It uses **global constraints** to exploit problem structure:
  - global constraint = constraint that contains many simpler constraints

Basic Ideas of CP

- It is both **procedural** and **declarative**.
  - procedural = write a computer program
  - declarative = state constraints on the solution
- It uses **global constraints** to exploit problem structure:
  - global constraint = constraint that contains many simpler constraints
- It uses **filtering** and **constraint propagation** to reduce the search space.
  - filtering = reduce variable domains
  - propagation = pass domains to next constraint

Procedural and Declarative Models

- Example: solve this:

$$3x_1 + x_2 + x_3 = 10$$

$$x_1, x_2, x_3 \text{ pairwise distinct}$$

$$x_1, x_2 \in \{1, 2\},\ x_3 \in \{1, 2, 3\}$$

Note that $x_1 = x_2 = x_3 = 2$ is not allowed.

Procedural and Declarative Models

- Example: solve this:

$$3x_1 + x_2 + x_3 = 10$$
$$x_1, x_2, x_3 \text{ pairwise distinct}$$
$$x_1, x_2 \in \{1, 2\},\ x_3 \in \{1, 2, 3\}$$

- Purely procedural model:

```
For $x_1$ = 1,2:
    For $x_2$ = 1,2:
        If $x_1 \neq x_2$ then
            For $x_3$ = 1,2,3:
                If $x_1 \neq x_3$ and $x_2 \neq x_3$ then
                    If $3x_1 + x_2 + x_3 = 10$ then print $x_1$, $x_2$, $x_3$
```

Procedural and Declarative Models

- Example: solve this:

$$3x_1 + x_2 + x_3 = 10$$

$$x_1, x_2, x_3 \text{ pairwise distinct}$$

$$x_1, x_2 \in \{1, 2\},\ x_3 \in \{1, 2, 3\}$$

- Purely declarative model:

$$3x_1 + x_2 + x_3 = 10$$

$$x_1 \neq x_2$$

$$x_1 \neq x_3$$

$$x_2 \neq x_3$$

$$x_1, x_2 \in \{1, 2\},\ x_3 \in \{1, 2, 3\}$$

Procedural and Declarative Models

- Example: solve this:

$$3x_1 + x_2 + x_3 = 10$$
$$x_1, x_2, x_3 \text{ pairwise distinct}$$
$$x_1, x_2 \in \{1, 2\},\ x_3 \in \{1, 2, 3\}$$

- Purely declarative model:

$$3x_1 + x_2 + x_3 = 10$$
$$x_1 \neq x_2$$
$$x_1 \neq x_3$$
$$x_2 \neq x_3$$
$$x_1, x_2 \in \{1, 2\},\ x_3 \in \{1, 2, 3\}$$

Looks simple, but how are we going to solve this?

Perhaps by integer programming…

Procedural and Declarative Models

- Example: solve this:

$$3x_1 + x_2 + x_3 = 10$$

$$x_1, x_2, x_3 \text{ pairwise distinct}$$

$$x_1, x_2 \in \{1, 2\},\ x_3 \in \{1, 2, 3\}$$

- Purely declarative model:

$$3x_1 + x_2 + x_3 = 10$$

$$x_1 - x_2 \geq 1 - 2y_{12}, \quad x_2 - x_1 \geq 2y_{12} - 1$$

$$x_1 - x_2 \geq 1 - 2y_{12}, \quad x_2 - x_1 \geq 2y_{12} - 1$$

$$x_1 - x_2 \geq 1 - 2y_{12}, \quad x_2 - x_1 \geq 2y_{12} - 1$$

$$1 \leq x_1, x_2 \leq 2, \quad 1 \leq x_3 \leq 3$$

$$x_1, x_2, x_3 \text{ integer}, \quad y_{12}, y_{13}, y_{23} \in \{0, 1\}$$

An integer programming model.

Don’t worry about why it works.

Can be solved by CPLEX, Gurobi, ExpressMP, SCIP, etc.

Procedural and Declarative Models

- Example: solve this:

$$
\begin{aligned}
&3x_1 + x_2 + x_3 = 10 \\
&x_1, x_2, x_3 \text{ pairwise distinct} \\
&x_1, x_2 \in \{1,2\},\ x_3 \in \{1,2,3\}
\end{aligned}
$$

- CP model:

$$
\begin{aligned}
&3x_1 + x_2 + x_3 = 10 \\
&\text{alldiff}\left(x_1, x_2, x_3\right) \\
&x_1, x_2 \in \{1,2\},\ x_3 \in \{1,2,3\}
\end{aligned}
$$

This **global constraint** (all-different) enforces $x_1 \neq x_2$, $x_1 \neq x_3$, $x_2 \neq x_3$.

Procedural and Declarative

- CP model:

$$3x_1 + x_2 + x_3 = 10$$
$$\text{alldiff}\left(x_1, x_2, x_3\right)$$
$$x_1, x_2 \in \{1, 2\},\ x_3 \in \{1, 2, 3\}$$

- The model looks **declarative**.
    - It consists of constraints.
    - They can be written in any order.
- But each constraint invokes a **procedure**.
    - The procedure reduces the search space by **filtering** and **propagation**.

Filtering

- CP model:

$$3x_1 + x_2 + x_3 = 10$$
$$\text{alldiff}(x_1, x_2, x_3)$$
$$x_1, x_2 \in \{1,2\},\ x_3 \in \{1,2,3\}$$

- Variable domains:

$$x_1 \in \{1,2,\quad\}$$
$$x_2 \in \{1,2,\quad\}$$
$$x_3 \in \{1,2,3\}$$

- Use the alldiff constraint to **filter** the domains (remove infeasible values).
  - x_1, x_2 must use the values 1,2.

Filtering

- CP model:

$$3x_1 + x_2 + x_3 = 10$$
$$\text{alldiff}\left(x_1, x_2, x_3\right)$$
$$x_1, x_2 \in \{1, 2\},\ x_3 \in \{1, 2, 3\}$$

- Variable domains:

$$x_1 \in \{1, 2, \quad\}$$
$$x_2 \in \{1, 2, \quad\}$$
$$x_3 \in \{\ ,\ ,3\}$$

- Use the alldiff constraint to **filter** the domains (remove infeasible values).
  - x_1, x_2 must use the values 1,2. So we filter these values from x_3's domain.

Filtering

- CP model:

$$3x_1 + x_2 + x_3 = 10$$
$$\text{alldiff}(x_1, x_2, x_3)$$
$$x_1, x_2 \in \{1,2\},\ x_3 \in \{1,2,3\}$$

- Variable domains:

$$x_1 \in \{1,2,\ \ \}$$
$$x_2 \in \{1,2,\ \ \}$$
$$x_3 \in \{\ \ ,\ \ ,3\}$$

- Use the alldiff constraint to **filter** the domains (remove infeasible values).
  - x_1, x_2 must use the values 1,2. So we filter these values from x_3's domain.
  - This can be generalized using network flow theory.

Filtering

- CP model:

$$3x_1 + x_2 + x_3 = 10$$
$$\text{alldiff}\left(x_1, x_2, x_3\right)$$
$$x_1, x_2 \in \{1, 2\},\ x_3 \in \{1, 2, 3\}$$

- Variable domains:

$$x_1 \in \{1, 2,\ \ \}$$
$$x_2 \in \{1, 2,\ \ \}$$
$$x_3 \in \{\ ,\ \ , 3\}$$

- Use the alldiff constraint to **filter** the domains (remove infeasible values).
  - x_1, x_2 must use the values 1,2. So we filter these values from x_3's domain.
  - Removing all infeasible values achieves **domain consistency**.

Propagation

- CP model:

$$3x_1 + x_2 + x_3 = 10$$
$$\text{alldiff}(x_1, x_2, x_3)$$
$$x_1, x_2 \in \{1,2\},\ x_3 \in \{1,2,3\}$$

- Variable domains:

$$x_1 \in \{1,2,\quad\}$$
$$x_2 \in \{1,2,\quad\}$$
$$x_3 \in \{\ ,\ \ ,3\}$$

- We now **propagate** the reduced domains to the first constraint.
  - Filter using first constraint:
  - Must have $3x_1 \geq 10 - \max\{1,2\} - \max\{3\} = 5$, or $x_1 \geq 2$.

Domain of x_2

Domain of x_3

Propagation

- CP model:

$$3x_1 + x_2 + x_3 = 10$$
$$\text{alldiff}\left(x_1, x_2, x_3\right)$$
$$x_1, x_2 \in \{1,2\},\ x_3 \in \{1,2,3\}$$

- Variable domains:

$$x_1 \in \{\ \ ,2,\ \ \}$$
$$x_2 \in \{1,2,\ \ \}$$
$$x_3 \in \{\ \ ,\ \ ,3\}$$

- We now **propagate** the reduced domains to the first constraint.
  - Filter using first constraint:
  - Must have $3x_1 \geq 10 - \max\{1,2\} - \max\{3\} = 5$, or $x_1 \geq 2$.
  - Filter domain of x_1.

Propagation

- CP model:

$$3x_1 + x_2 + x_3 = 10$$
$$\text{alldiff}\left(x_1, x_2, x_3\right)$$
$$x_1, x_2 \in \{1,2\},\ x_3 \in \{1,2,3\}$$

- Variable domains:

$$x_1 \in \{\ \ ,2,\ \ \}$$
$$x_2 \in \{1,2,\ \ \}$$
$$x_3 \in \{\ \ ,\ \ ,3\}$$

- **Propagate** this to alldiff constraint.
  - Filter domain of x_2.

Propagation

- CP model:

$$3x_1 + x_2 + x_3 = 10$$
$$\text{alldiff}\left(x_1, x_2, x_3\right)$$
$$x_1, x_2 \in \{1,2\},\ x_3 \in \{1,2,3\}$$

- Variable domains:

$$x_1 \in \{\ \ ,2,\ \ \}$$
$$x_2 \in \{1,\ \ ,\ \ \}$$
$$x_3 \in \{\ \ ,\ \ ,3\}$$

- **Propagate** this to alldiff constraint.
  - Filter domain of x_2.

Solution Found

- CP model:

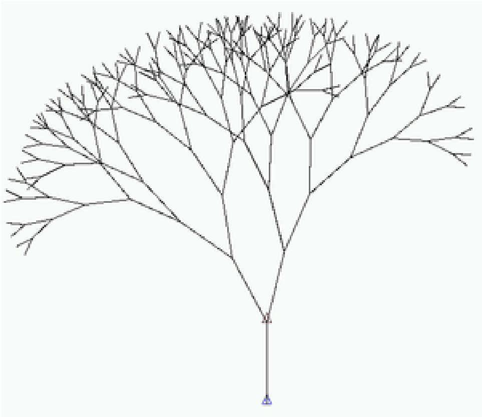
$$3x_1 + x_2 + x_3 = 10$$
$$\text{alldiff}\left(x_1, x_2, x_3\right)$$
$$x_1, x_2 \in \{1, 2\},\ x_3 \in \{1, 2, 3\}$$

- Variable domains:

$$x_1 \in \{\ \ , 2,\ \ \}$$
$$x_2 \in \{1,\ \ ,\ \ \}$$
$$x_3 \in \{\ \ ,\ \ , 3\}$$

- Because each domain is a **singleton**, we have a solution.
    - No more propagation needed.

Branching



- CP model:

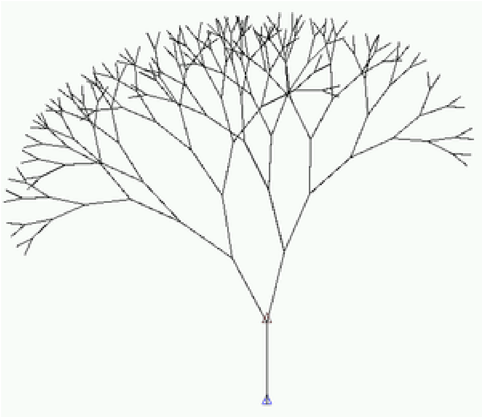
$$3x_1 + x_2 + x_3 = 10$$
$$\text{alldiff}\left(x_1, x_2, x_3\right)$$
$$x_1, x_2 \in \{1, 2\},\ x_3 \in \{1, 2, 3\}$$

- Variable domains:

$$x_1 \in \{\ \ , 2,\ \ \}$$
$$x_2 \in \{1,\ \ ,\ \ \}$$
$$x_3 \in \{\ \ ,\ \ , 3\}$$

- Branching is often necessary.

Branching



- CP model:

$$3x_1 + x_2 + x_3 = 10$$
$$\text{alldiff}\left(x_1, x_2, x_3\right)$$
$$x_1, x_2 \in \{1, 2\},\ x_3 \in \{1, 2, 3\}$$

- Variable domains:

$$x_1 \in \{\ \ , 2,\ \ \}$$
$$x_2 \in \{1, 2,\ \ \}$$
$$x_3 \in \{\ \ ,\ \ , 3\}$$

- Branching is often necessary.
  - Suppose we don’t filter x_2’s domain.
  - Then we can branch:
    - Set x_2 = 1 and repeat process.
    - Set x_2 = 2 and repeat process.

Global constraints

- Global constraints like alldiff **exploit problem structure**.
    - Filtering for a global constraint takes advantage of the “global” structure of the elementary constraints it represents.
    - This is more effective than propagating the individual constraints

Global constraints

- Global constraints like alldiff **exploit problem structure**.
  - Filtering for a global constraint takes advantage of the “global” structure of the elementary constraints it represents.
  - This is more effective than propagating the individual constraints

- Example: alldiff(x_1,x_2,x_3) with domains

$$x_1 \in \{1,2,\ \}$$
$$x_2 \in \{1,2,\ \}$$
$$x_3 \in \{1,2,3\}$$

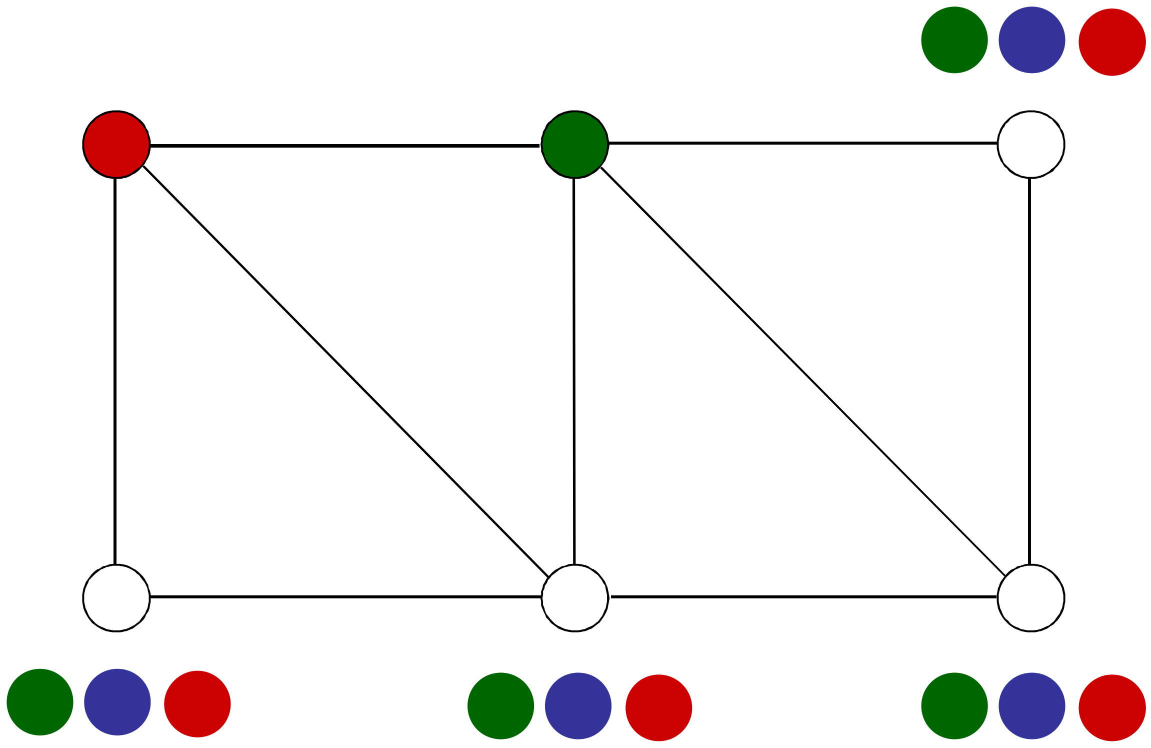
- Filtering individual constraints has no effect:

$$x_1 \neq x_2$$
$$x_1 \neq x_3$$
$$x_2 \neq x_3$$

Example: Graph Coloring

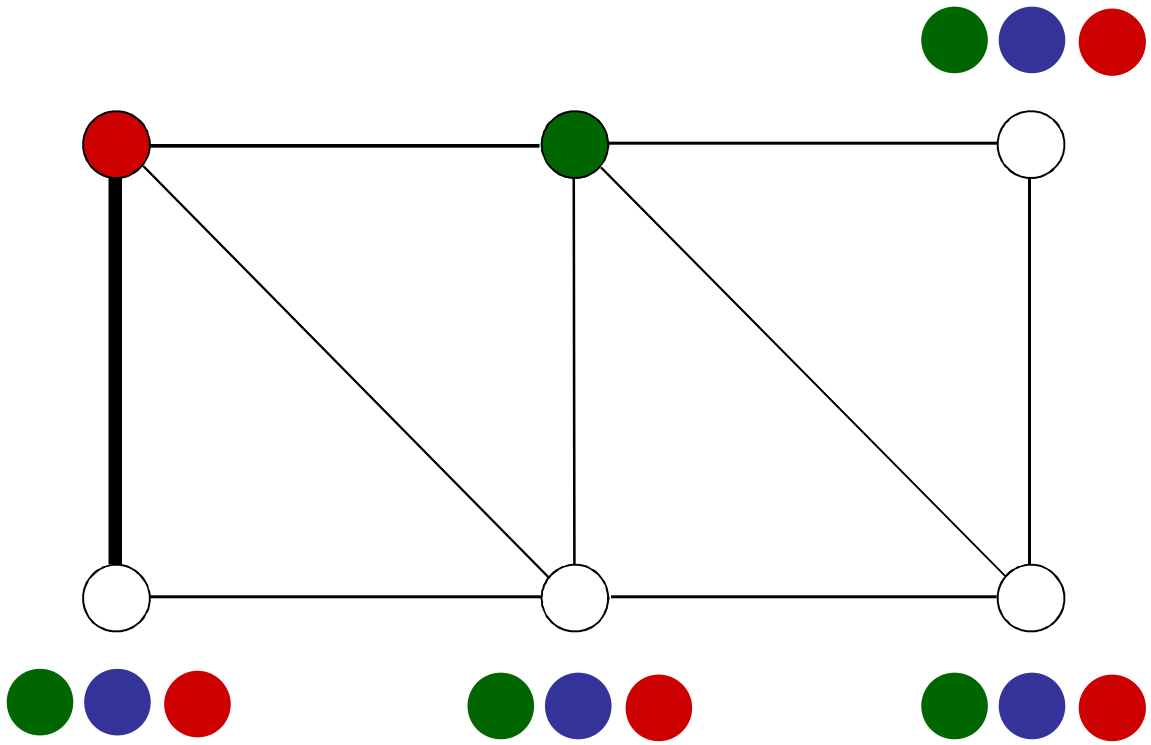
- Graph coloring problem:
  - Color vertices so that no two adjacent vertices have the same color.
  - Constraints are **binary**:
    - $x_i \neq x_j$ for each pair *i*, *j* of adjacent vertices.
    - where x_i = color of vertex *i*.

Graph coloring problem that can be solved by filtering and propagation alone. Color nodes with red, green, blue.

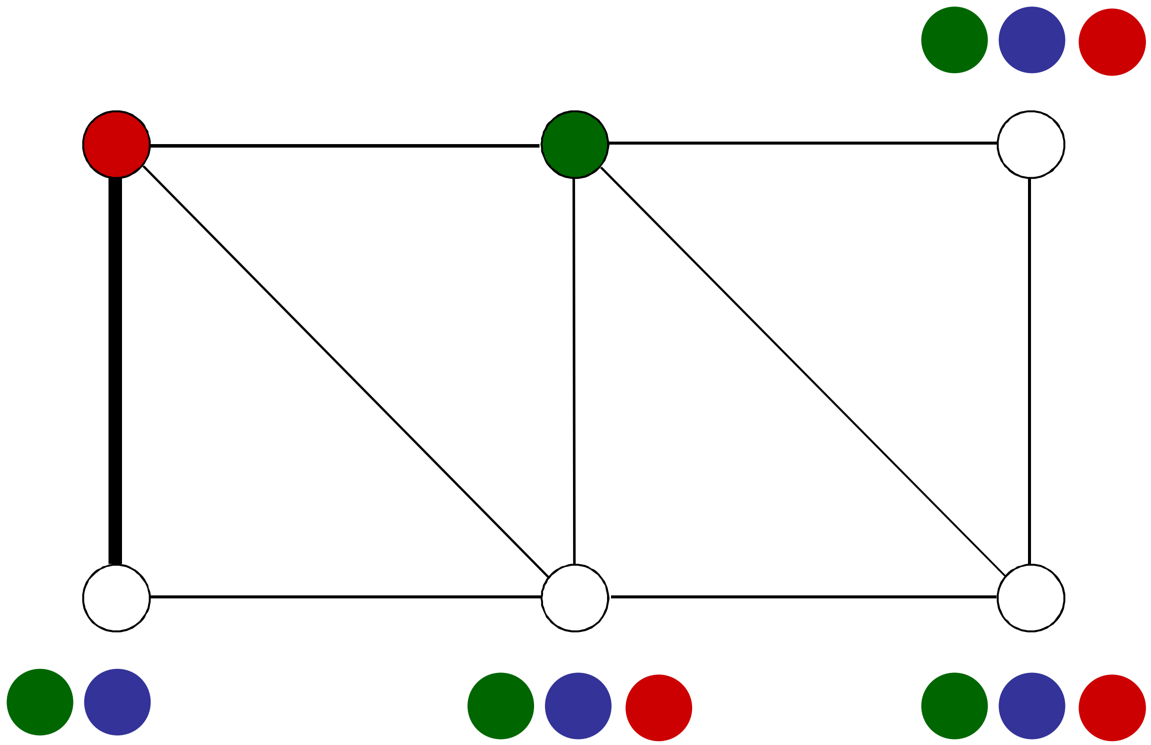


Domain of variable associated with vertex

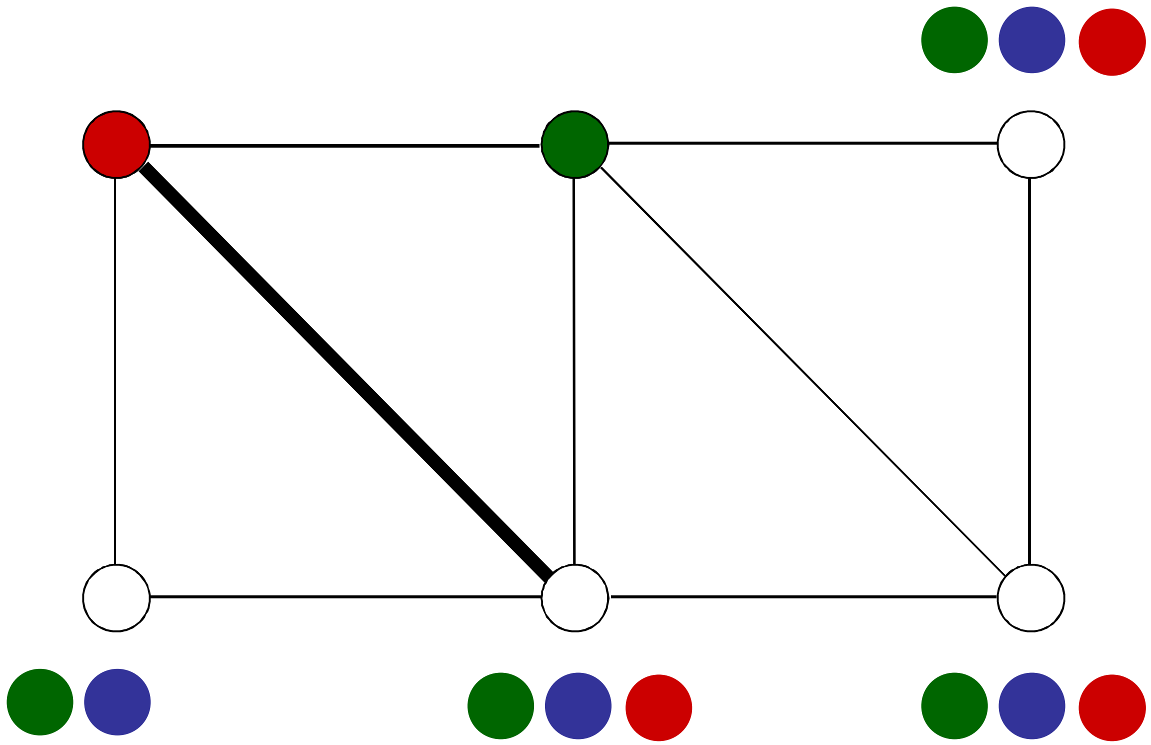
Graph coloring problem that can be solved by filtering and propagation alone. Color nodes with red, green, blue.



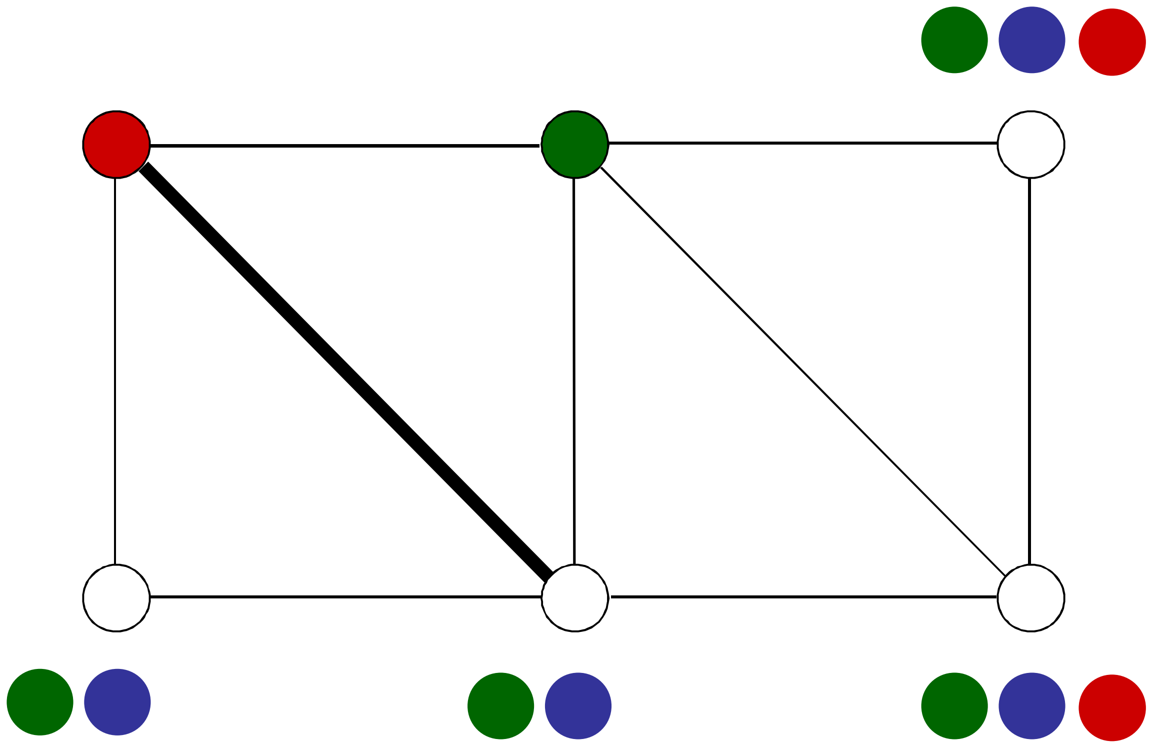
Graph coloring problem that can be solved by filtering and propagation alone. Color nodes with red, green, blue.



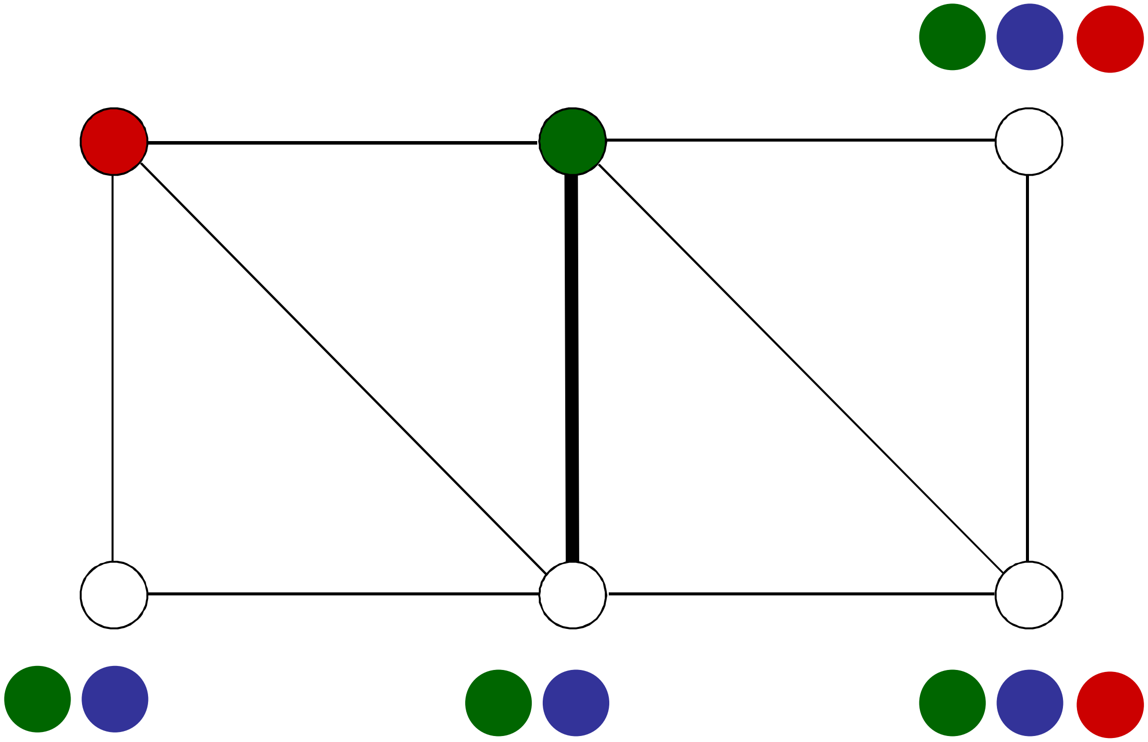
Graph coloring problem that can be solved by filtering and propagation alone. Color nodes with red, green, blue.



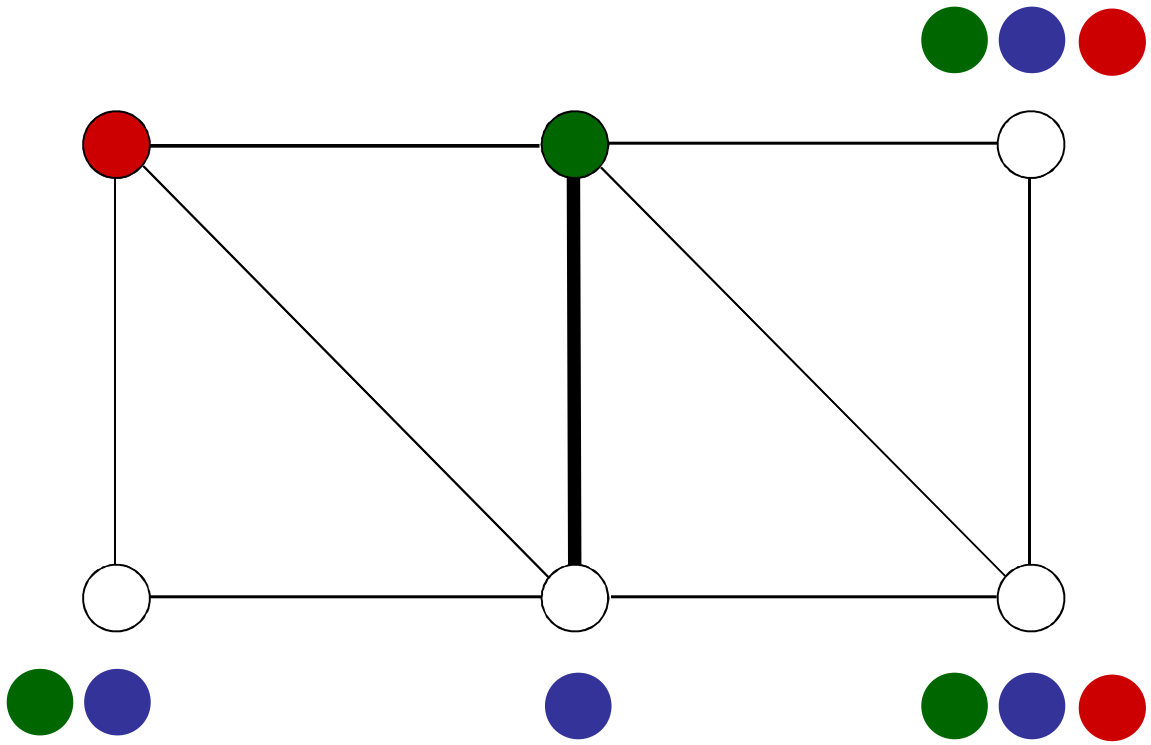
Graph coloring problem that can be solved by filtering and propagation alone. Color nodes with red, green, blue.



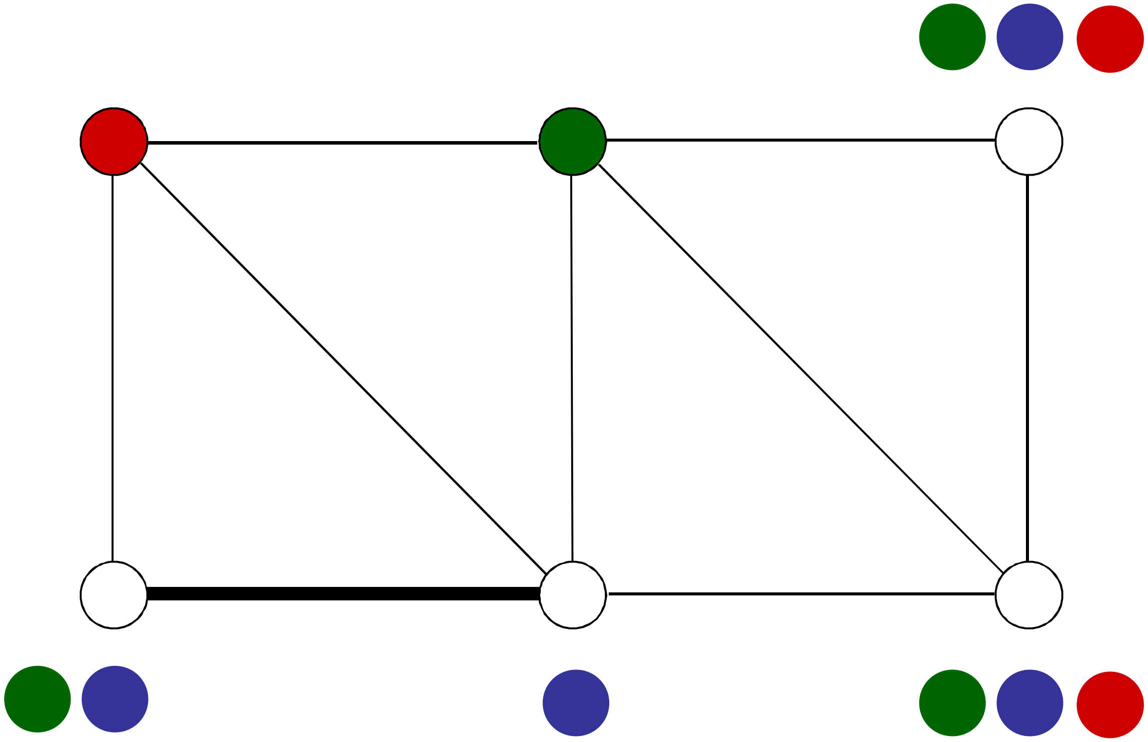
Graph coloring problem that can be solved by filtering and propagation alone. Color nodes with red, green, blue.



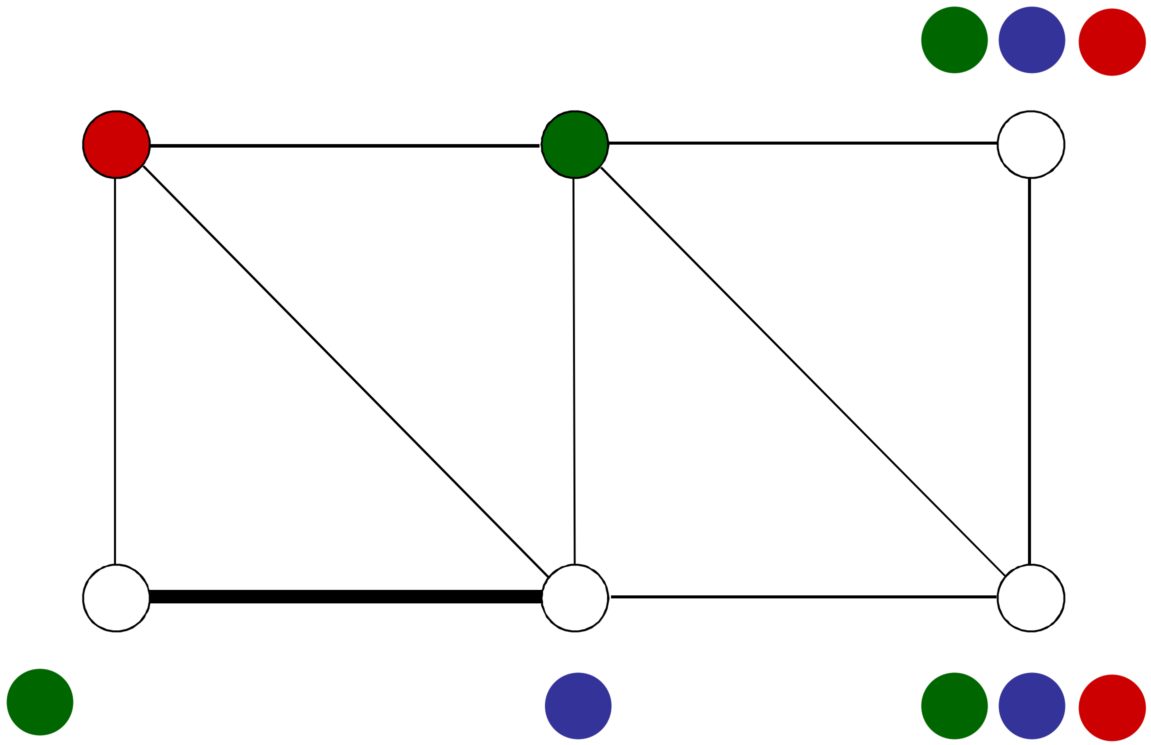
Graph coloring problem that can be solved by filtering and propagation alone. Color nodes with red, green, blue.



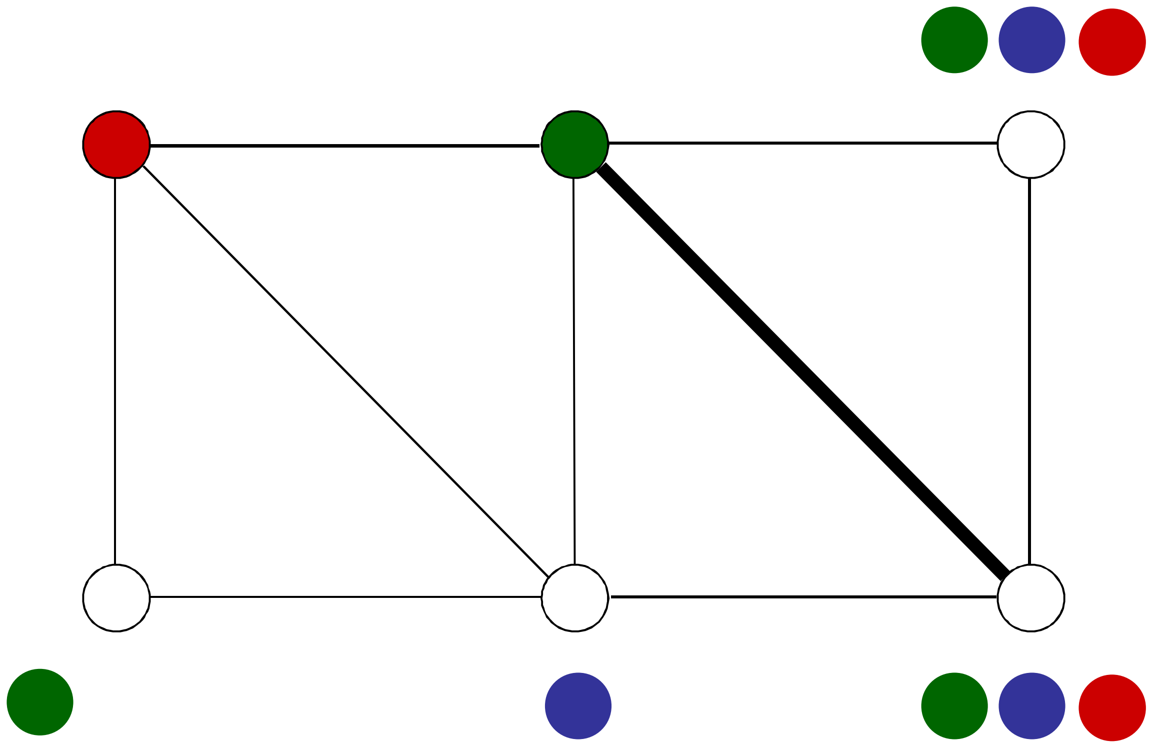
Graph coloring problem that can be solved by filtering and propagation alone. Color nodes with red, green, blue.



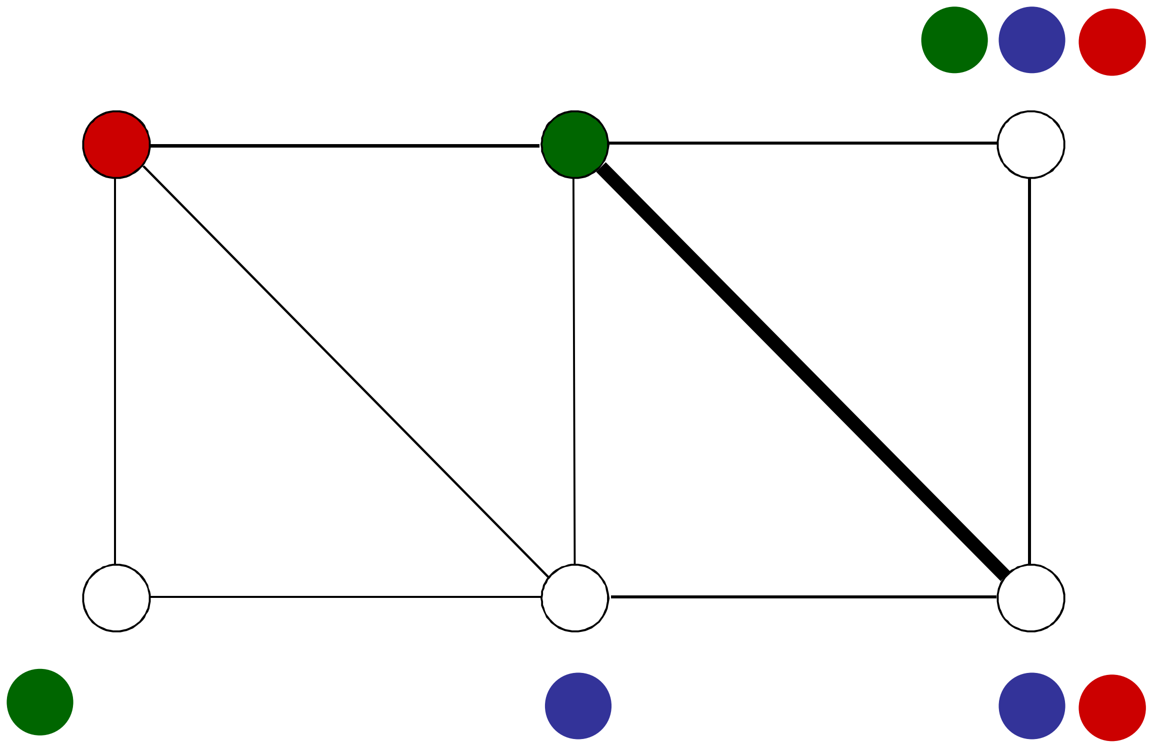
Graph coloring problem that can be solved by filtering and propagation alone. Color nodes with red, green, blue.



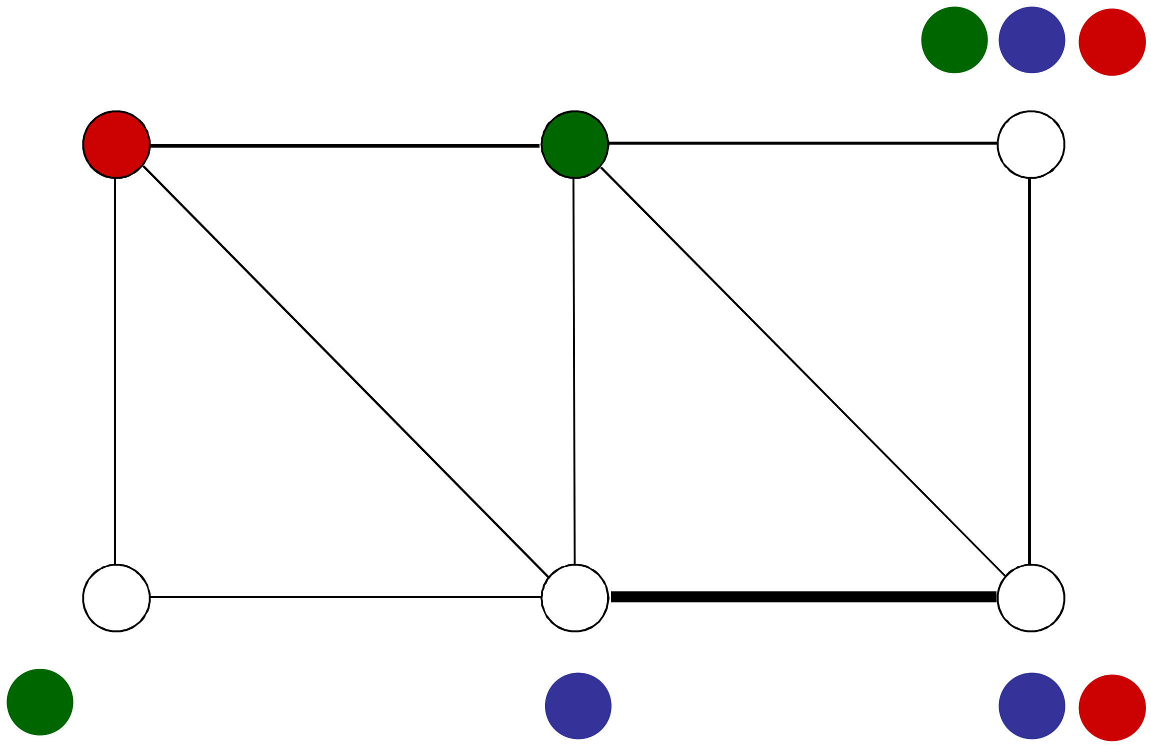
Graph coloring problem that can be solved by filtering and propagation alone. Color nodes with red, green, blue.



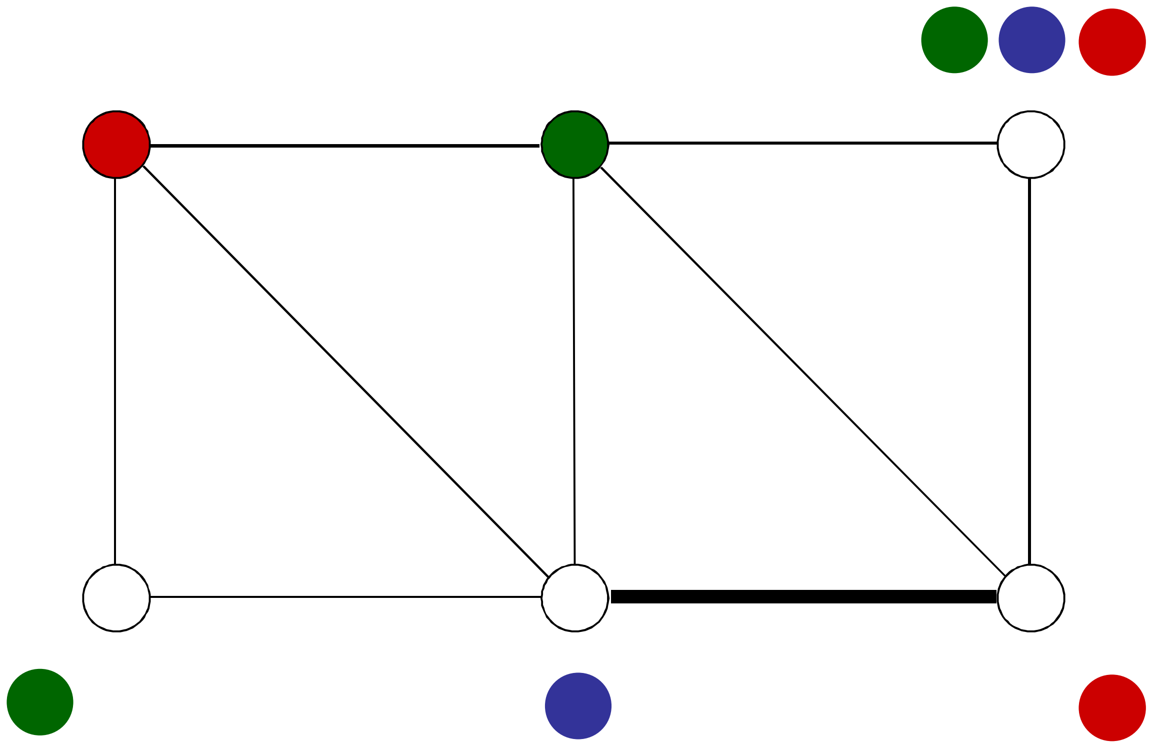
Graph coloring problem that can be solved by filtering and propagation alone. Color nodes with red, green, blue.



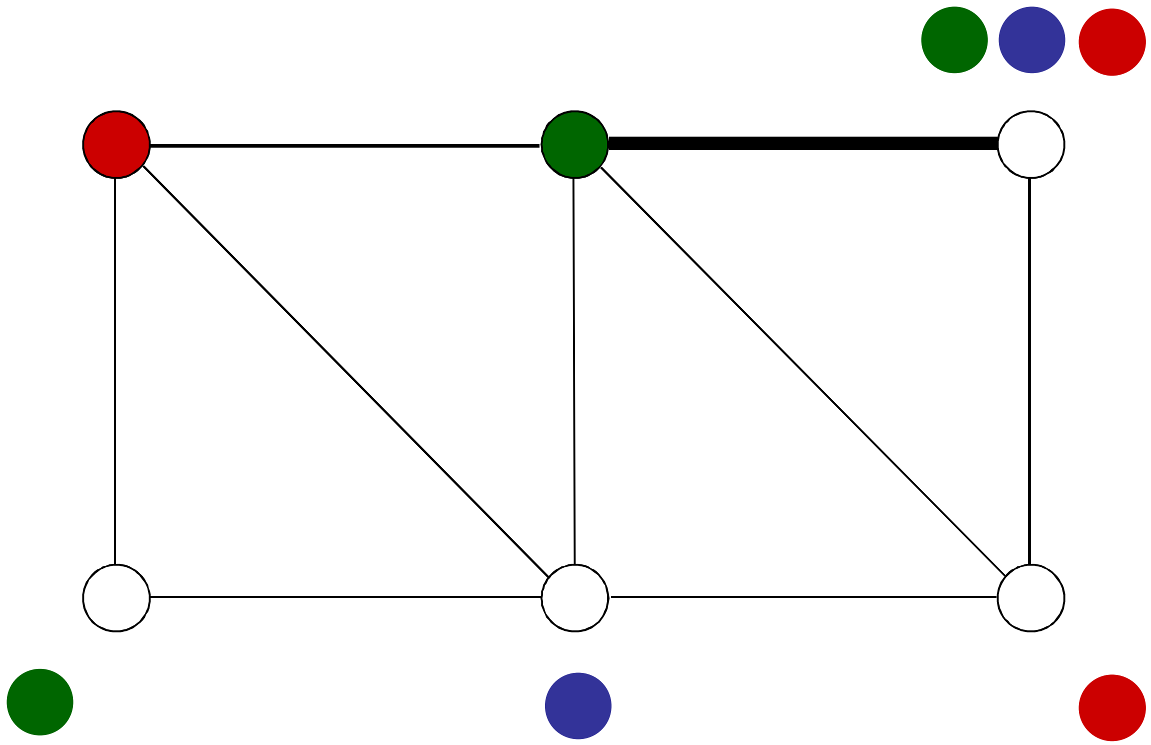
Graph coloring problem that can be solved by filtering and propagation alone. Color nodes with red, green, blue.



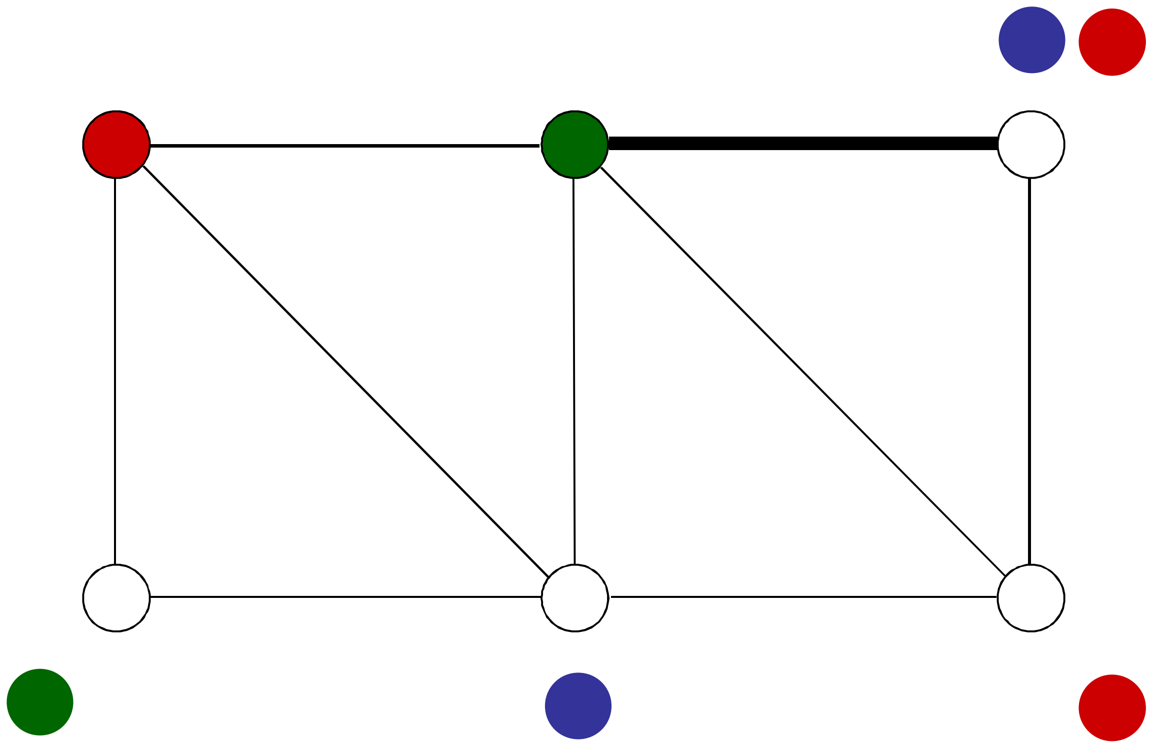
Graph coloring problem that can be solved by filtering and propagation alone. Color nodes with red, green, blue.



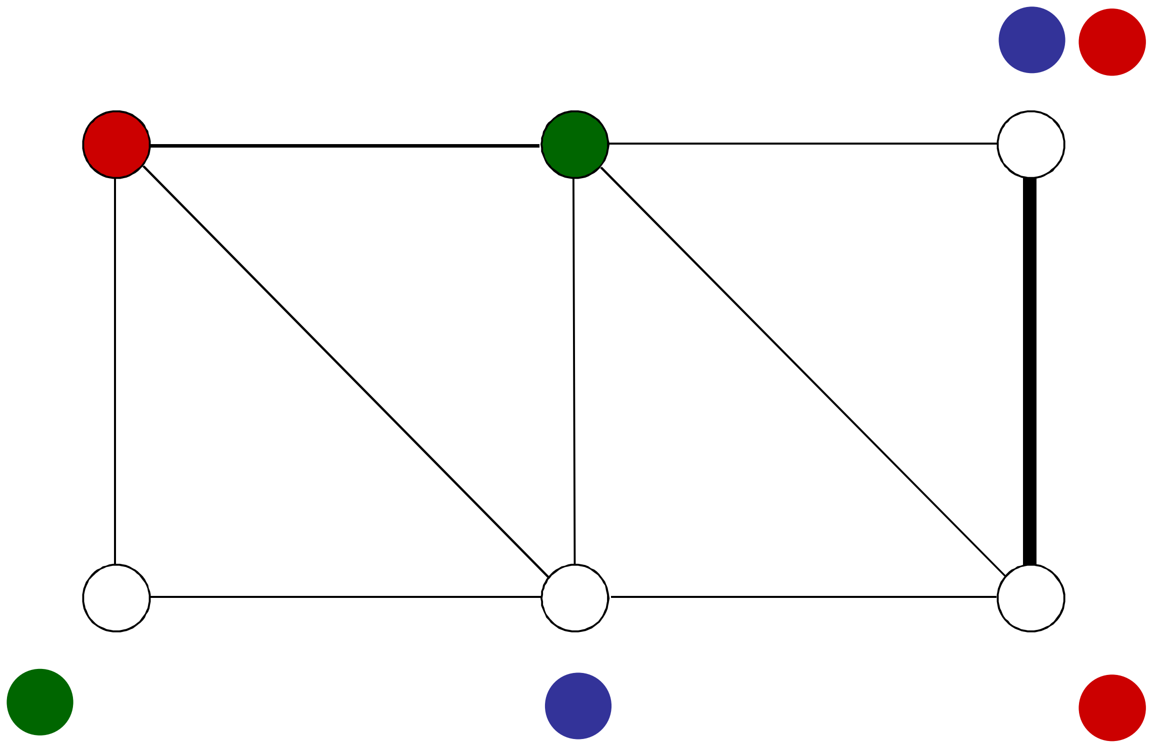
Graph coloring problem that can be solved by filtering and propagation alone. Color nodes with red, green, blue.



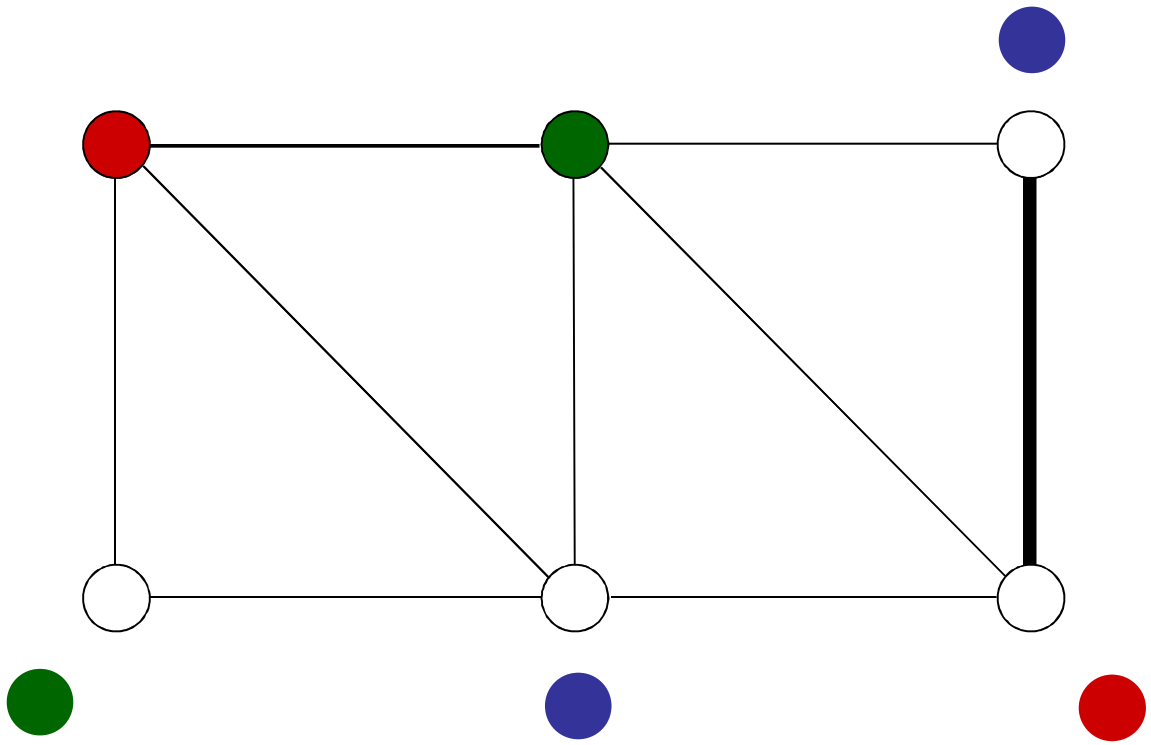
Graph coloring problem that can be solved by filtering and propagation alone. Color nodes with red, green, blue.

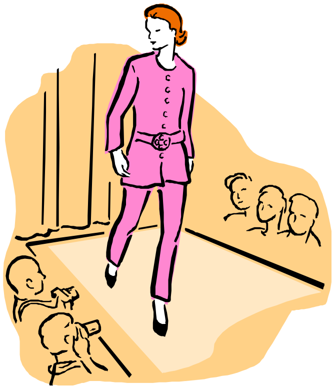


Graph coloring problem that can be solved by filtering and propagation alone. Color nodes with red, green, blue.



Graph coloring problem that can be solved by filtering and propagation alone. Color nodes with red, green, blue.





Some CP Models

Sudoku

Traveling salesman

Cumulative scheduling

Employee scheduling

Car sequencing

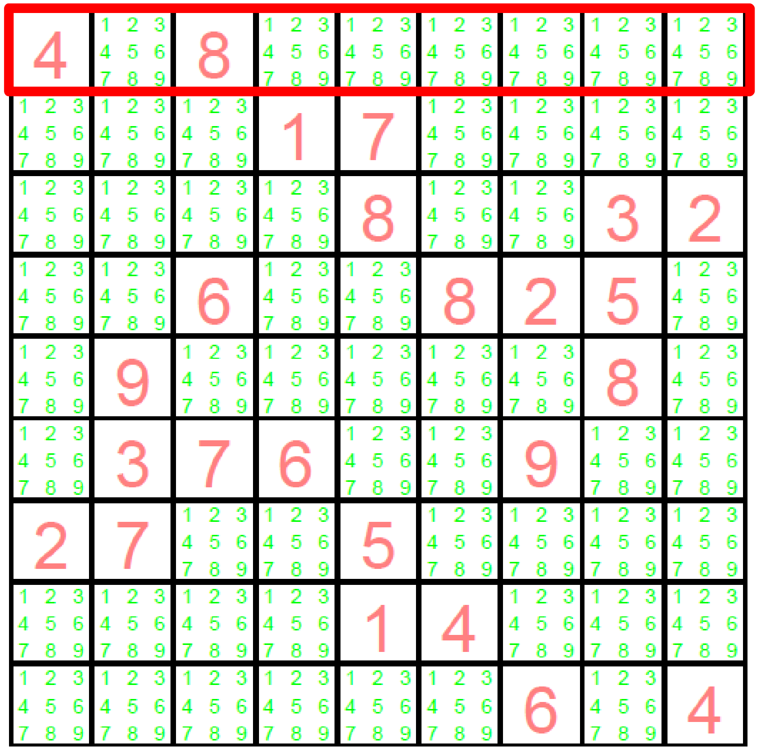
Sudoku

4	1 2 3 4 5 6 7 8 9	8	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1	7	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	8	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	3	2
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	6	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	8	2	5	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	8	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	3	7	6	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
2	7	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	5	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1	4	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	6	1 2 3 4 5 6 7 8 9	4

Fill blanks with numbers1-9.

Thanks to Helmut Simonis for this example.

Sudoku

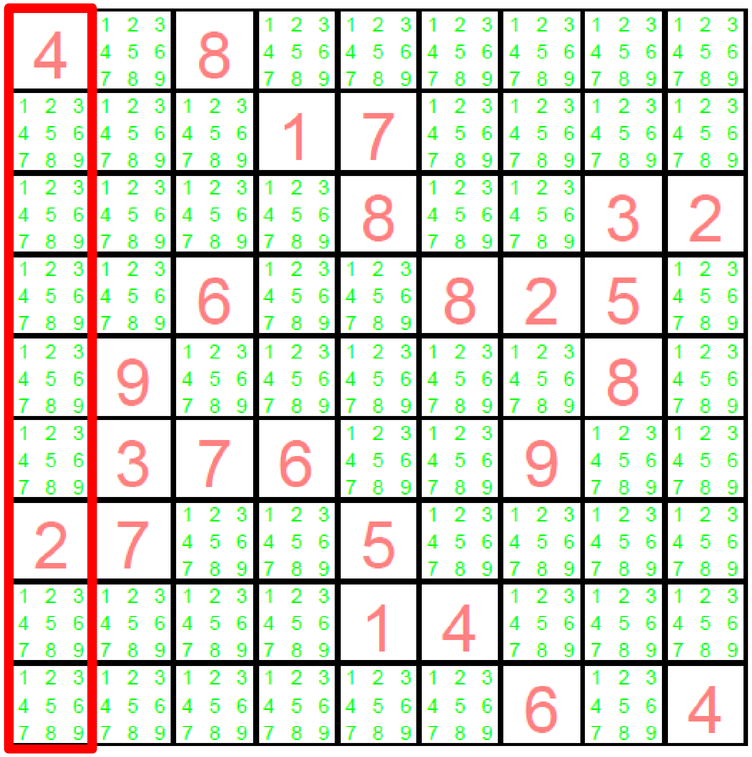


4	1 2 3 4 5 6 7 8 9	8	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1	7	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	8	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	3	2
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	6	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	8	2	5	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	8	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	3	7	6	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
2	7	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	5	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1	4	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	6	1 2 3 4 5 6 7 8 9	4

Fill blanks with numbers1-9.

Numbers all different in each row,

Sudoku



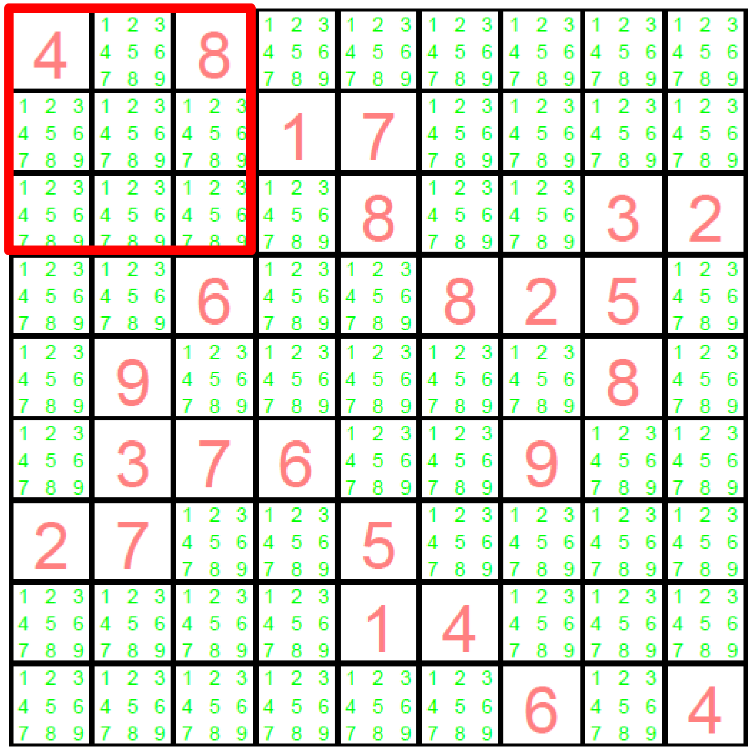
4	1 2 3 4 5 6 7 8 9	8	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1	7	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	8	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	3	2
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	6	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	8	2	5	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	8	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	3	7	6	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
2	7	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	5	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1	4	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	6	1 2 3 4 5 6 7 8 9	4

Fill blanks with numbers1-9.

Numbers all different in each row,

In each column,

Sudoku



4	1 2 3 4 5 6 7 8 9	8	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1	7	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	8	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	3	2
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	6	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	8	2	5	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	8	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	3	7	6	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
2	7	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	5	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1	4	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	6	1 2 3 4 5 6 7 8 9	4

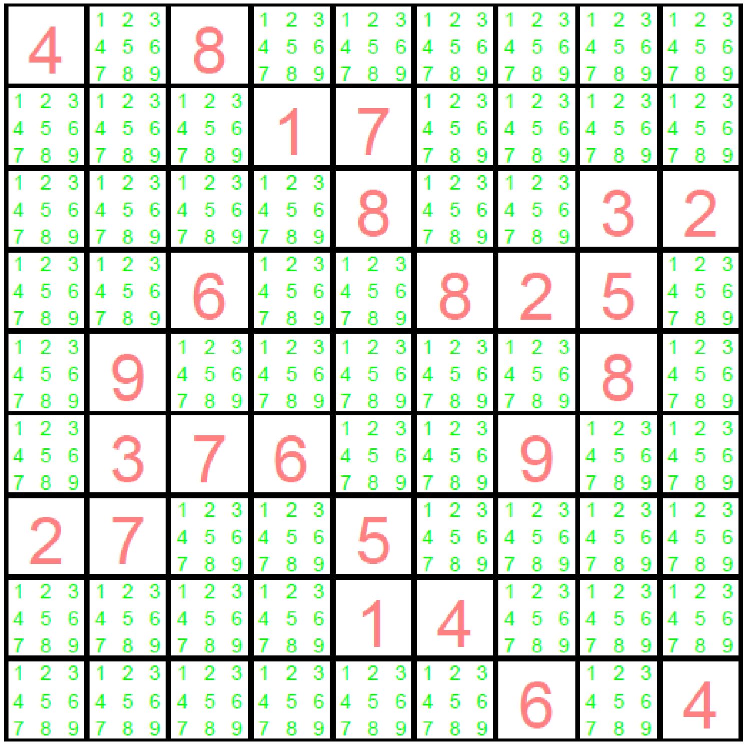
Fill blanks with numbers1-9.

Numbers all different in each row,

In each column,

And in each 3x3 square.

Sudoku



4	1 2 3 4 5 6 7 8 9	8	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1	7	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	8	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	3	2
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	6	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	8	2	5	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	8	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	3	7	6	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
2	7	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	5	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1	4	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	6	1 2 3 4 5 6 7 8 9	4

Fill blanks with numbers1-9.

Numbers all different in each row,

In each column,

And in each 3x3 square.

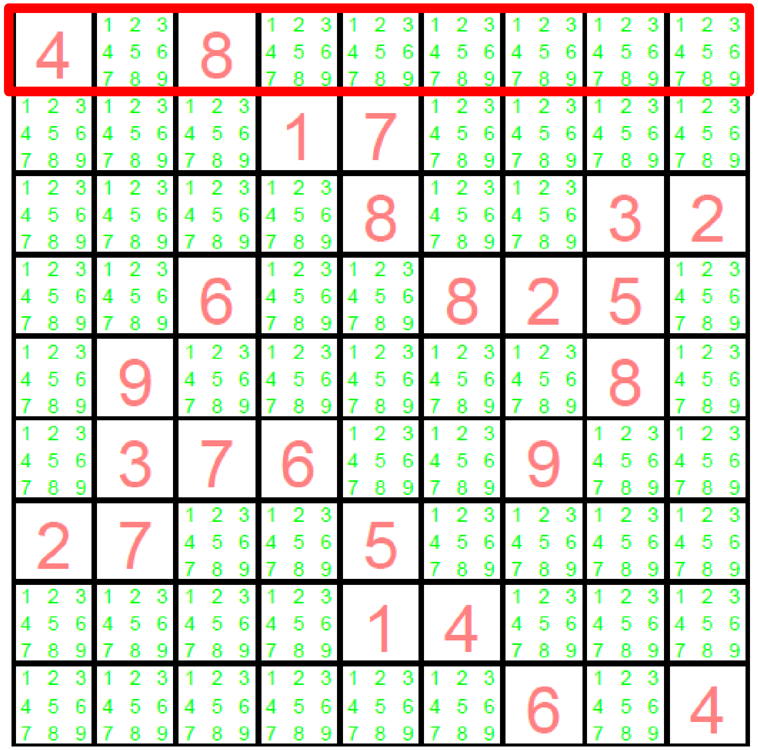
Use **alldiff** constraints!

Sudoku

4	1 2 3 4 5 6 7 8 9	8	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1	7	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	8	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	3	2
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	6	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	8	2	5	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	8	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	3	7	6	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
2	7	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	5	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1	4	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	6	1 2 3 4 5 6 7 8 9	4

Let x_{ij} = number in cell *i,j*

Sudoku

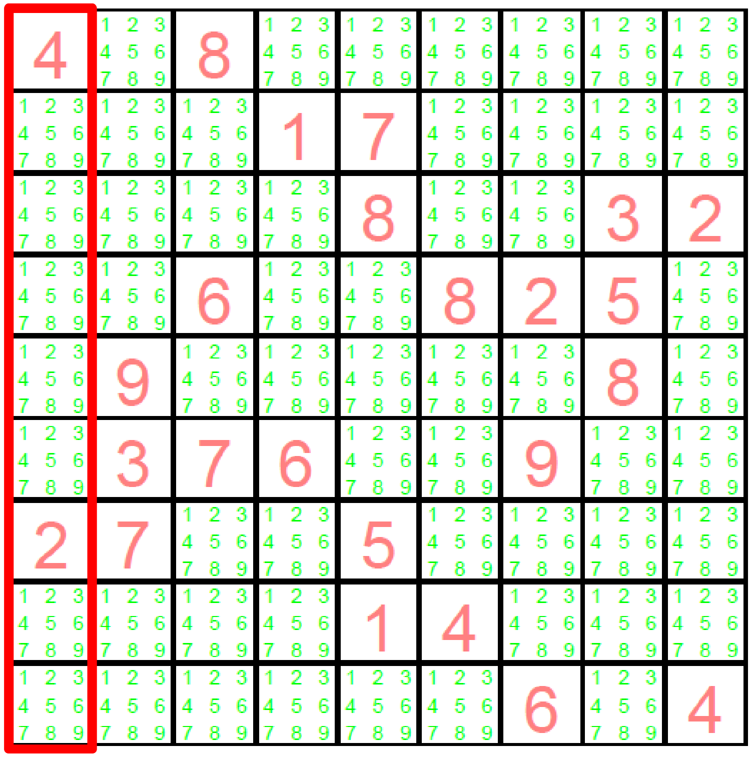


4	1 2 3 4 5 6 7 8 9	8	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1	7	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	8	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	3	2
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	6	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	8	2	5	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	8	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	3	7	6	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
2	7	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	5	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1	4	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	6	1 2 3 4 5 6 7 8 9	4

Let x_{ij} = number in cell i,j

alldiff(x_{11}, …, x_{19})

Sudoku



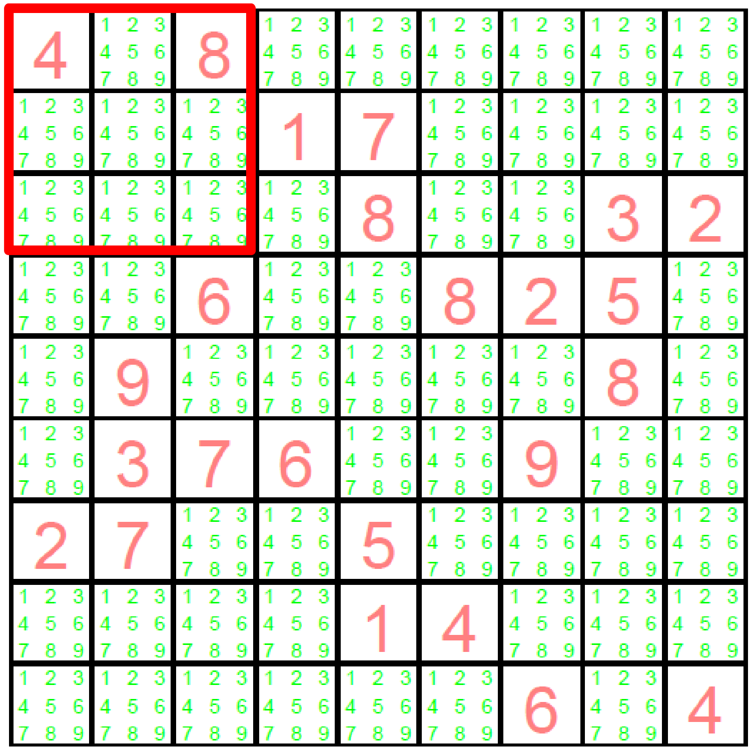
4	1 2 3 4 5 6 7 8 9	8	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1	7	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	8	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	3	2
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	6	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	8	2	5	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	8	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	3	7	6	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
2	7	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	5	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1	4	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	6	1 2 3 4 5 6 7 8 9	4

Let x_{ij} = number in cell *i,j*

alldiff(x_{11}, …, x_{19})

alldiff(x_{11}, …, x_{91})

Sudoku



4	1 2 3 4 5 6 7 8 9	8	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1	7	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	8	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	3	2
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	6	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	8	2	5	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	8	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	3	7	6	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
2	7	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	5	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1	4	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	6	1 2 3 4 5 6 7 8 9	4

Let x_{ij} = number in cell *i,j*

alldiff(x_{11}, …, x_{19})

alldiff(x_{11}, …, x_{91})

alldiff(x_{11},x_{12},x_{13},x_{21},x_{22},x_{23},x_{31},x_{32},x_{33})

etc.

Sudoku

4	1 2 3 4 5 6 7 8 9	8	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1	7	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	8	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	3	2
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	6	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	8	2	5	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	8	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	3	7	6	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
2	7	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	5	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1	4	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9
1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	1 2 3 4 5 6 7 8 9	6	1 2 3 4 5 6 7 8 9	4

Solution

4	2	8	5	6	3	1	7	9
3	5	9	1	7	2	4	6	8
7	6	1	4	8	9	5	3	2
1	4	6	3	9	8	2	5	7
5	9	2	7	4	1	3	8	6
8	3	7	6	2	5	9	4	1
2	7	4	9	5	6	8	1	3
6	8	3	2	1	4	7	9	5
9	1	5	8	3	7	6	2	4

Sudoku

How to solve it?

Filtering, propagation, and branching (see demonstration).

Solve it first with very simple filtering (forward checking) that only checks for constraint violations.

Then solve it with complete filter for the alldiffs.

Solution

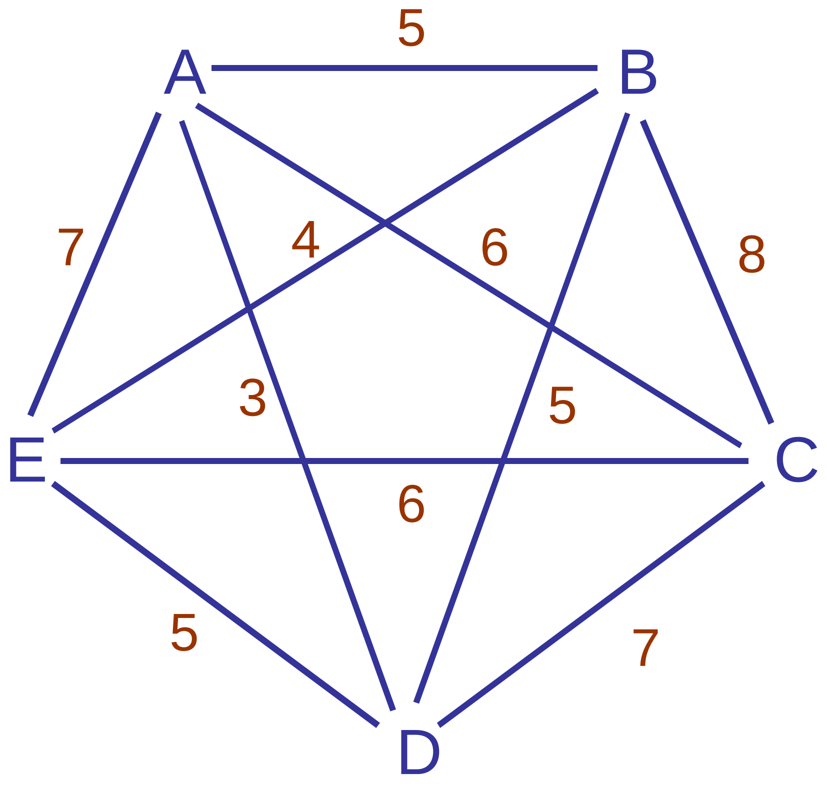
4	2	8	5	6	3	1	7	9
3	5	9	1	7	2	4	6	8
7	6	1	4	8	9	5	3	2
1	4	6	3	9	8	2	5	7
5	9	2	7	4	1	3	8	6
8	3	7	6	2	5	9	4	1
2	7	4	9	5	6	8	1	3
6	8	3	2	1	4	7	9	5
9	1	5	8	3	7	6	2	4

Traveling Salesman

Traveling salesman problem:

Let c_{ij} = distance from city *i* to city *j*.

Find the shortest route that visits each of *n* cities exactly once.



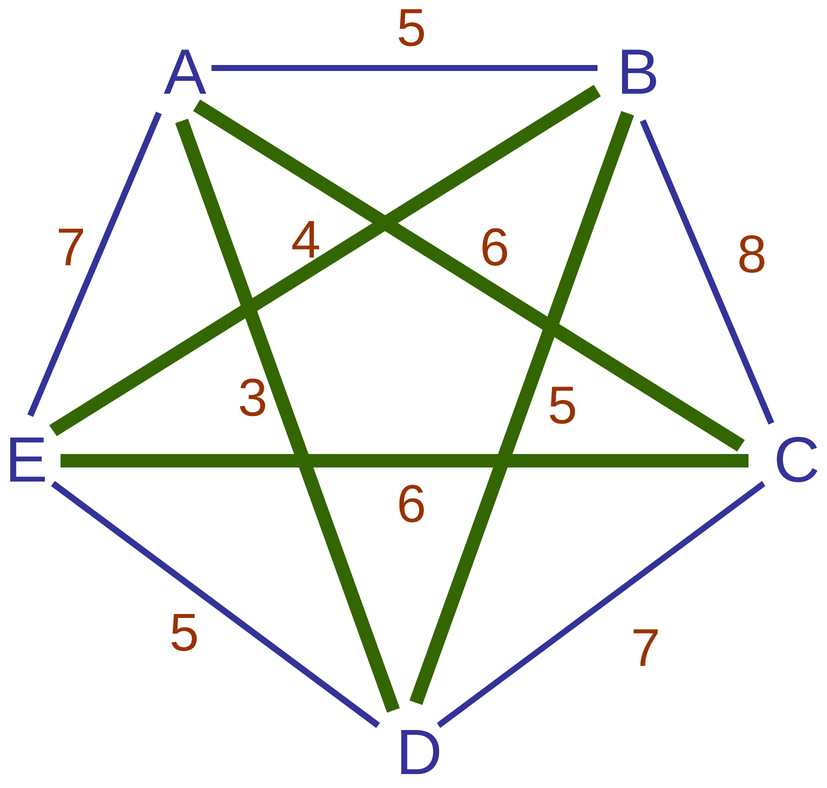
Traveling Salesman

Traveling salesman problem:

Let c_{ij} = distance from city *i* to city *j*.

Find the shortest route that visits each of *n* cities exactly once.

Optimal tour:



Popular 0-1 model

Let x_{ij} = 1 if city *i* immediately precedes city *j*, 0 otherwise

$$
\begin{aligned}
\text{min} \quad & \sum_{ij} c_{ij} x_{ij} \\
\text{s.t.} \quad & \sum_{i} x_{ij} = 1, \quad \text{all } j \\
& \sum_{j} x_{ij} = 1, \quad \text{all } i \\
& \sum_{i \in V} \sum_{j \in W} x_{ij} \geq 1, \quad \text{all disjoint } V, W \subset \{1, \ldots, n\} \\
& x_{ij} \in \{0,1\}
\end{aligned}
$$

Subtour elimination constraints

CP model

Let y_k = the *k*th city visited.

Variable indices

$$\begin{aligned} \text{min} \quad & \sum_k c_{y_k y_{k+1}} \\ \text{s.t.} \quad & \text{alldiff}(y_1, \ldots, y_n) \\ & y_k \in \{1, \ldots, n\} \end{aligned}$$

In objective function, identify city n + 1 with city 1.

An alternate CP model

Let y_k = the city visited after city k.

$$\begin{aligned} \min \quad & \sum_k c_{ky_k} \\ \text{s.t.} \quad & \text{circuit}(y_1,\ldots,y_n) \\ & y_k \in \{1,\ldots,n\} \end{aligned}$$

Hamiltonian circuit constraint

Element constraint

The constraint $c_y \leq 5$ can be implemented:

$$z \leq 5$$

$$\text{element}\left(y,(c_1,\ldots,c_n),z\right)$$

Assign z the yth value in the list

The constraint $x_y \leq 5$ can be implemented

$$z \leq 5$$

$$\text{element}\left(y,(x_1,\ldots,x_n),z\right)$$

Add the constraint $z = x_y$

(this is a slightly different constraint)

Cumulative scheduling

- Used for resource-constrained scheduling.
- Total resources consumed by jobs at any one time must not exceed *L*.

$$\text{cumulative}\big((t_1,\ldots,t_n),(p_1,\ldots,p_n),(c_1,\ldots,c_n),L\big)$$

Job start times (variables)

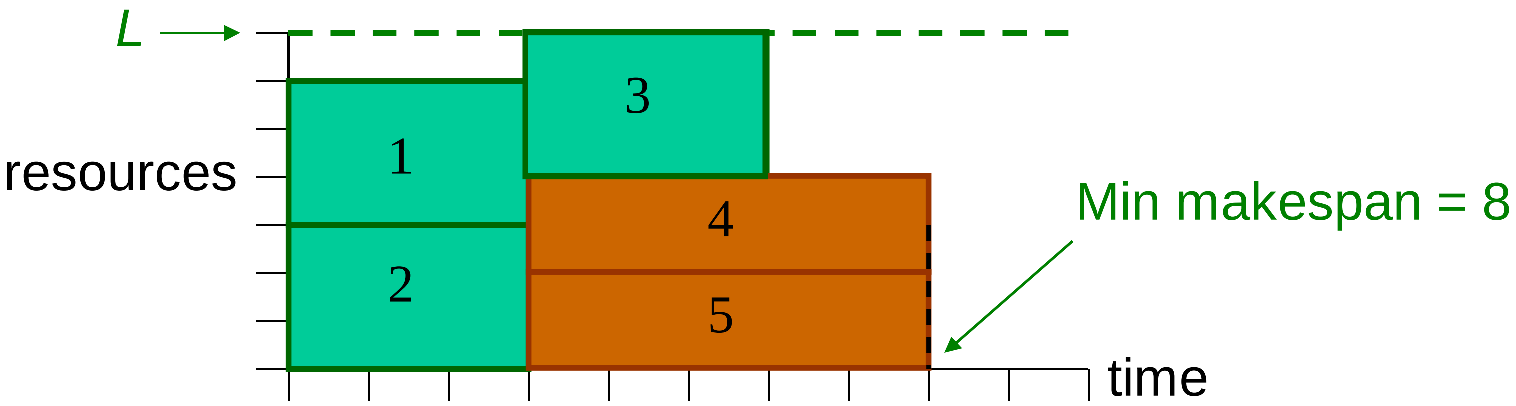
Job processing times

Job resource requirements

- Time windows (if any) indicated by domains of t_i.

Cumulative scheduling

Minimize makespan (no deadlines, all release times = 0):



$$\begin{aligned}
&\text{min} \quad z \\
&\text{s.t.} \quad \text{cumulative}\big((t_1,\ldots,t_5),(3,3,3,5,5),(3,3,3,2,2),7\big) \\
&\qquad\; z \geq t_1 + 3 \\
&\qquad\; \vdots \\
&\qquad\; z \geq t_5 + 2
\end{aligned}$$

L

Resources used

Processing times

Job start times

Example: Ship loading

- The problem
  - Examples is from OPL manual.
  - Load 34 items on the ship in minimum time (min makespan)
  - Each item requires a certain time and certain number of workers.
  - Total of 8 workers available.

Problem data

Item	Duration	Labor
1	3	4
2	4	4
3	4	3
4	6	4
5	5	5
6	2	5
7	3	4
8	4	3
9	3	4
10	2	8
11	3	4
12	2	5
13	1	4
14	5	3
15	2	3
16	3	3
17	2	6
18	2	7
19	1	4
20	1	4
21	1	4
22	2	4
23	4	7
24	5	8
25	2	8
26	1	3
27	1	3
28	2	6
29	1	8
30	3	3
31	2	3
32	1	3
33	2	3
34	2	3

Precedence constraints

1 → 2,4

2 →3

3 →5,7

4 →5

5 →6

6 →8

7 →8

8 →9

9 →10

9 →14

10 →11

10 →12

11 →13

12 →13

13 →15,16

14 →15

15 →18

16 →17

17 →18

18 →19

18 →20,21

19 →23

20 → 23

21 → 22

22 →23

23 →24

24 →25

25 →26,30,31,32

26 → 27

27 → 28

28 → 29

30 → 28

31 → 28

32 → 33

33 → 34

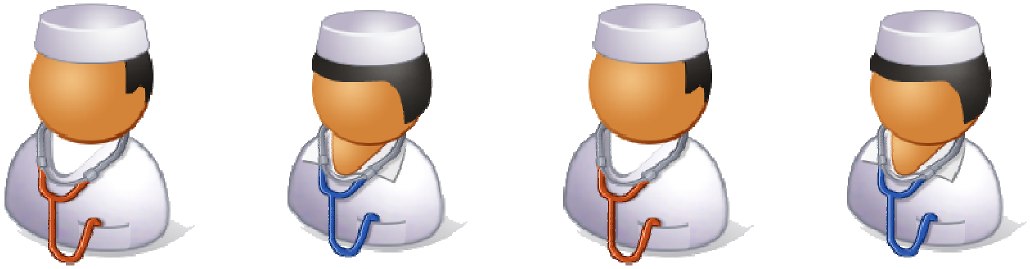
Use the cumulative scheduling constraint.

$$
\begin{aligned}
\text{min} \quad & z \\
\text{s.t.} \quad & z \geq t_1 + 3, \quad z \geq t_2 + 4, \quad \text{etc.} \\
& \text{cumulative}\big((t_1,\ldots,t_{34}),(3,4,\ldots,2),(4,4,\ldots,3),8\big) \\
& t_2 \geq t_1 + 3, \quad t_4 \geq t_1 + 3, \quad \text{etc.}
\end{aligned}
$$

Precedence constraints

Employee scheduling

- Schedule four nurses in 8-hour shifts.
- A nurse works at most one shift a day, at least 5 days a week.
- Same schedule every week.
- No shift staffed by more than two different nurses in a week.
- A nurse cannot work different shifts on two consecutive days.
- A nurse who works shift 2 or 3 must do so at least two days in a row.



Two ways to view the problem

Assign nurses to shifts

	Sun	Mon	Tue	Wed	Thu	Fri	Sat
Shift 1	A	B	A	A	A	A	A
Shift 2	C	C	C	B	B	B	B
Shift 3	D	D	D	D	C	C	D

Assign shifts to nurses

	Sun	Mon	Tue	Wed	Thu	Fri	Sat
Nurse A	1	0	1	1	1	1	1
Nurse B	0	1	0	2	2	2	2
Nurse C	2	2	2	0	3	3	0
Nurse D	3	3	3	3	0	0	3

0 = day off

Use **both** formulations in the same model!

First, assign nurses to shifts.

Let w_{sd} = nurse assigned to shift *s* on day *d*

$$\text{alldiff}(w_{1d}, w_{2d}, w_{3d}), \quad \text{all } d \longleftarrow$$

The variables w_{1d}, w_{2d}, w_{3d} take different values

That is, schedule 3 different nurses on each day

Use **both** formulations in the same model!

First, assign nurses to shifts.

Let w_{sd} = nurse assigned to shift *s* on day *d*

$$\text{alldiff}\left(w_{1d}, w_{2d}, w_{3d}\right), \quad \text{all } d$$

$$\text{cardinality}\left(w \mid (A,B,C,D),(5,5,5,5),(6,6,6,6)\right)$$

A occurs at least 5 and at most 6 times in the array *w*, and similarly for B, C, D.

That is, each nurse works at least 5 and at most 6 days a week

Use **both** formulations in the same model!

First, assign nurses to shifts.

Let w_{sd} = nurse assigned to shift *s* on day *d*

$$\text{alldiff}\left(w_{1d}, w_{2d}, w_{3d}\right), \quad \text{all } d$$

$$\text{cardinality}\left(w \mid (A,B,C,D),(5,5,5,5),(6,6,6,6)\right)$$

$$\text{nvalues}\left(w_{s,\text{Sun}},\ldots,w_{s,\text{Sat}} \mid 1,2\right), \quad \text{all } s$$

The variables $w_{s,\text{Sun}}, \ldots, w_{s,\text{Sat}}$ take at least 1 and at most 2 different values.

That is, at least 1 and at most 2 nurses work any given shift.

Remaining constraints are not easily expressed in this notation.

So, assign shifts to nurses.

Let y_{id} = shift assigned to nurse *i* on day *d*

$$\text{alldiff}\left(y_{1d}, y_{2d}, y_{3d}\right),\ \ \text{all } d$$

Assign a different nurse to each shift on each day.

This constraint is redundant of previous constraints, but redundant constraints speed solution.

Remaining constraints are not easily expressed in this notation.

So, assign shifts to nurses.

Let y_{id} = shift assigned to nurse *i* on day *d*

$$\text{alldiff}\left(y_{1d}, y_{2d}, y_{3d}\right),\ \text{all } d$$
$$\text{stretch}\left(y_{i,\text{Sun}}, \ldots, y_{i,\text{Sat}} \mid (2,3),(2,2),(6,6),P\right),\ \text{all } i$$

Every stretch of 2’s has length between 2 and 6.
Every stretch of 3’s has length between 2 and 6.

So a nurse who works shift 2 or 3 must do so at least two days in a row.

Remaining constraints are not easily expressed in this notation.

So, assign shifts to nurses.

Let y_{id} = shift assigned to nurse *i* on day *d*

$$\text{alldiff}\left(y_{1d}, y_{2d}, y_{3d}\right),\ \ \text{all } d$$
$$\text{stretch}\left(y_{i,\text{Sun}}, \ldots, y_{i,\text{Sat}} \mid (2,3),(2,2),(6,6),P\right),\ \ \text{all } i$$

Here $P = \{(s,0),(0,s) \mid s = 1,2,3\}$

Whenever a stretch of *a*'s immediately precedes a stretch of *b*'s, (*a*,*b*) must be one of the pairs in *P*.

So a nurse cannot switch shifts without taking at least one day off.

Now we must connect the w_{sd} variables to the y_{id} variables.

Use **channeling constraints**:

$$w_{y_{id}d} = i, \quad \text{all } i, d$$

$$y_{w_{sd}d} = s, \quad \text{all } s, d$$

Channeling constraints increase propagation and make the problem easier to solve.

The complete model is:

$$\begin{aligned}
&\text{alldiff}\left(w_{1d}, w_{2d}, w_{3d}\right), \quad \text{all } d \\
&\text{cardinality}\left(w \mid (A, B, C, D), (5,5,5,5), (6,6,6,6)\right) \\
&\text{nvalues}\left(w_{s,\text{Sun}}, \ldots, w_{s,\text{Sat}} \mid 1, 2\right), \quad \text{all } s
\end{aligned}$$

$$\begin{aligned}
&\text{alldiff}\left(y_{1d}, y_{2d}, y_{3d}\right), \quad \text{all } d \\
&\text{stretch}\left(y_{i,\text{Sun}}, \ldots, y_{i,\text{Sat}} \mid (2,3), (2,2), (6,6), P\right), \quad \text{all } i
\end{aligned}$$

$$w_{y_{id}d} = i, \quad \text{all } i, d$$

$$y_{w_{sd}d} = s, \quad \text{all } s, d$$

Car sequencing

- An assembly line produces cars with 2 options.
  - Air conditioning and sun roof.
  - Four types of cars, each with an output requirement.

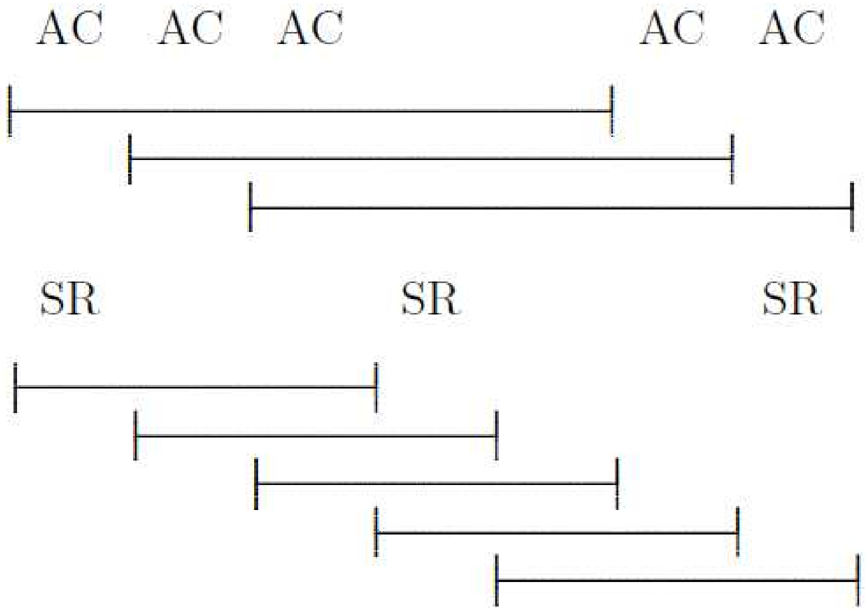
Car type	Number	AC option	SR option
a	1	0	0
b	3	1	0
c	1	0	1
d	2	1	1

- At most 3 cars in every sequence of 5 can have AC
- At most 1 car in every sequence of 3 can have SR.
- How to sequence the cars?

Car sequencing

A feasible solution

Position j:	1	2	3	4	5	6	7
Car model x_j:	d	b	b	c	a	b	d



Type	Num.	AC	SR
a	1	0	0
b	3	1	0
C	1	0	1
D	2	1	1

Car sequencing

We will use the **sequence** constraint:

$$\text{sequence}\big((y_1,\ldots,y_n),q,\ell,u\big)$$

Requires that at least ℓ and at most u ones occur in every sequence of q consecutive binary variables y_i.

Car sequencing

CP model:

$$\text{cardinality}\big((x_1,\ldots,x_7),(a,b,c,d),(1,3,1,2),(1,3,1,2)\big)$$
$$\text{element}\big(x_i,(0,1,0,1),y_i\big)$$
$$\text{element}\big(x_i,(0,0,1,1),z_i\big)$$
$$\text{sequence}\big((y_1,\ldots,y_7),5,0,3\big)$$
$$\text{sequence}\big((z_1,\ldots,z_7),3,0,1\big)$$
$$\boxed{x_i} \in \{a,b,c,d\},\quad \boxed{y_i},\boxed{z_i} \in \{0,1\}$$

= 1 if SR in position *i*

Car type in position *i*

= 1 if AC in position *i*

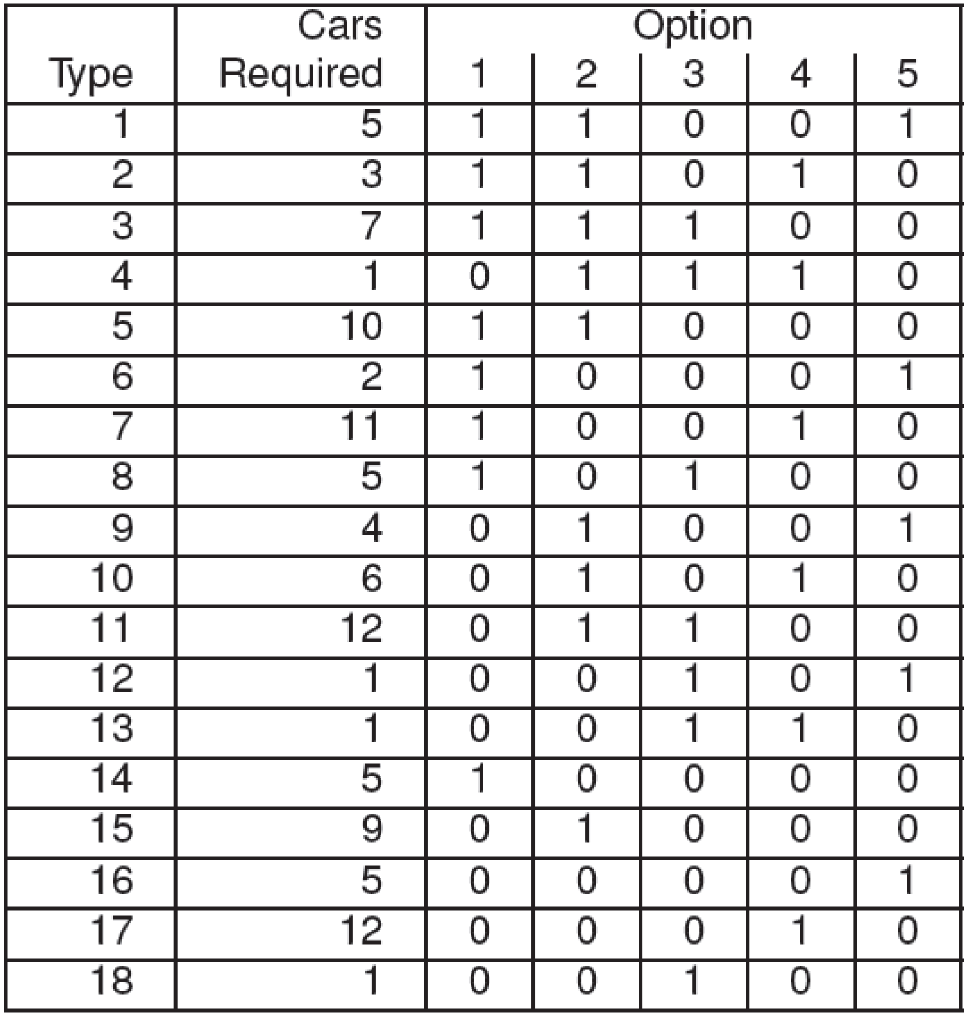
Type	Num.	AC	SR
a	1	0	0
b	3	1	0
C	1	0	1
D	2	1	1

Car sequencing

A larger instance:

Sequence constraints

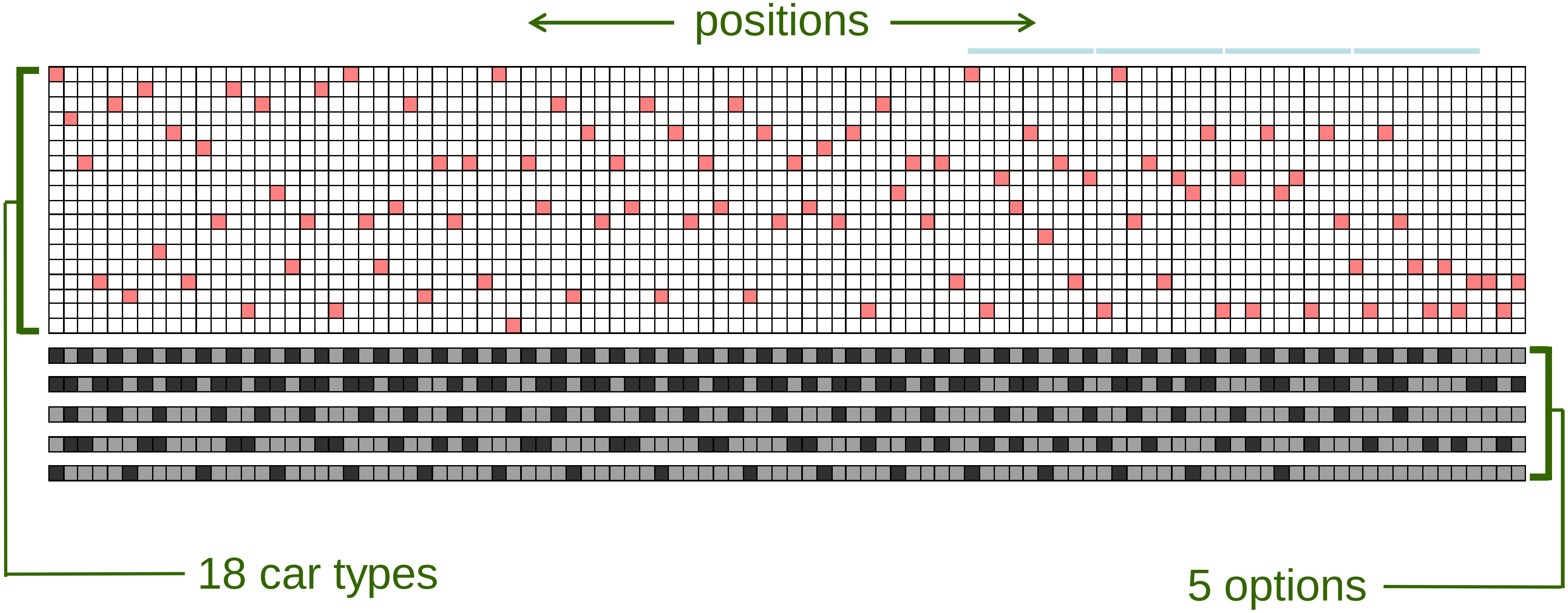
Option 1: ≤ 1 out of 2
Option 2: ≤ 2 out of 3
Option 3: ≤ 1 out of 3
Option 4: ≤ 2 out of 5
Option 5: ≤ 1 out of 5



	Cars	Option				
Type	Required	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

Car sequencing

A solution:



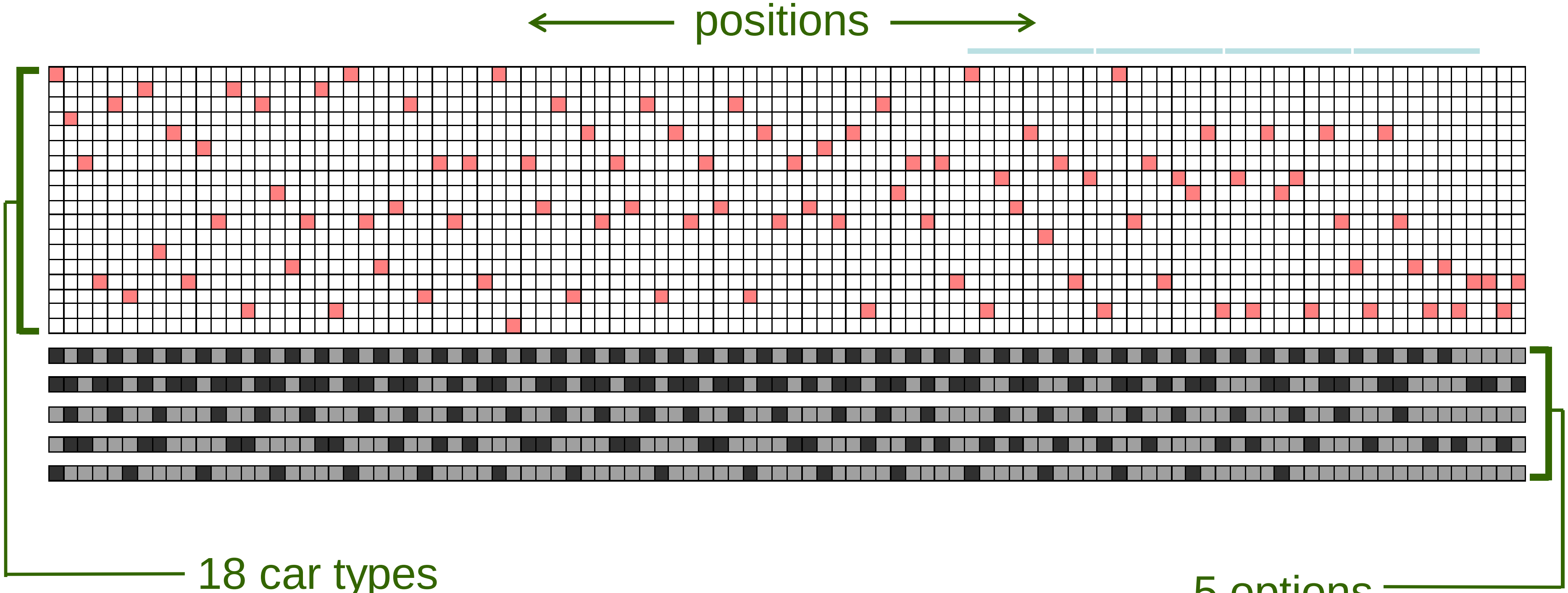
positions

18 car types

5 options

Car sequencing

A solution:



18 car types

5 options

Solve by filtering, propagation and branching (see demonstration)

Consistency

Domain Consistency

Bounds Consistency

k-consistency and Backtracking

Domain Consistency

- A constraint set is **domain consistent** if every value in every variable domain is consistent with the constraints.
    - That is, each domain value occurs in some feasible solution.

Domain Consistency

- A constraint set is **domain consistent** if every value in every variable domain is consistent with the constraints.
    - That is, each domain value occurs in some feasible solution.
    - For each x_i and each value v in the domain of x_i, some $x = (x_1,\ldots,x_n)$ with $x_i = v$ satisfies the constraint set.

Domain Consistency

- A constraint set is **domain consistent** if every value in every variable domain is consistent with the constraints.
    - That is, each domain value occurs in some feasible solution.
    - For each x_i and each value v in the domain of x_i, some $x = (x_1,\ldots,x_n)$ with $x_i = v$ satisfies the constraint set.
- Equivalent terms:
    - Hyperarc consistency, generalized arc consistency.

Domain Consistency

- A constraint set is **domain consistent** if every value in every variable domain is consistent with the constraints.
    - That is, each domain value occurs in some feasible solution.
    - For each x_i and each value v in the domain of x_i, some $x = (x_1,\ldots,x_n)$ with $x_i = v$ satisfies the constraint set.
- Equivalent terms:
    - Hyperarc consistency, generalized arc consistency.
- To achieve domain consistency:
    - **Filter** inconsistent values from the domains.

Domain consistency

Consider the constraint set

$$\begin{aligned} x_1 + x_{100} &\geq 1 \\ x_1 - x_{100} &\geq 0 \\ x_1, x_{100} &\in \{0,1\} \end{aligned}$$

The solutions are $(x_1, x_{100}) = (1,0), (1,1)$.

Domain consistency

Consider the constraint set

$$\begin{aligned} x_1 + x_{100} &\geq 1 \\ x_1 - x_{100} &\geq 0 \\ x_1, x_{100} &\in \{0,1\} \end{aligned}$$

The solutions are $(x_1, x_{100}) = (1,0), (1,1)$.

It is **not** domain consistent, because $x_1 = 0$ is infeasible.
No solution has $x_1 = 0$.

Domain consistency

Consider the constraint set

$$\begin{aligned} x_1 + x_{100} &\geq 1 \\ x_1 - x_{100} &\geq 0 \\ x_1 \in \{1\},\ \ x_{100} &\in \{0,1\} \end{aligned}$$

The solutions are $(x_1, x_{100}) = (1,0), (1,1)$.

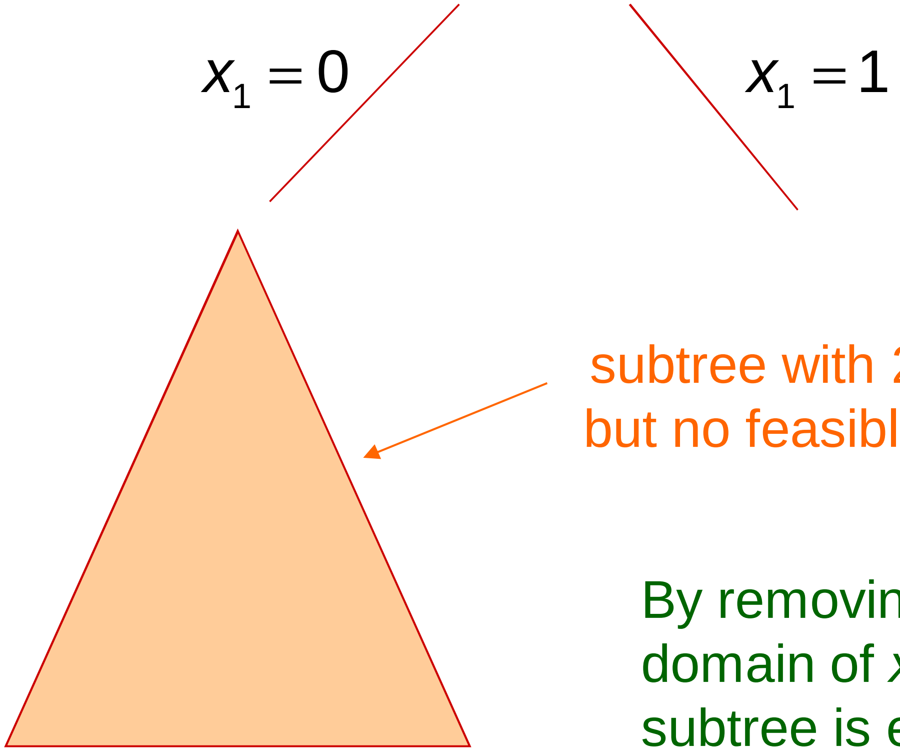
It is **not** domain consistent, because $x_1 = 0$ is infeasible.
No solution has $x_1 = 0$.

Filtering 1 from the domain of x_1 achieves domain consistency.

Domain consistency

Domain consistency can reduce branching.

$$
\begin{aligned}
&x_1 + x_{100} \geq 1 \\
&x_1 - x_{100} \geq 1 \\
&\text{other constraints} \\
&x_j \in \{0,1\}
\end{aligned}
$$



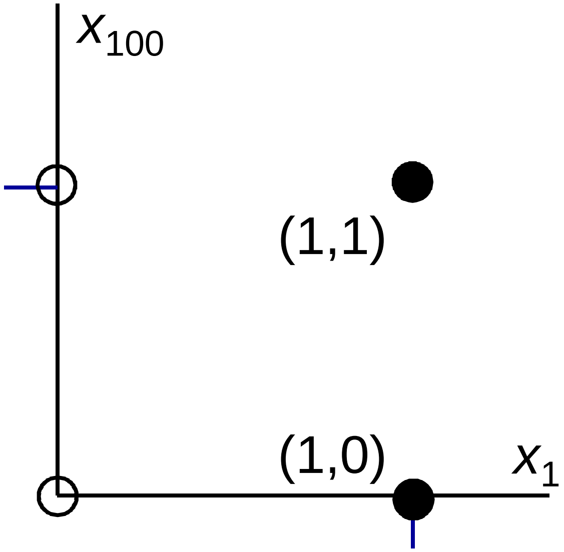
subtree with 2^{99} nodes but no feasible solution

By removing 0 from the domain of x_1, the left subtree is eliminated

Domain consistency and projection

A constraint set is domain consistent if the domain of each variable x_i is the projection of the feasible set onto x_i.

$$x_1 + x_{100} \geq 1$$
$$x_1 - x_{100} \geq 0$$
$$x_1, x_{100} \in \{0,1\}$$

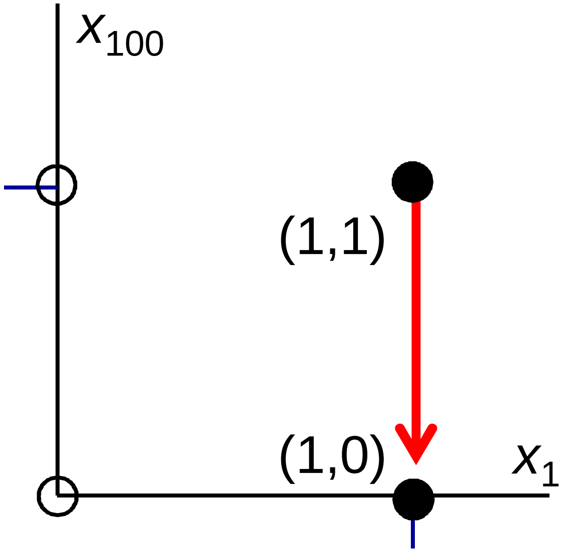


Domain consistency and projection

A constraint set is domain consistent if the domain of each variable x_i is the projection of the feasible set onto x_i.

$$x_1 + x_{100} \geq 1$$
$$x_1 - x_{100} \geq 0$$
$$x_1, x_{100} \in \{0,1\}$$

Projection onto $x_1 = \{1\}$



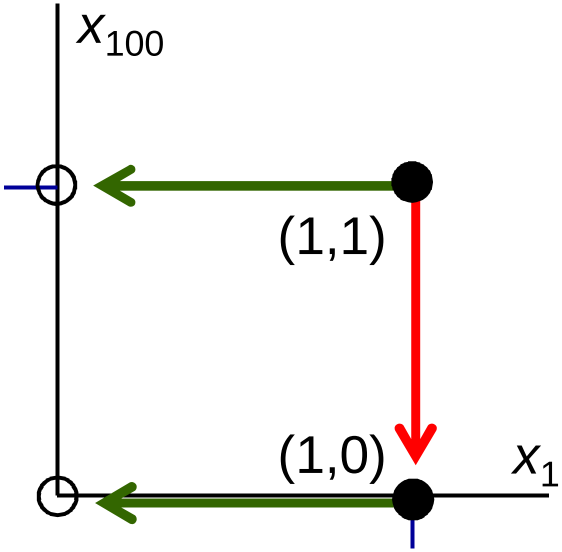
Domain consistency and projection

A constraint set is domain consistent if the domain of each variable x_i is the projection of the feasible set onto x_i.

$$x_1 + x_{100} \geq 1$$
$$x_1 - x_{100} \geq 0$$
$$x_1, x_{100} \in \{0,1\}$$

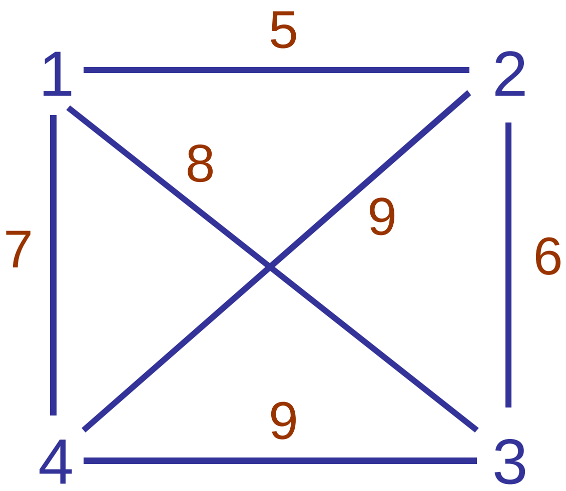
Projection onto $x_1 = \{1\}$

Projection onto $x_{100} = \{0,1\}$



Domain consistency

- Example: Traveling salesman.



$$\min \sum_{j=1}^{4} c_{jx_j} \le 28$$

$$\text{circuit}\left(x_1, x_2, x_3, x_4\right)$$

$$x_1 \in \{2,3,4\}$$

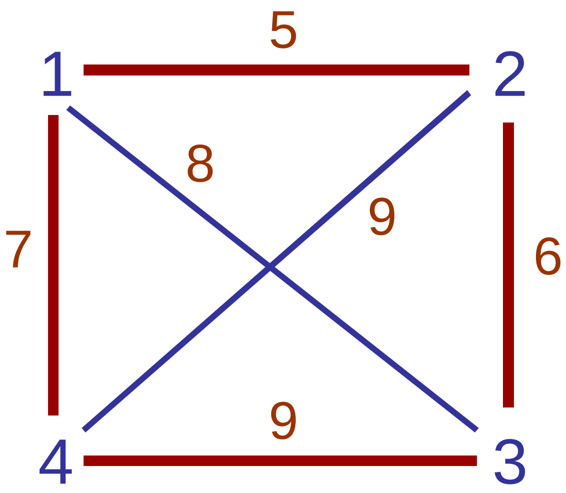
$$x_2 \in \{1,3,4\}$$

$$x_3 \in \{1,2,4\}$$

$$x_4 \in \{1,2,3\}$$

Domain consistency

- Example: Traveling salesman.



$$\min \sum_{j=1}^{4} c_{jx_j} \leq 28$$

$$\text{circuit}(x_1, x_2, x_3, x_4)$$

$$x_1 \in \{2,3,4\}$$

$$x_2 \in \{1,3,4\}$$

$$x_3 \in \{1,2,4\}$$

$$x_4 \in \{1,2,3\}$$

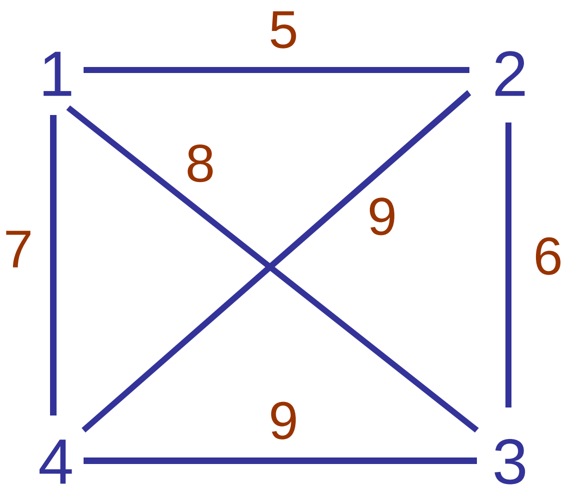
Two feasible solutions:

$(x_1,x_2,x_3,x_4) = (2,3,4,1)$

$(x_1,x_2,x_3,x_4) = (4,1,2,3)$

Domain consistency

- Example: Traveling salesman.



$$\min \sum_{j=1}^{4} c_{jx_j} \le 28$$

$$\text{circuit}\left(x_1, x_2, x_3, x_4\right)$$

$$x_1 \in \{2,3,4\}$$

$$x_2 \in \{1,3,4\}$$

$$x_3 \in \{1,2,4\}$$

$$x_4 \in \{1,2,3\}$$

Two feasible solutions:

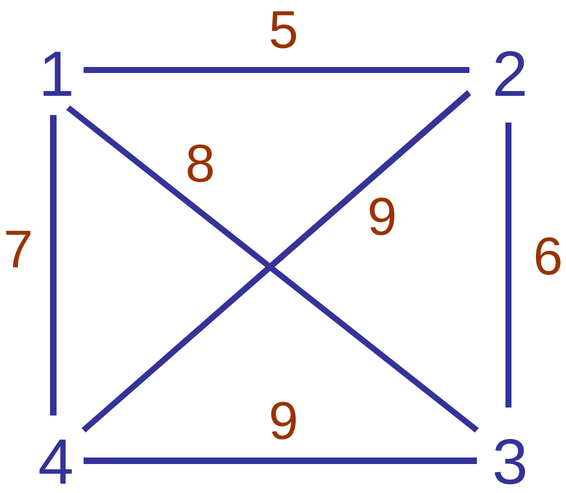
$(x_1,x_2,x_3,x_4) = (2,3,4,1)$

$(x_1,x_2,x_3,x_4) = (4,1,2,3)$

For domain consistency: compute projection onto each variable.

Domain consistency

- Example: Traveling salesman.



$$\min \sum_{j=1}^{4} c_{jx_j} \leq 28$$

$$\text{circuit}\left(x_1, x_2, x_3, x_4\right)$$

$$x_1 \in \{2,4\}$$

$$x_2 \in \{1,3,4\}$$

$$x_3 \in \{1,2,4\}$$

$$x_4 \in \{1,2,3\}$$

Two feasible solutions:

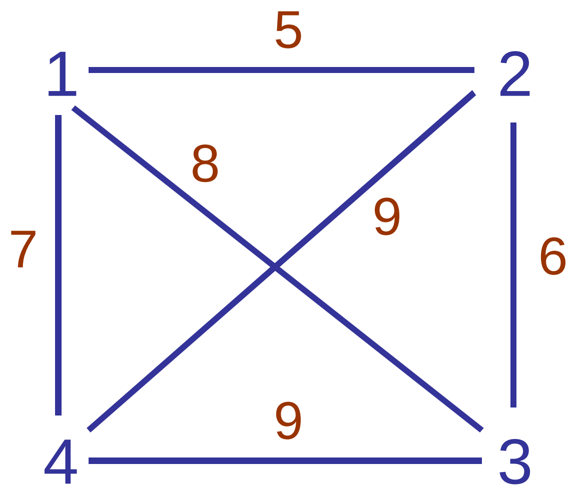
$(x_1,x_2,x_3,x_4) = (2,3,4,1)$

$(x_1,x_2,x_3,x_4) = (4,1,2,3)$

For domain consistency: compute projection onto each variable.

Domain consistency

- Example: Traveling salesman.



$$\min \sum_{j=1}^{4} c_{jx_j} \leq 28$$

$$\text{circuit}(x_1, x_2, x_3, x_4)$$

$$x_1 \in \{2,4\}$$

$$x_2 \in \{1,3\}$$

$$x_3 \in \{1,2,4\}$$

$$x_4 \in \{1,2,3\}$$

Two feasible solutions:

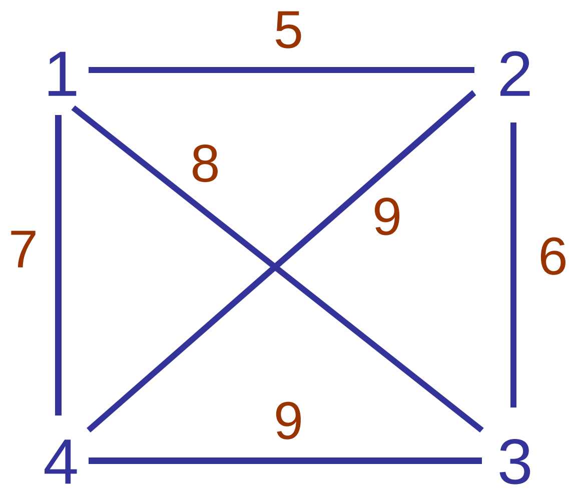
$(x_1,x_2,x_3,x_4) = (2,3,4,1)$

$(x_1,x_2,x_3,x_4) = (4,1,2,3)$

For domain consistency: compute projection onto each variable.

Domain consistency

- Example: Traveling salesman.



$$\min \sum_{j=1}^{4} c_{jx_j} \leq 28$$

$$\text{circuit}(x_1, x_2, x_3, x_4)$$

$$x_1 \in \{2,4\}$$

$$x_2 \in \{1,3\}$$

$$x_3 \in \{2,4\}$$

$$x_4 \in \{1,2,3\}$$

Two feasible solutions:

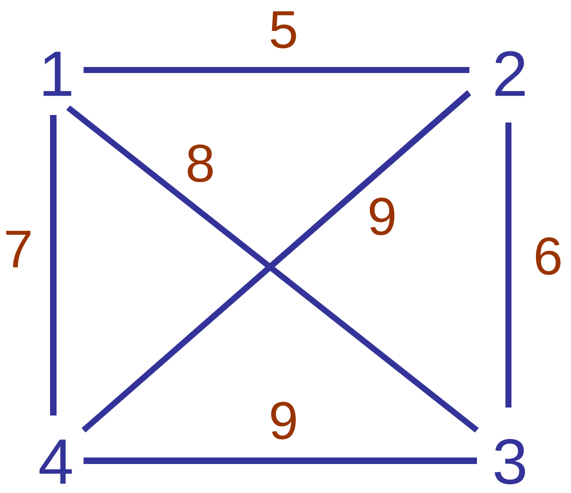
$(x_1,x_2,x_3,x_4) = (2,3,4,1)$

$(x_1,x_2,x_3,x_4) = (4,1,2,3)$

For domain consistency: compute projection onto each variable.

Domain consistency

- Example: Traveling salesman.



$$\min \sum_{j=1}^{4} c_{jx_j} \leq 28$$

$$\text{circuit}(x_1, x_2, x_3, x_4)$$

$$x_1 \in \{2,4\}$$

$$x_2 \in \{1,3\}$$

$$x_3 \in \{2,4\}$$

$$x_4 \in \{1,3\}$$

Two feasible solutions:

$(x_1,x_2,x_3,x_4) = (2,3,4,1)$

$(x_1,x_2,x_3,x_4) = (4,1,2,3)$

For domain consistency: compute projection onto each variable.

Bounds consistency

- A constraint set is **bounds consistent** if the **min** and **max** of each variable domain appear in some feasible solution, assuming the other domains are replaced by interval relaxations.
    - Interval relaxation of {2,4,7} is [2,7].

Bounds consistency

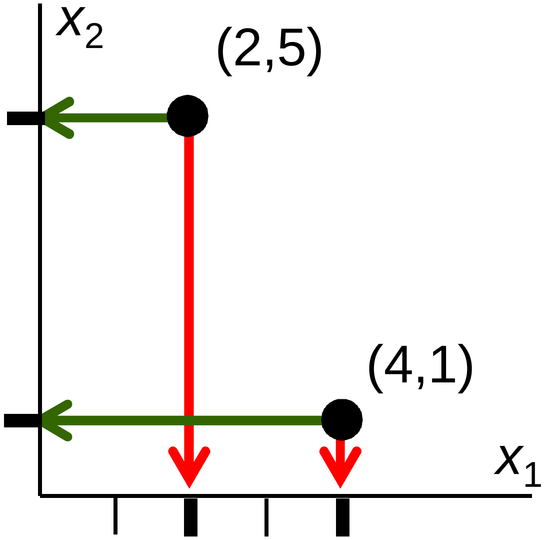
- A constraint set is **bounds consistent** if the **min** and **max** of each variable domain appear in some feasible solution, assuming the other domains are replaced by interval relaxations.

- Example: $2x_1 + x_2 = 9$

  $x_1 \in \{1,2,3,4\}$

  $x_2 \in \{1,5\}$

Projection for **domain** consistency:

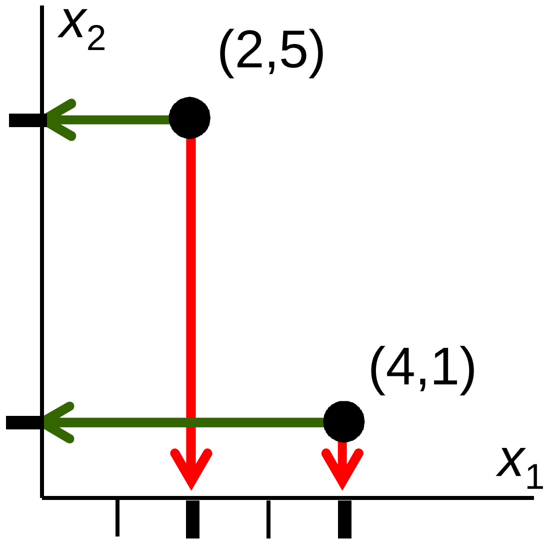


Bounds consistency

- A constraint set is **bounds consistent** if the **min** and **max** of each variable domain appear in some feasible solution, assuming the other domains are replaced by interval relaxations.
- Example: $2x_1 + x_2 = 9$

  $x_1 \in \{\ ,2,\ ,4\}$

  $x_2 \in \{1,5\}$



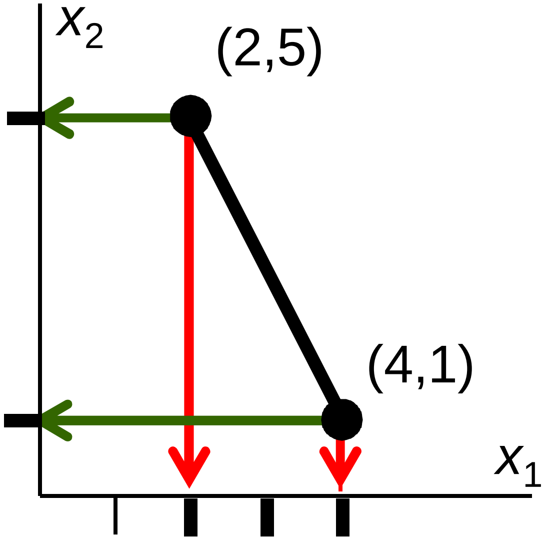
Projection for **domain** consistency:

Filtered domain of x_1 has a “hole.”

Bounds consistency

- A constraint set is **bounds consistent** if the **min** and **max** of each variable domain appear in some feasible solution, assuming the other domains are replaced by interval relaxations.
- Example: $2x_1 + x_2 = 9$
  $x_1 \in \{1, 2, 3, 4\}$
  $x_2 \in \{1, 5\}$

Projection for **bounds** consistency:



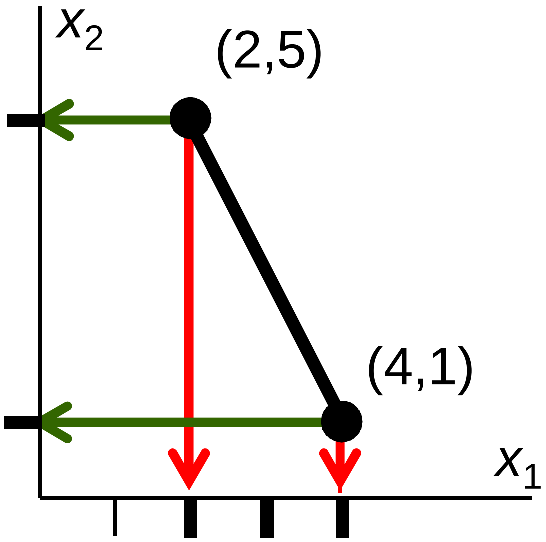
Bounds consistency

- A constraint set is **bounds consistent** if the **min** and **max** of each variable domain appear in some feasible solution, assuming the other domains are replaced by interval relaxations.

- Example: $2x_1 + x_2 = 9$

$$x_1 \in \{\ ,2,3,4\}$$

$$x_2 \in \{1,5\}$$



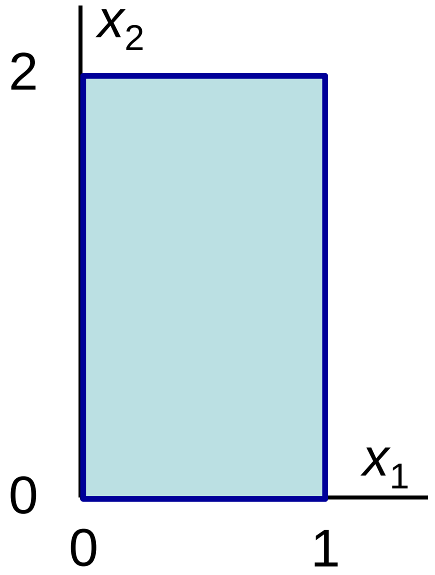
Projection for **bounds** consistency:

Filtered domain for x_1 has no hole.

Bounds propagation

- Bounds obtained by achieving bound consistency can be propagated.
    - This is important in global optimization.
- Example:

$$4x_1x_2 = 1$$
$$2x_1 + x_2 \le 2$$
$$x_1 \in [0,1]$$
$$x_2 \in [0,2]$$



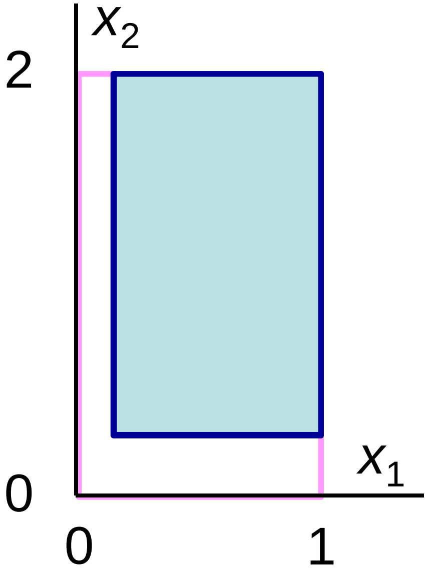
Bounds propagation

- Bounds obtained by achieving bound consistency can be propagated.
    - This is important in global optimization.
- Example:

$$4x_1x_2 = 1$$
$$2x_1 + x_2 \le 2$$
$$x_1 \in [0.125, 1]$$
$$x_2 \in [0.25,\ 2]$$

Filter using constraint 1: $x_1 = \frac{1}{4x_2} \ge \frac{1}{4 \cdot 2} = 0.125$

$$x_2 = \frac{1}{4x_1} \ge \frac{1}{4 \cdot 1} = 0.25$$



Bounds propagation

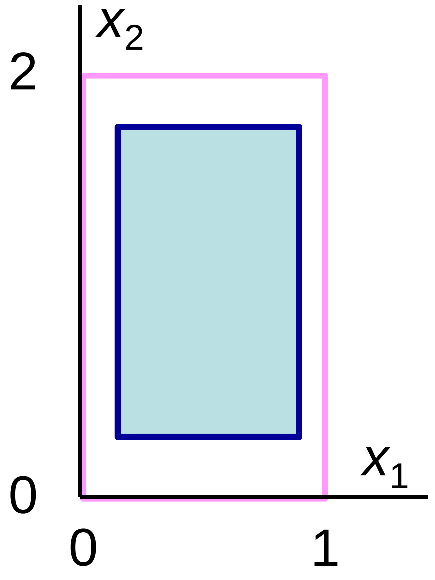
- Bounds obtained by achieving bound consistency can be propagated.
  - This is important in global optimization.
- Example:

$$4x_1x_2 = 1$$
$$2x_1 + x_2 \le 2$$
$$x_1 \in [0.125, 0.875]$$
$$x_2 \in [0.25, 1.75]$$

Propagate to constraint 2::

$$x_1 \le 1 - \frac{x_2}{2} \le \frac{0.25}{2} = 0.875$$

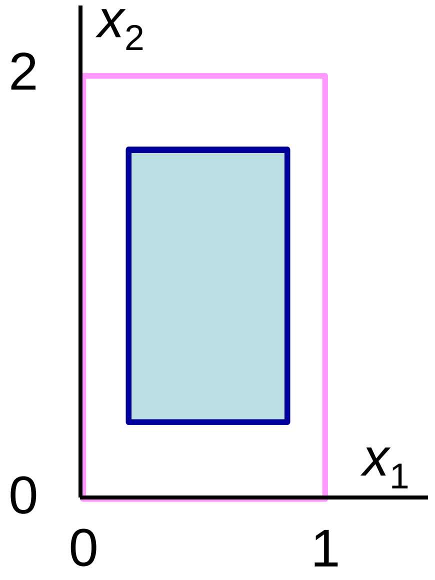
$$x_2 \le 2 - 2x_1 \le 2 - 2 \cdot 0.125 = 1.75$$



Bounds propagation

- Bounds obtained by achieving bound consistency can be propagated.
    - This is important in global optimization.
- Example:

$$
\begin{aligned}
&4x_1x_2 = 1 \\
&2x_1 + x_2 \le 2 \\
&x_1 \in [0.146, 0.854] \\
&x_2 \in [0.293, 1.707]
\end{aligned}
$$

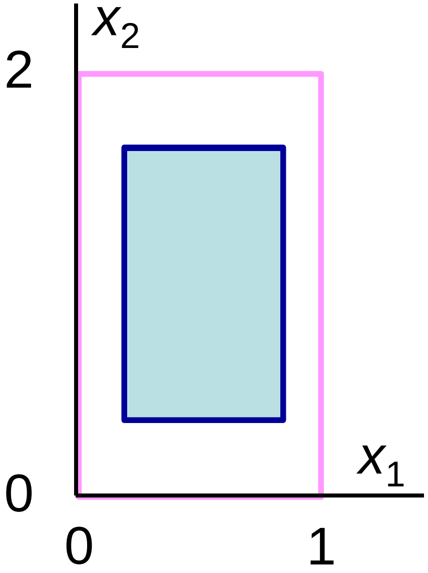


Continuing, bounds asymptotically converge:

Bounds propagation

- Bounds obtained by achieving bound consistency can be propagated.
  - This is important in global optimization.
- Example:

$$4x_1x_2 = 1$$
$$2x_1 + x_2 \le 2$$
$$x_1 \in [0.146, 0.854]$$
$$x_2 \in [0.293, 1.707]$$



Continuing, bounds asymptotically converge:

Solvers truncate the process.

k-consistency

- *k*-consistency is closely related to backtracking.
    - If a feasible problem is strongly k-consistent, and the width of its dependency graph is less than *k* with respect to some ordering of the variables, then forward checking with respect to that order solves the problem without backtracking.

k-consistency

- Definition:
  - A constraint set is *k*-consistent if any assignment to *k* – 1 variables that violates no constraints can be extended to an assignment to *k* variables without violating any constraints.

x_{j_k}

k-consistency

- Definition:
  - A constraint set is *k*-consistent if any assignment to *k* – 1 variables that violates no constraints can be extended to an assignment to *k* variables without violating any constraints.
  - More precisely, given any partial assignment

$$(x_{j_1},\ldots,x_{j_{k-1}}) = (v_1,\ldots v_{k-1})$$

that violates no constraints, and any other variable x_{j_k} there is a value v_k such that

$$(x_{j_1},\ldots,x_{j_{k-1}},x_{j_k}) = (v_1,\ldots v_{k-1},v_k)$$

violates no constraints.

  - A constraint can be violated only if all of its variables are assigned values.

k-consistency

- Example
$$\begin{aligned} x_1 + x_2 \quad\quad\quad + x_4 &\ge 1 \\ x_1 - x_2 + x_3 \quad\quad\quad &\ge 0 \\ x_1 \quad\quad\quad\quad\quad - x_4 &\ge 0 \\ x_j \in \{0,1\} & \end{aligned}$$

  - 1-consistent: trivial

k-consistency

- Example

$$\begin{aligned} x_1 + x_2 \phantom{{}+x_3} + x_4 &\ge 1 \\ x_1 - x_2 + x_3 \phantom{{}+x_4} &\ge 0 \\ x_1 \phantom{{}+x_2+x_3} - x_4 &\ge 0 \\ x_j \in \{0,1\} & \end{aligned}$$

  - 1-consistent: trivial
  - 2-consistent: need only check x_1

k-consistency

- Example

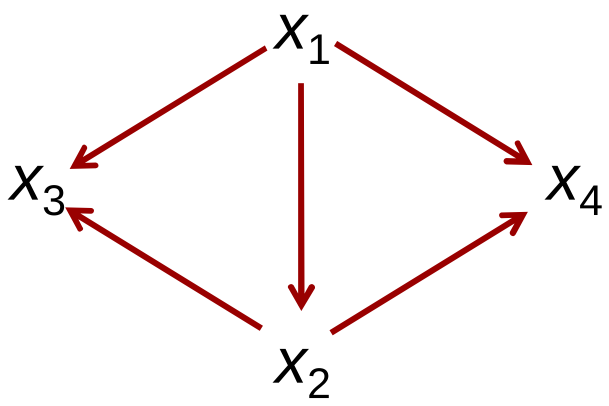
$$\begin{aligned} x_1 + x_2 \qquad\qquad + x_4 &\geq 1 \\ x_1 - x_2 + x_3 \qquad\qquad &\geq 0 \\ x_1 \qquad\qquad\qquad - x_4 &\geq 0 \\ x_j \in \{0,1\} \end{aligned}$$

  - 1-consistent: trivial
  - 2-consistent: need only check x_1
  - not 3-consistent:
    $(x_1,x_2) = (0,0)$ cannot be extended to $(x_1,x_2,x_4) = (0,0,?)$.
    $(x_1,x_3) = (0,0)$ cannot be extended to $(x_1,x_3,x_4) = (0,0,?)$.
    - There are the only pairs that can’t be extended.

Dependency graph

- **Dependency graph**: variables are connected by edges when they occur in a common constraint.
  - Also called **primal graph**.

$$
\begin{aligned}
x_1 + x_2 \qquad\qquad + x_4 &\geq 1 \\
x_1 - x_2 + x_3 \qquad\qquad &\geq 0 \\
x_1 \qquad\qquad\qquad - x_4 &\geq 0 \\
x_j \in \{0,1\} &
\end{aligned}
$$

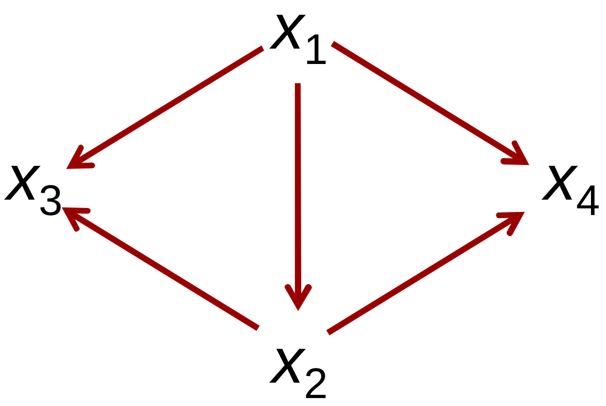


Dependency graph for ordering 1,2,3,4

Dependency graph

- **Dependency graph**: variables are connected by edges when they occur in a common constraint.
  - Also called **primal graph**.

$$
\begin{aligned}
x_1 + x_2 \quad\quad + x_4 &\geq 1 \\
x_1 - x_2 + x_3 \quad\quad &\geq 0 \\
x_1 \quad\quad\quad\quad - x_4 &\geq 0 \\
x_j \in \{0,1\}
\end{aligned}
$$



Dependency graph for ordering 1,2,3,4

Width of the graph is the maximum in-degree (here, 2).

Backtracking

- A constraint set is strongly *k*-consistent if it is *i*-consistent for *i* = 1,…,*k*.

Theorem (Freuder). If a feasible problem is strongly *k*-consistent, and the width of its dependency graph is less than *k* with respect to some ordering of the variables, then forward checking with respect to that order solves the problem without backtracking.

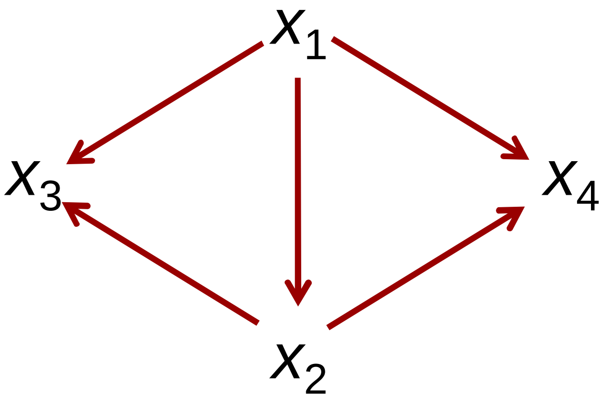
Backtracking

- The example doesn't satisfy the conditions of the theorem.
  - Width = 2, not strongly 3-consistent.
  - Backtracking is possible, and it occurs when we set

    $(x_1,x_2,x_3,x_4) = (0,0,0,?)$

$$\begin{aligned} x_1 + x_2 \quad\quad + x_4 &\ge 1 \\ x_1 - x_2 + x_3 \quad\quad &\ge 0 \\ x_1 \quad\quad\quad\quad - x_4 &\ge 0 \\ x_j \in \{0,1\} & \end{aligned}$$

  - A feasible solution is $(x_1,x_2,x_3,x_4) = (1,0,0,0)$.



Width = 2

Backtracking

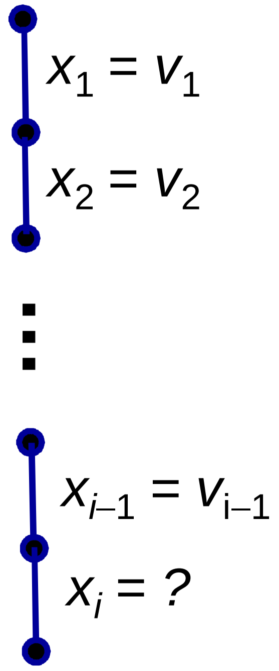
- Suppose we add two constraints:.

$$\begin{array}{llllr}
x_1 + x_2 & & + x_4 & \geq 1 \\
x_1 - x_2 & + x_3 & & \geq 0 \\
x_1 & & - x_4 & \geq 0 \\
x_1 + x_2 & & & \geq 1 \\
x_1 & + x_3 & & \geq 1 \\
x_j \in \{0,1\} & & & &
\end{array}$$

  - This is strongly 3-consistent.
    - Extra constraints rule out the only partial solutions that couldn't be extended: $(x_1,x_2) = (0,0)$, $(x_1,x_3) = (0,0)$
  - Now it satisfies conditions of the theorem.
    - Backtracking does not occur.
    - For example, $(x_1,x_2,x_3,x_4) = (0,1,1,0)$.

Backtracking

- Proof of theorem, by induction on k.
  - x_1 can be assigned a value without violating a constraint, because problem is feasible.
  - Suppose $x_1, \ldots, x_{i-1}$ have been assigned values without violating a constraint. Show x_i can be assigned a value.
    - x_i occurs in the same constraint as at most $k - 1$ earlier variables.
    - So these variable assignments can be extended to x_i.
    - Thus assignments to $x_1, \ldots, x_{i-1}$ can be extended to x_i.

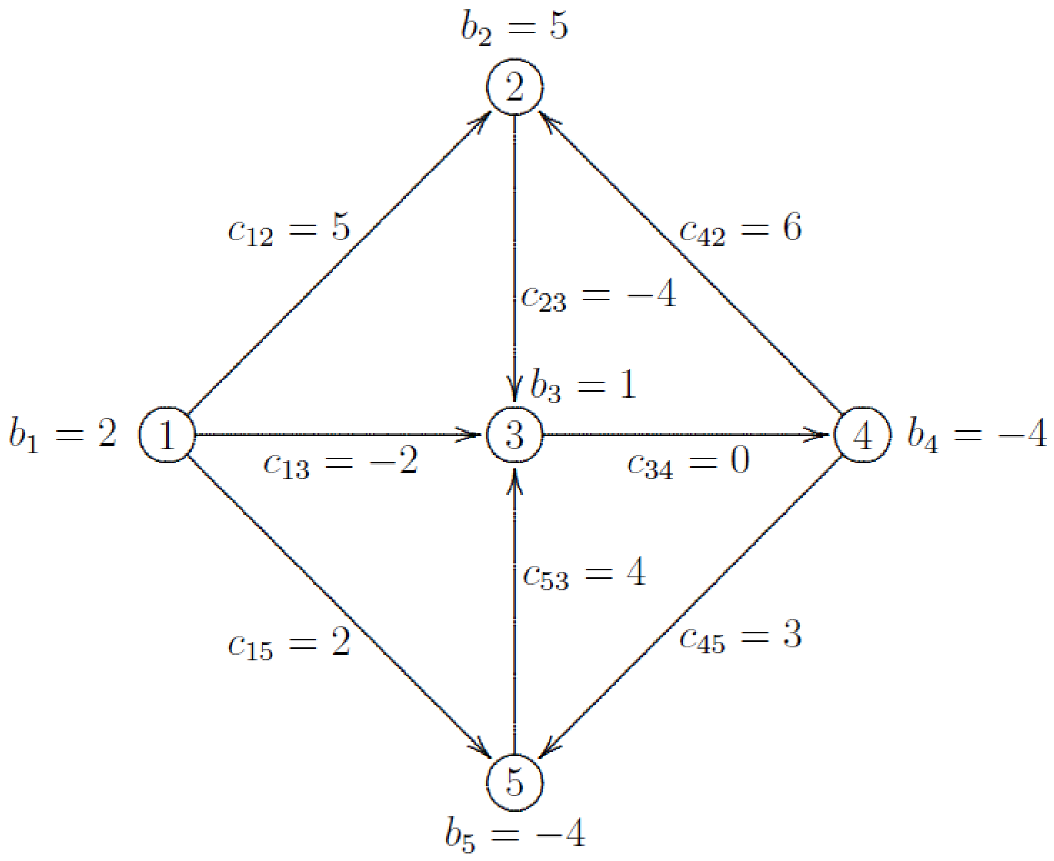


Review of Network Flow Theory

Min cost network flow
Basis tree theorem
Max flow
Bipartite matching

Min cost network flow problem

- Example of a min cost network flow problem:



It is a linear programming problem:

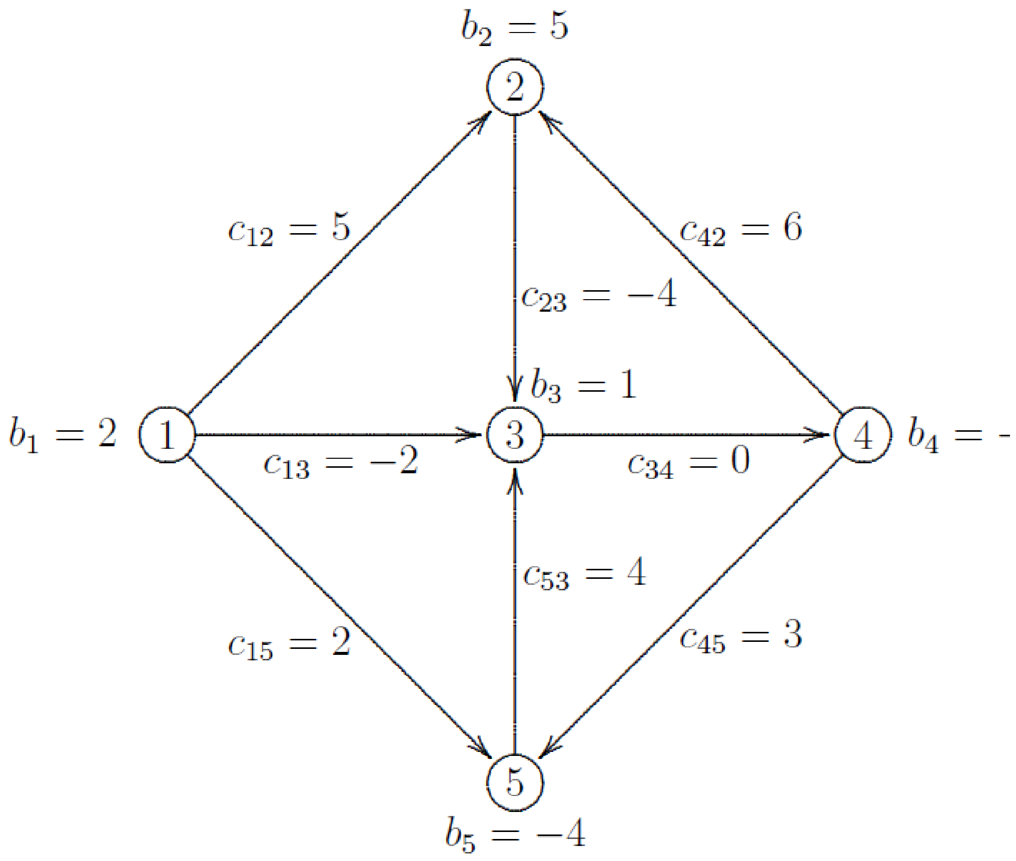
$$\min \sum_{ij} c_{ij} x_{ij}$$

$$\sum_j x_{ij} - \sum_j x_{ji} = b_i, \text{ all } i$$

$$x_{ij} \geq 0, \text{ all } i, j$$

Min cost network flow problem

- Example of a min cost network flow problem:



In matrix form:

$$\min \sum_{ij} c_{ij} x_{ij}$$

$$\begin{bmatrix} 1 & 1 & 1 & & & & & \\ -1 & & & 1 & & -1 & & \\ & -1 & & -1 & 1 & & & -1 \\ & & & & -1 & 1 & 1 & \\ & & -1 & & & & -1 & 1 \end{bmatrix} \begin{bmatrix} x_{12} \\ x_{13} \\ x_{15} \\ x_{23} \\ x_{34} \\ x_{42} \\ x_{45} \\ x_{53} \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \\ 1 \\ -4 \\ -4 \end{bmatrix}$$

$$x_{ij} \geq 0, \text{ all } i, j$$

Min cost network flow problem

- If the matrix is *m* x *n*, it has rank *m* – 1.
  - So a basic solution of the LP has *m* – 1 basic variables.
- **Basis tree theorem:** Every basis corresponds to a spanning tree.

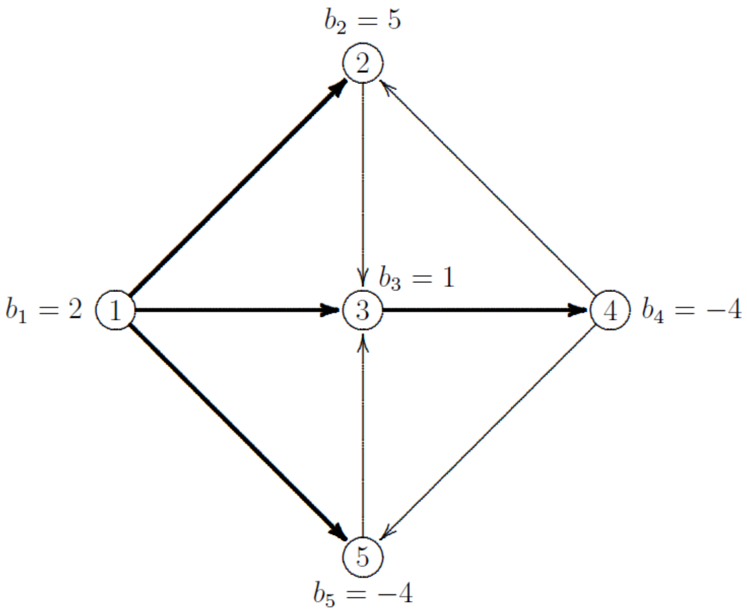
$$\min \sum_{ij} c_{ij} x_{ij}$$

$$\begin{bmatrix} 1 & 1 & 1 & & & & & \\ -1 & & & 1 & & -1 & & \\ & -1 & & -1 & 1 & & & -1 \\ & & & & -1 & 1 & 1 & \\ & & -1 & & & & -1 & 1 \end{bmatrix} \begin{bmatrix} x_{12} \\ x_{13} \\ x_{15} \\ x_{23} \\ x_{34} \\ x_{42} \\ x_{45} \\ x_{53} \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \\ 1 \\ -4 \\ -4 \end{bmatrix}$$

$$x_{ij} \geq 0, \ \text{all } i, j$$

Min cost network flow problem

- If the matrix is *m* x *n*, it has rank *m* – 1.
  - So a basic solution of the LP has *m* – 1 basic variables.
- **Basis tree theorem:** Every basis corresponds to a spanning tree.



$$\min \sum_{ij} c_{ij} x_{ij}$$

$$\begin{bmatrix} 1 & 1 & 1 & & & & & \\ -1 & & & 1 & & -1 & & \\ & -1 & & -1 & 1 & & & -1 \\ & & & & -1 & 1 & 1 & \\ & & -1 & & & & -1 & 1 \end{bmatrix} \begin{bmatrix} x_{12} \\ x_{13} \\ x_{15} \\ x_{23} \\ x_{34} \\ x_{42} \\ x_{45} \\ x_{53} \end{bmatrix} = \begin{bmatrix} 2 \\ 5 \\ 1 \\ -4 \\ -4 \end{bmatrix}$$

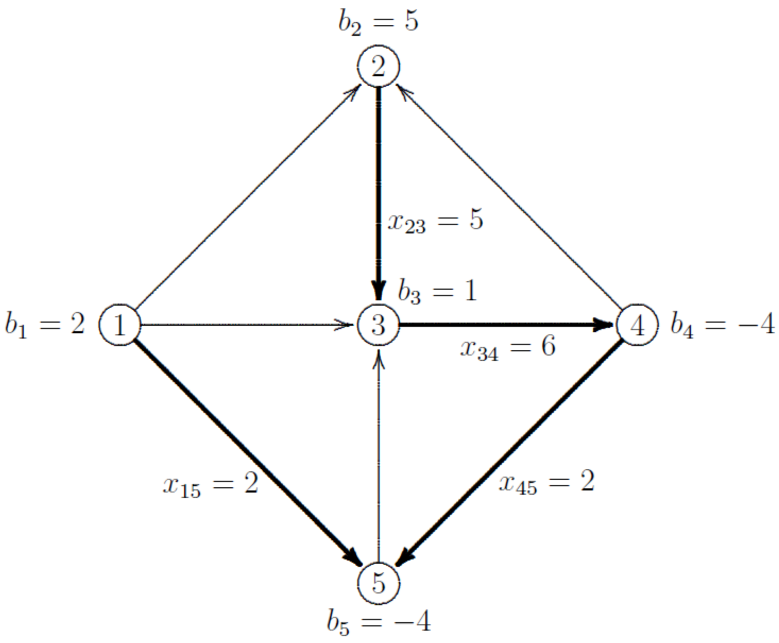
$$x_{ij} \geq 0, \text{ all } i, j$$

Min cost network flow problem

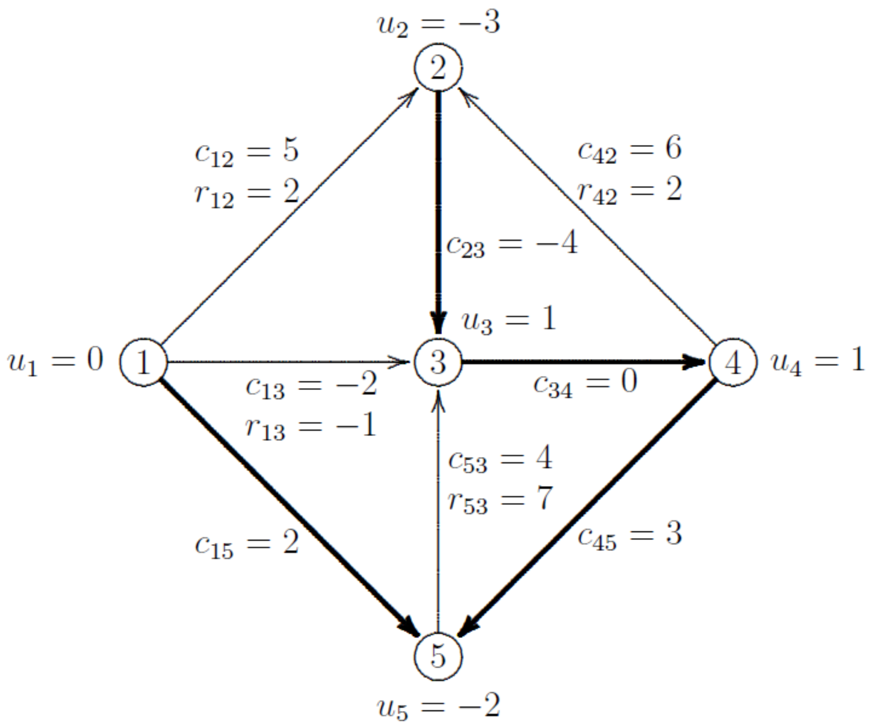
- Optimality test.
  - A basic solution (flow) is optimal if all reduced costs are nonnegative.
  - The reduced cost of a nonbasic flow x_{ij} is $c_{ij} – u_i – u_j$, where u_i is the dual multiplier (potential) for the flow balance constraint at node *i*.
  - Due to complementary slackness, we can find the potentials u_i by solving the equations $u_i – u_j = c_{ij}$ for all basic arcs (*i*,*j*).

Min cost network flow problem

- Finding potentials and reduced costs.
  - We find the potentials u_i by solving the equations $u_i - u_j = c_{ij}$ for all basic arcs (i,j). Then the reduced cost of nonbasic x_{ij} is $r_{ij} = c_{ij} - u_i + u_j$



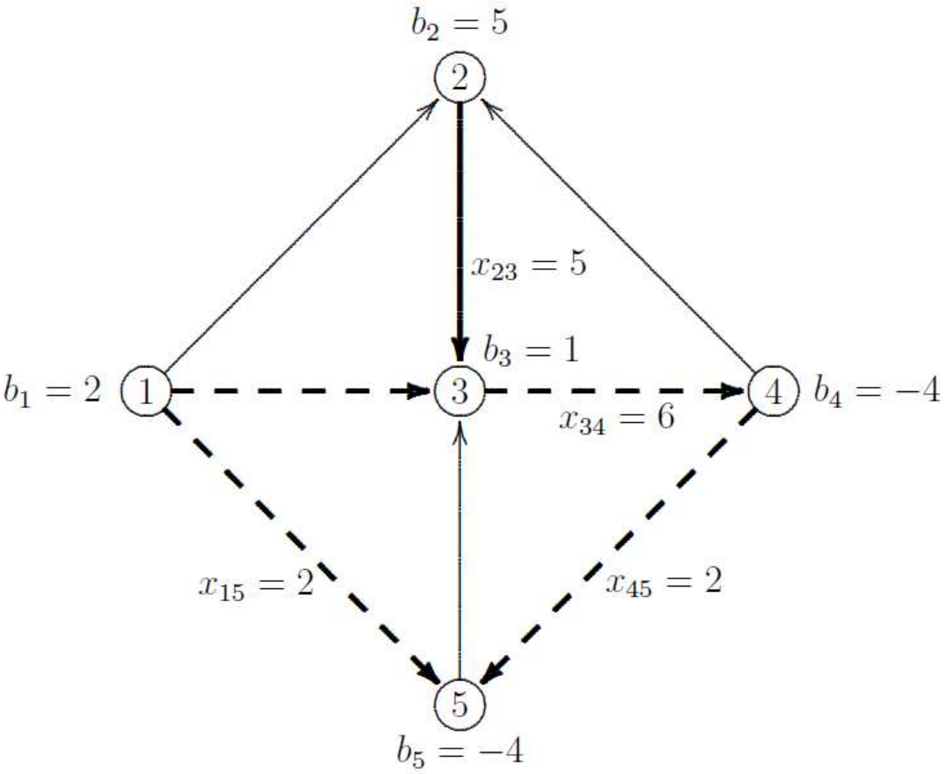
A basic solution

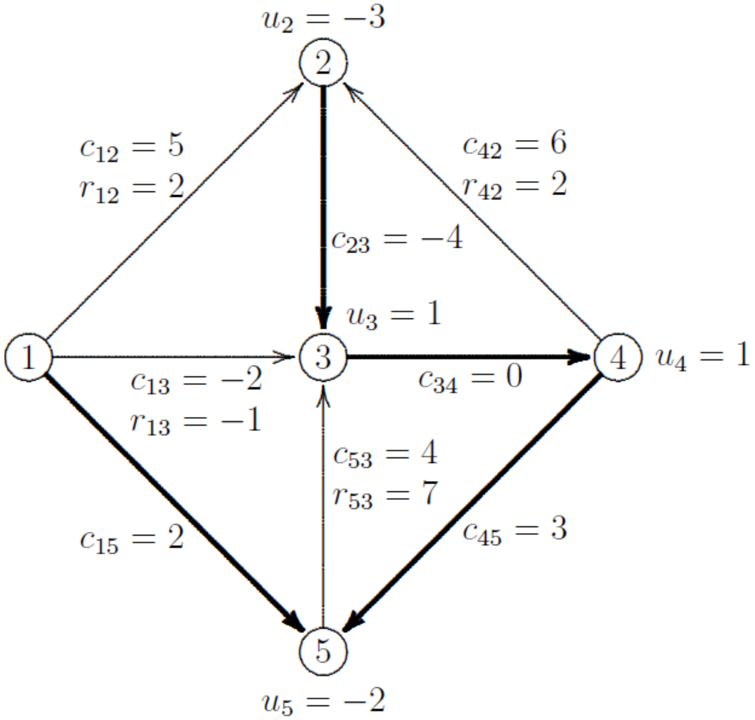


Potentials and reduced costs

Min cost network flow problem

- Improving the solution.
  - Since x_{13} has reduced cost $r_{13} < 0$, we increase flow on (1,3).
    - Adding (1,3) to basis tree creates a cycle.

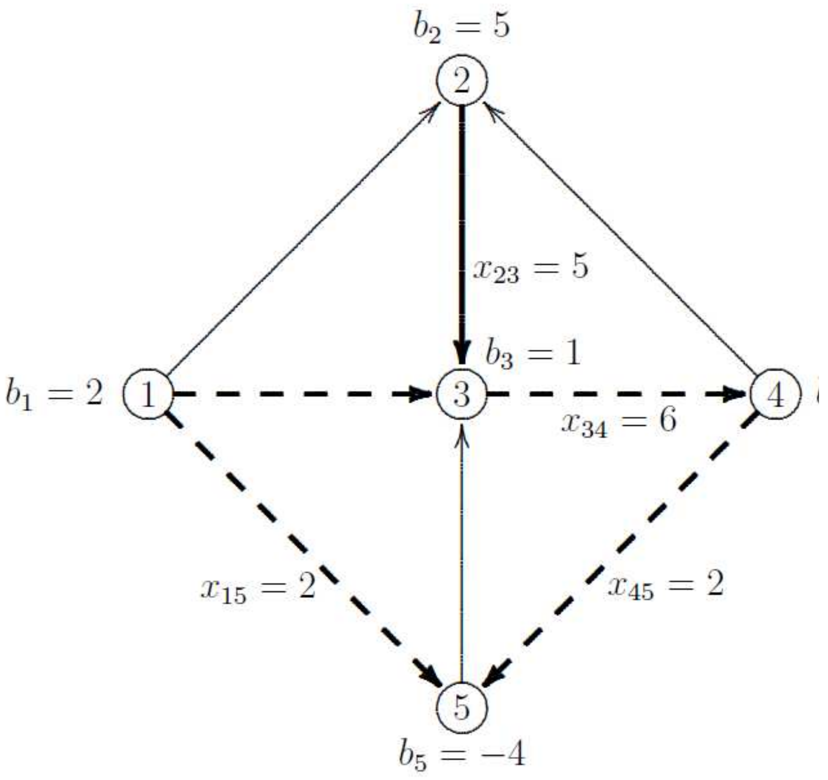


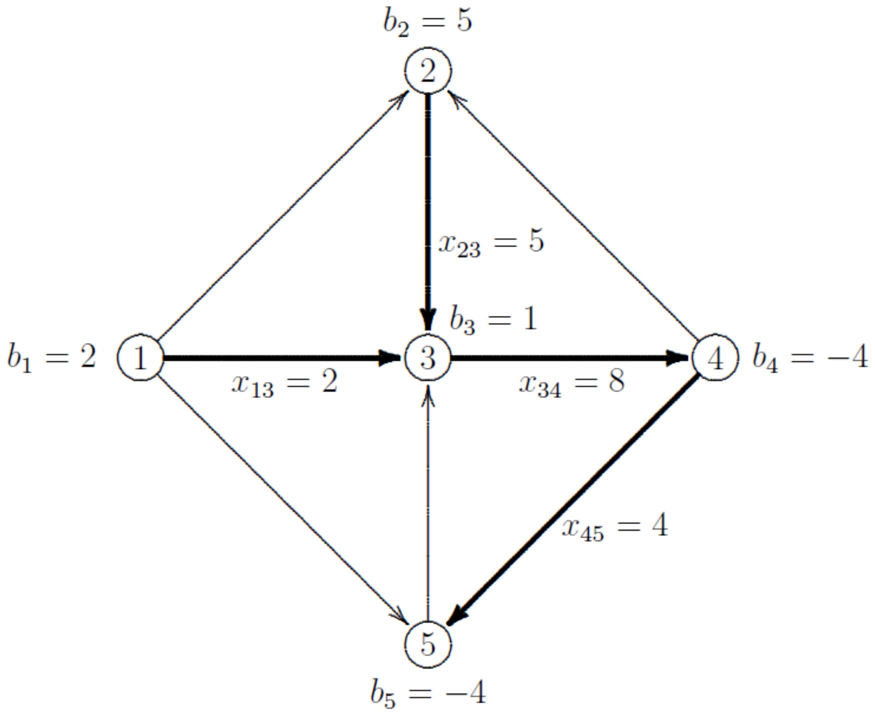


Potentials and reduced costs

Min cost network flow problem

- Improving the solution.
  - Remove from cycle the arc on which flow first hits zero.

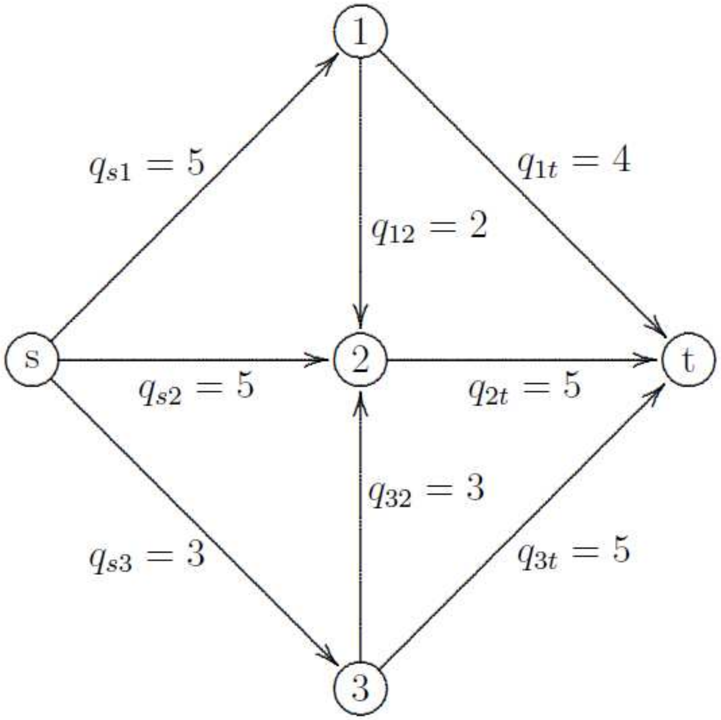




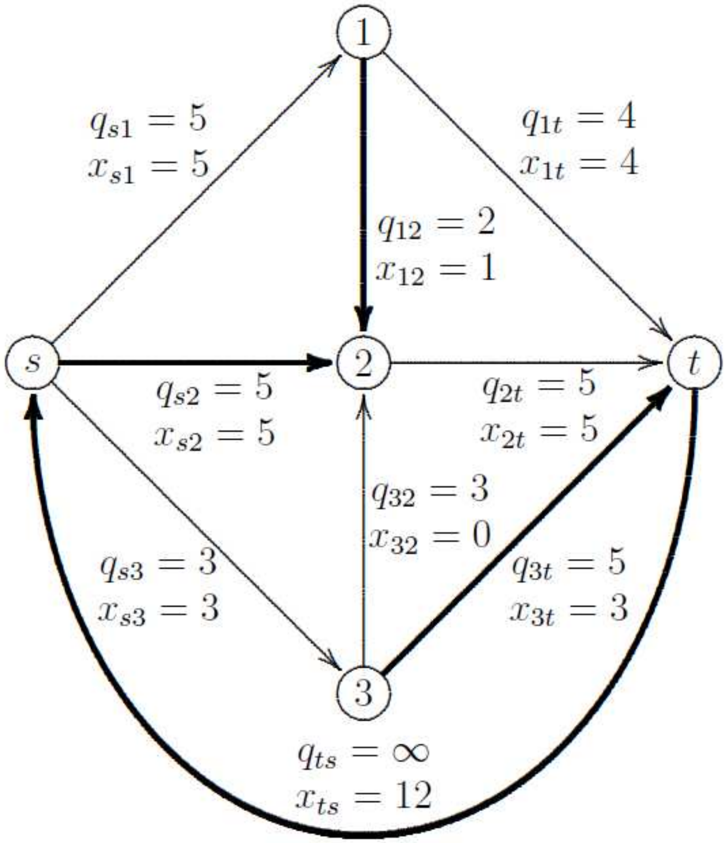
Optimal solution

Maximum flow problem

- The max flow problem is a special case of the min (max) cost network flow problem. Cost on return arc is +1.



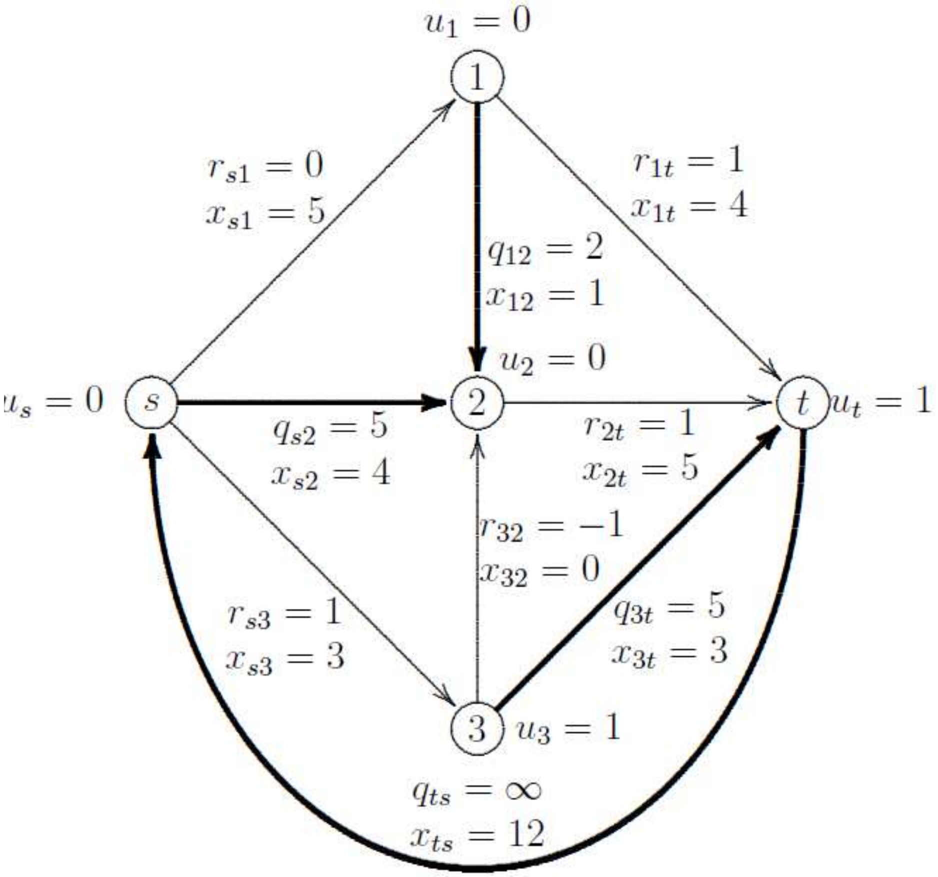
A max flow problem



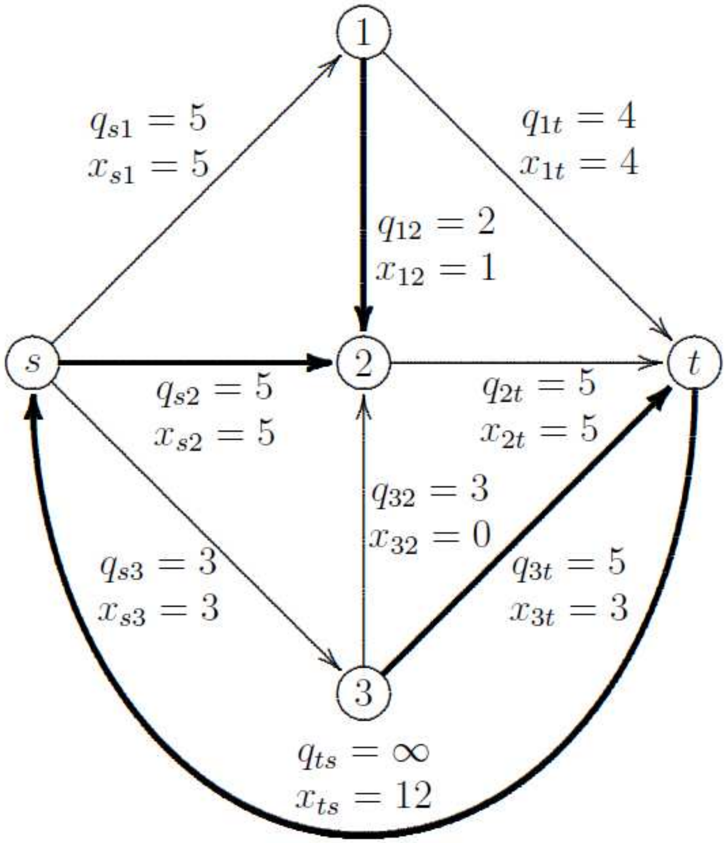
Max cost network flow formulation

Maximum flow problem

- The max flow problem is a special case of the min (max) cost network flow problem.



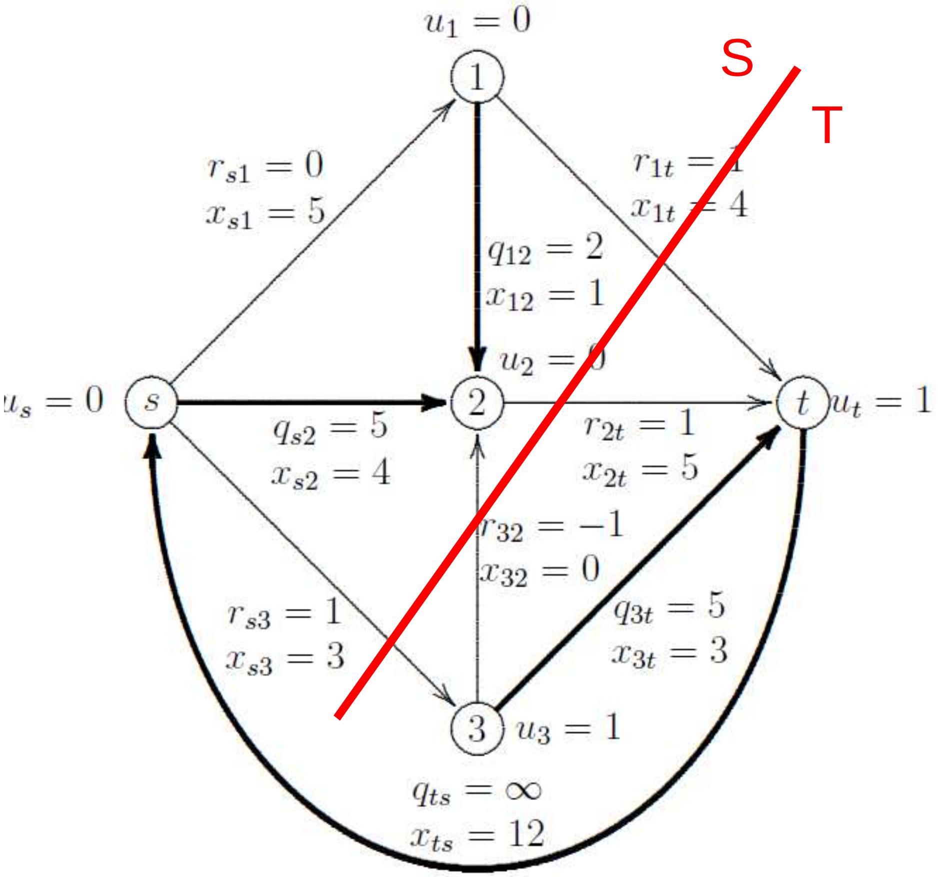
Potentials and reduced costs



Max cost network flow formulation

Maximum flow problem

- The max flow problem is a special case of the min (max) cost network flow problem.



(S,T) cut.

Potentials in S are 0.
Potentials in T are 1.

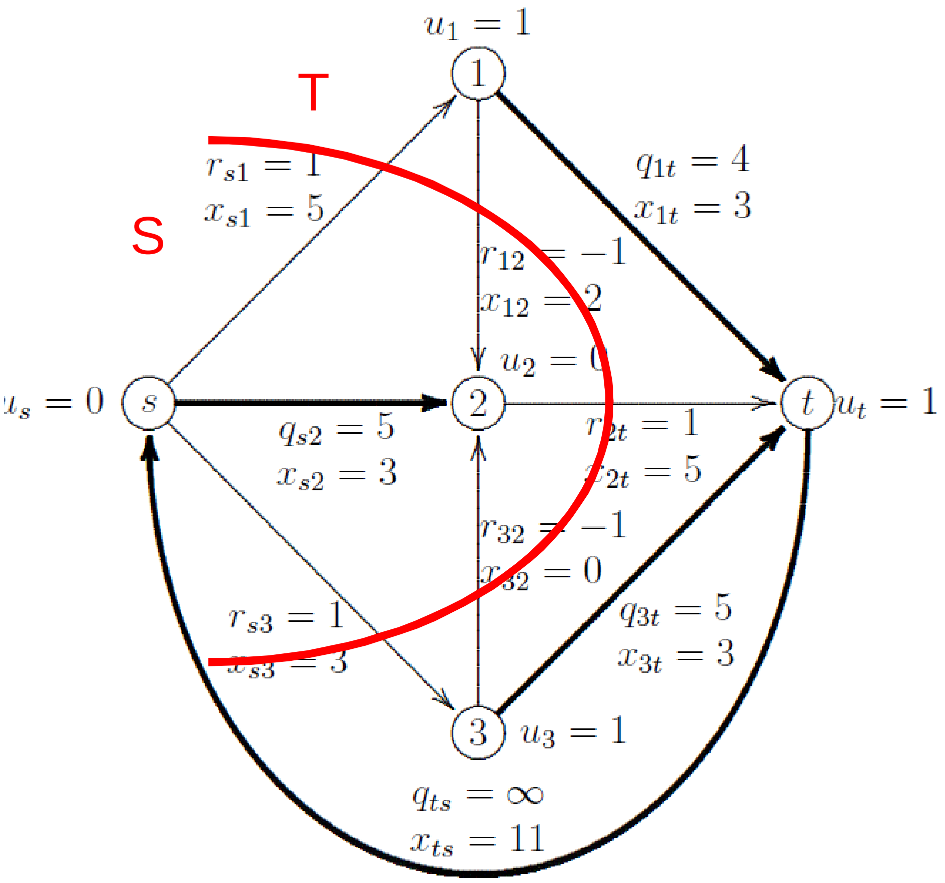
So reduced costs S→ T are 1.
Redued costs T→S are –1.

Flow is max if S→ T arcs are saturated and costs T→S arcs are empty.

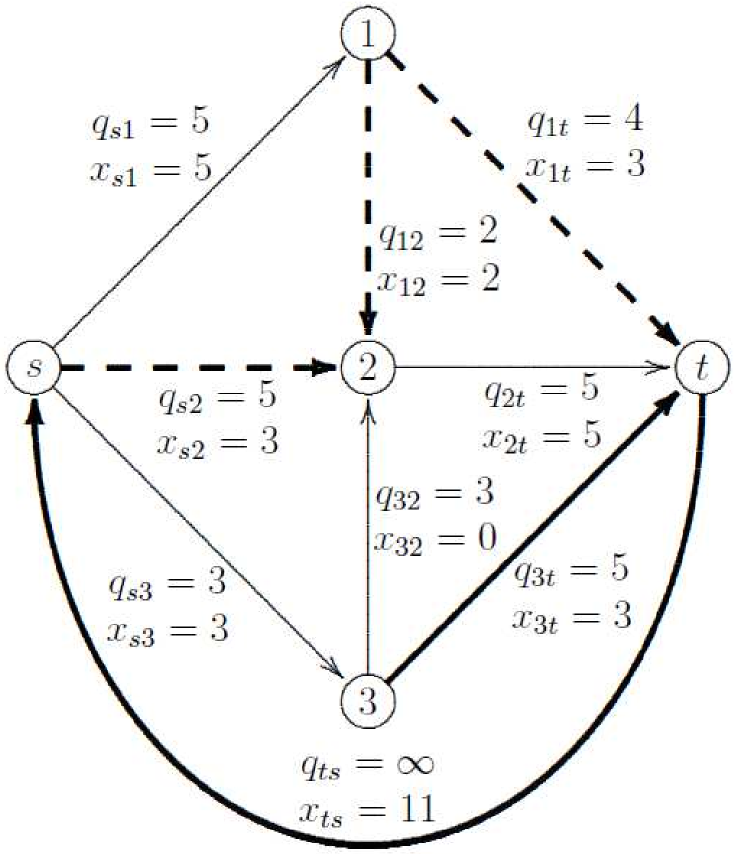
Potentials and reduced costs

Maximum flow problem

- If solution is suboptimal, adding arc to the basis creates a cycle.



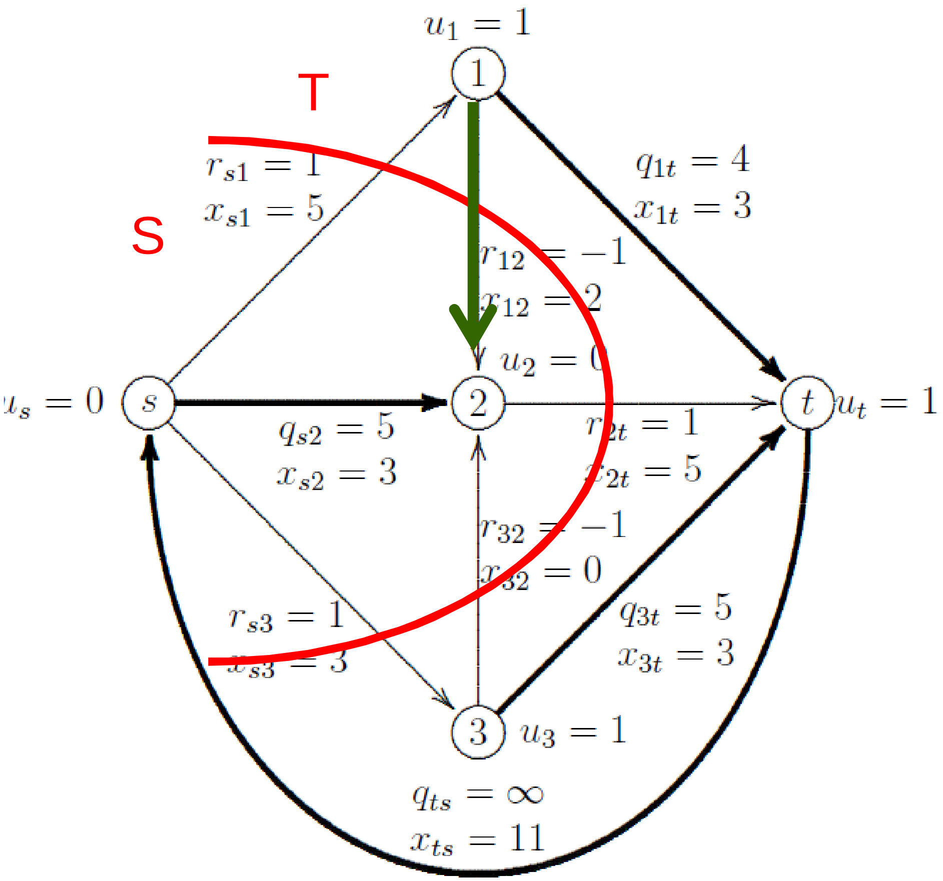
Suboptimal flow



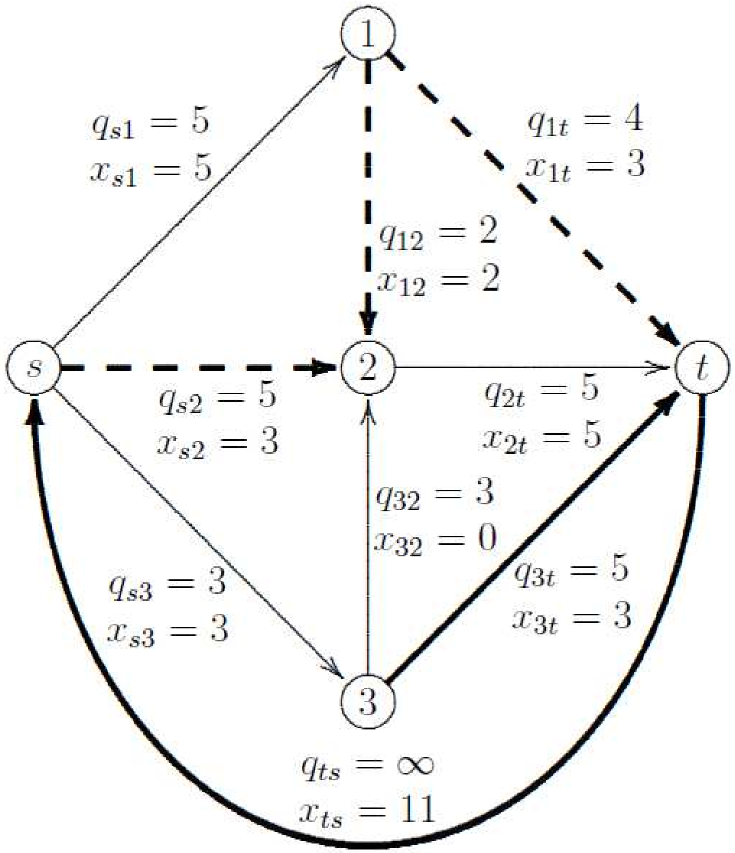
Cycle created by arc (1,2)

Maximum flow problem

- If solution is suboptimal, adding arc to the basis creates a cycle.



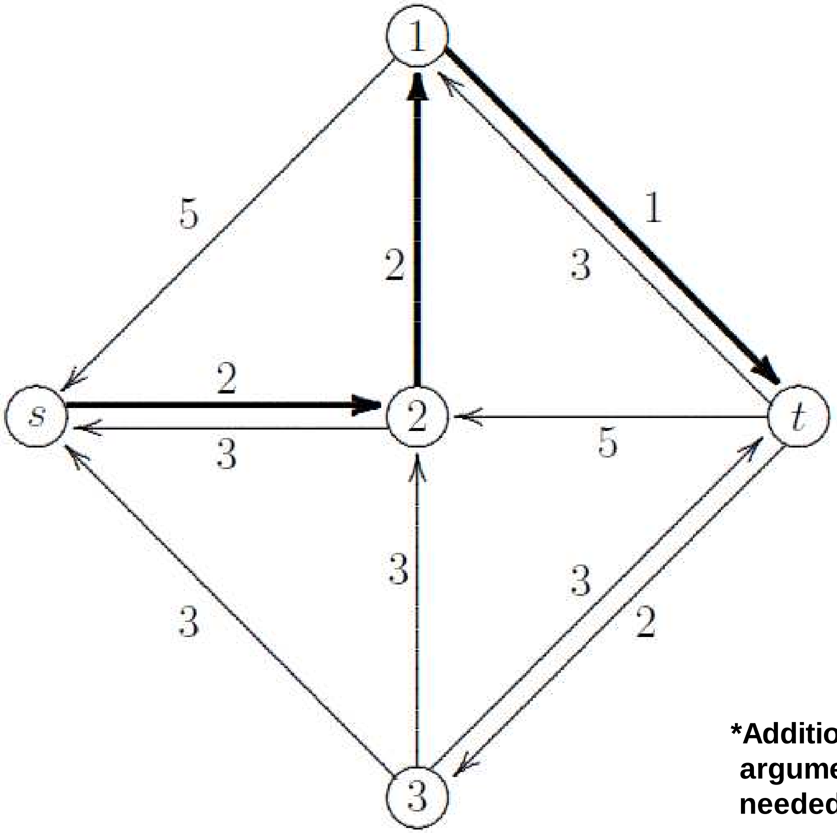
Suboptimal flow



Cycle created by arc (1,2)

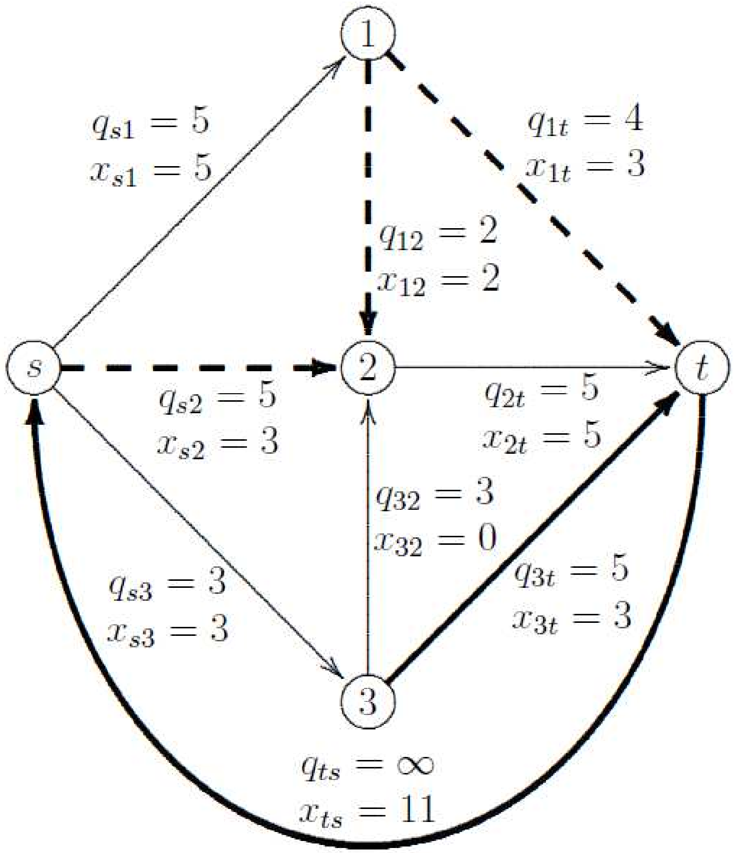
Maximum flow problem

- Cycle defines an **augmenting path** in **residual graph**.
  - So if solution is suboptimal, there is an augmenting path.*



Residual graph

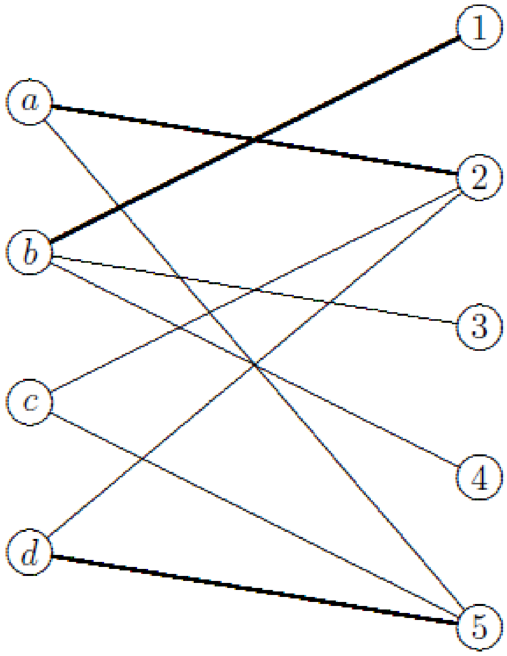
*Additional argument needed in case of degeneracy.



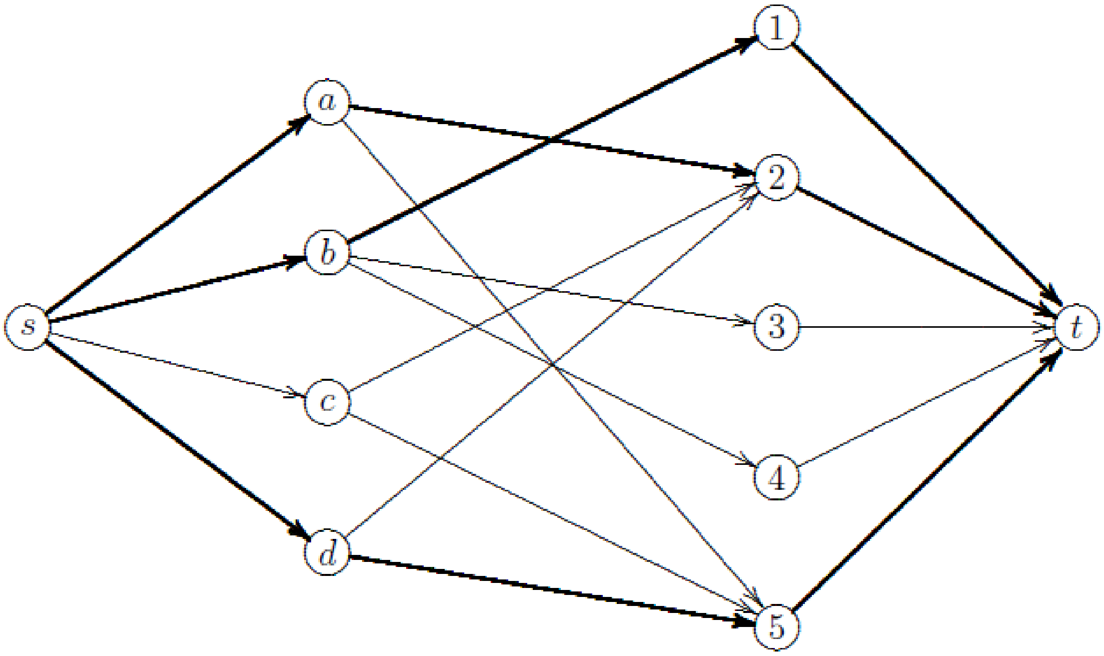
Cycle created by arc (1,2)

Bipartite matching

- Max cardinality bipartite matching can be formulated as max flow.



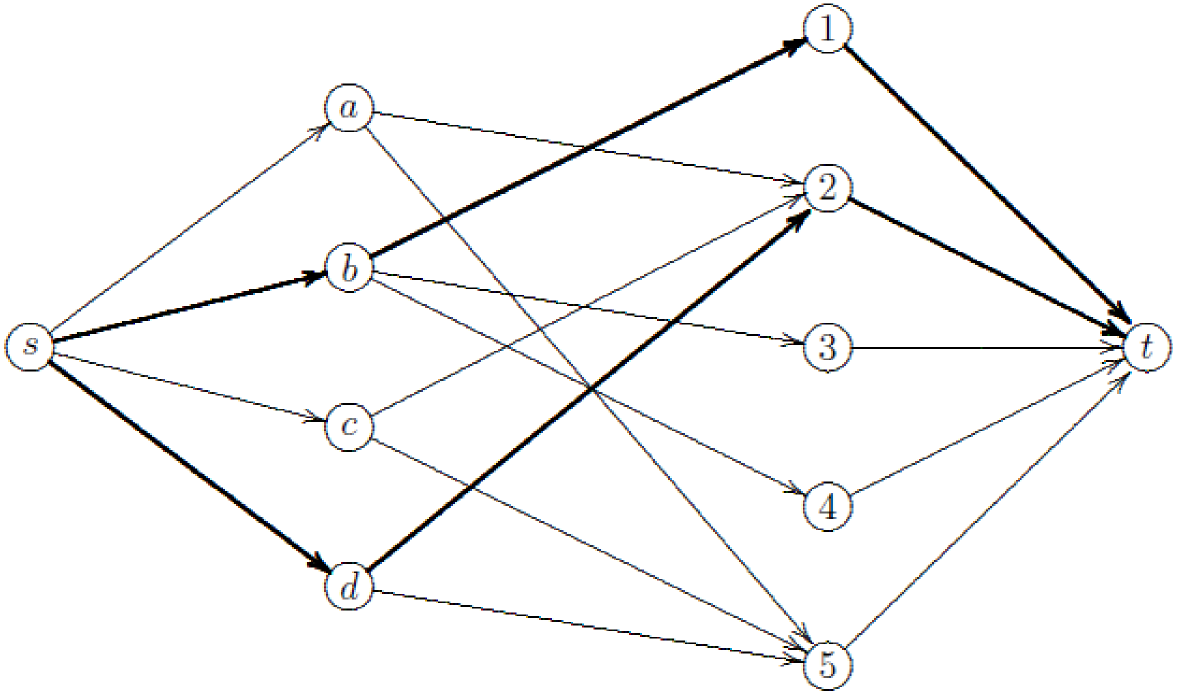
A max cardinality matching



Max flow problem

Bipartite matching

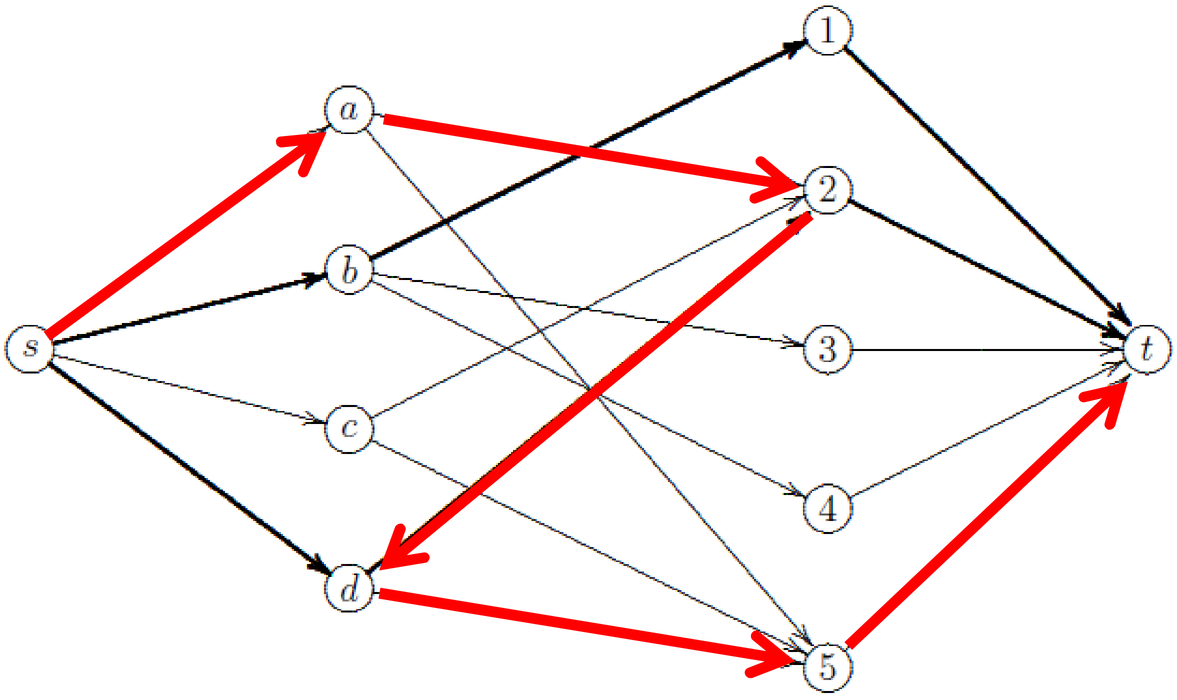
- Augmenting paths in max flow correspond to **alternating paths**.



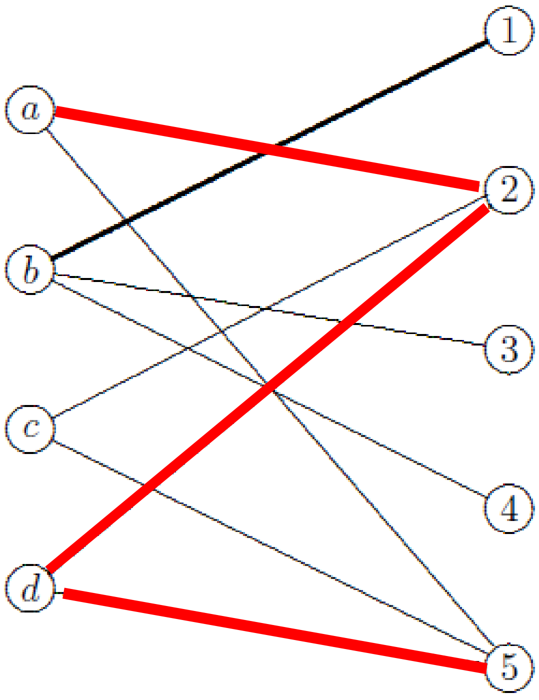
A suboptimal flow

Bipartite matching

- Augmenting paths in max flow correspond to **alternating paths**.



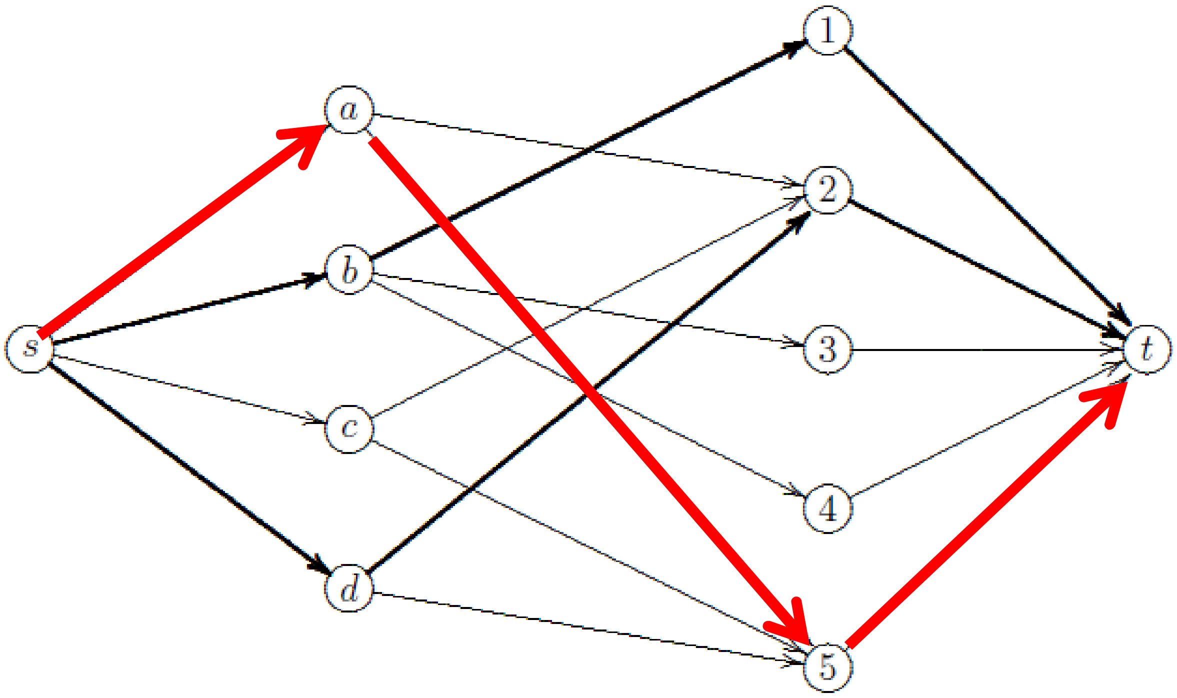
Augmenting path



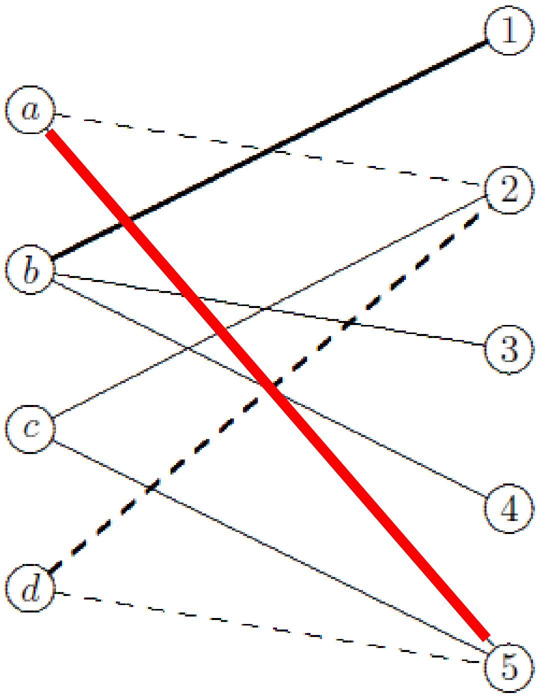
Alternating path

Bipartite matching

- Augmenting paths in max flow correspond to **alternating paths**.



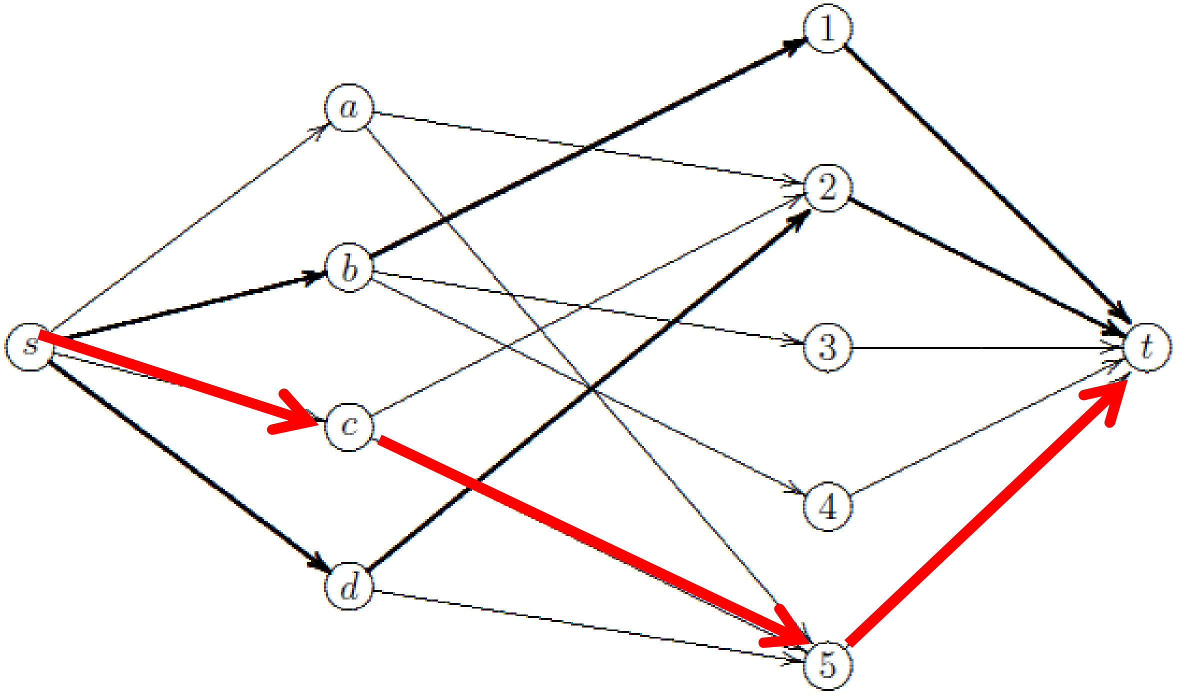
Augmenting path



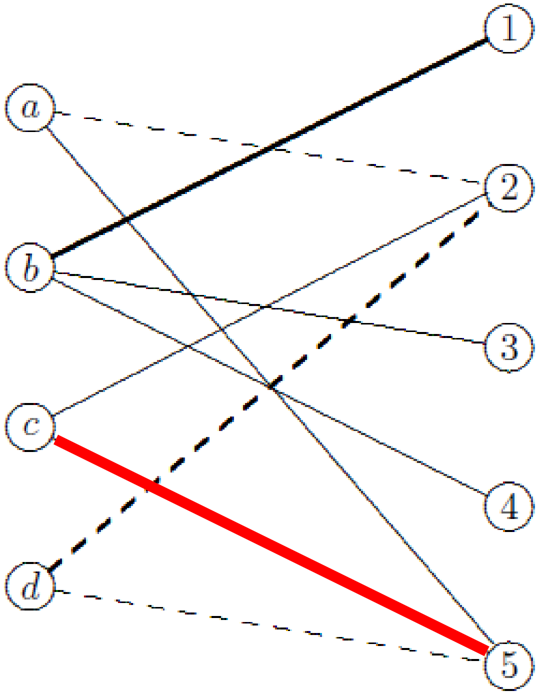
Alternating path

Bipartite matching

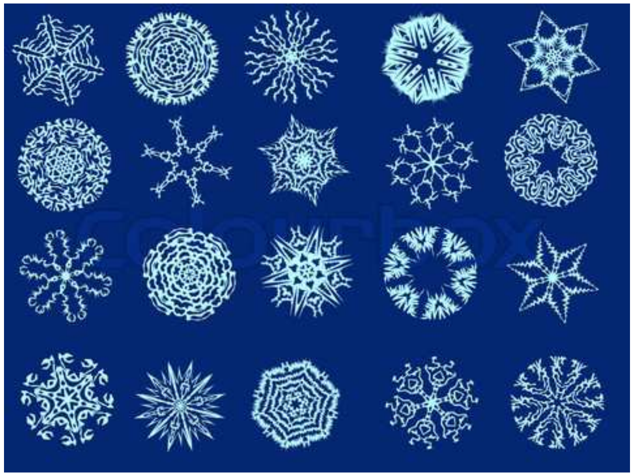
- Augmenting paths in max flow correspond to **alternating paths**.



Augmenting path



Alternating path



All-different Constraint

Matching Model
Domain Consistency
Bounds Consistency

All-different constraint

- The alldiff constraint requires x_1, …, x_n to take pairwise distinct values.

$$\text{alldiff}\left(x_1,\ldots,x_n\right)$$

Matching model

- Alldiff has a solution if and only if there is a perfect matching.

$$\text{alldiff}\left(x_1, x_2, x_3, x_4, x_5\right)$$

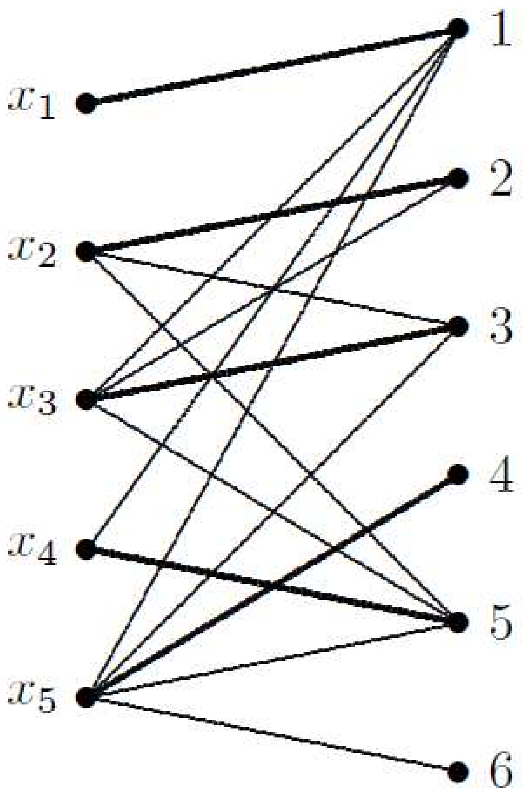
$$x_1 \in \{1\}$$

$$x_2 \in \{2,3,5\}$$

$$x_1 \in \{1,2,3,5\}$$

$$x_1 \in \{1,5\}$$

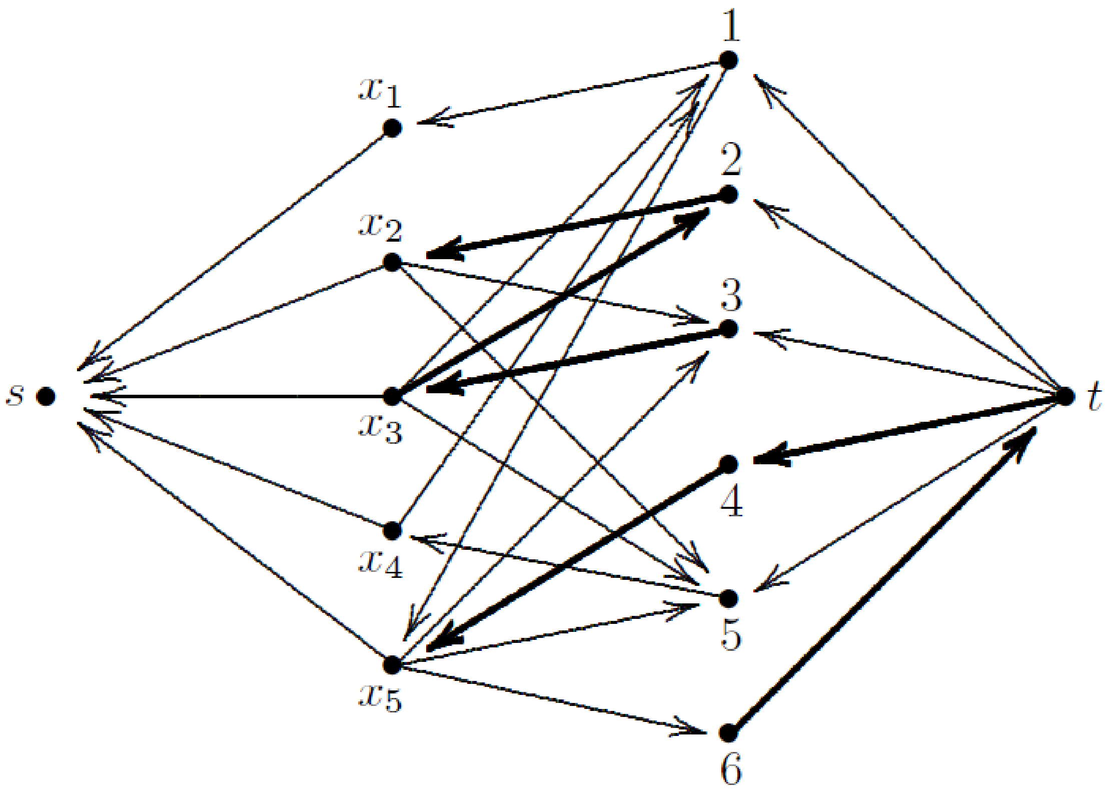
$$x_1 \in \{1,3,4,5,6\}$$



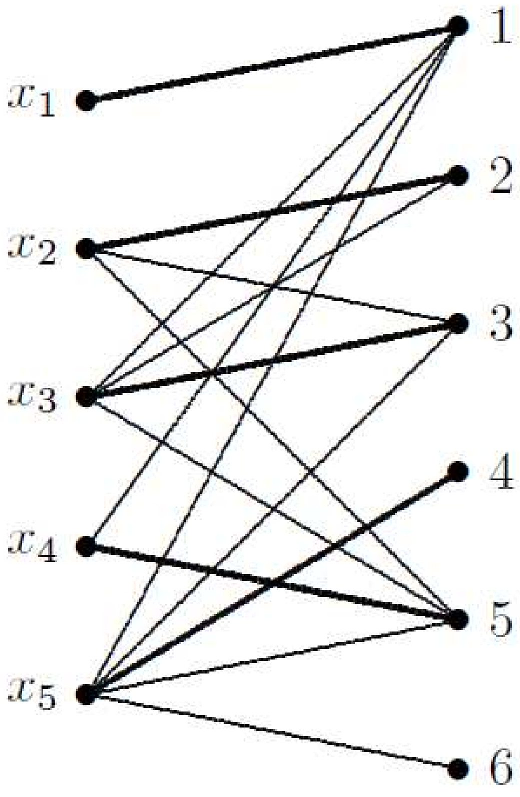
- Solution shown:
  $(x_1,x_2,x_3,x_4,x_5) = (1,2,3,5,4)$

Max flow model

- Alldiff has a solution if and only if max flow = 5.
    - All arcs have capacity 1, except return arc with capacity 5.

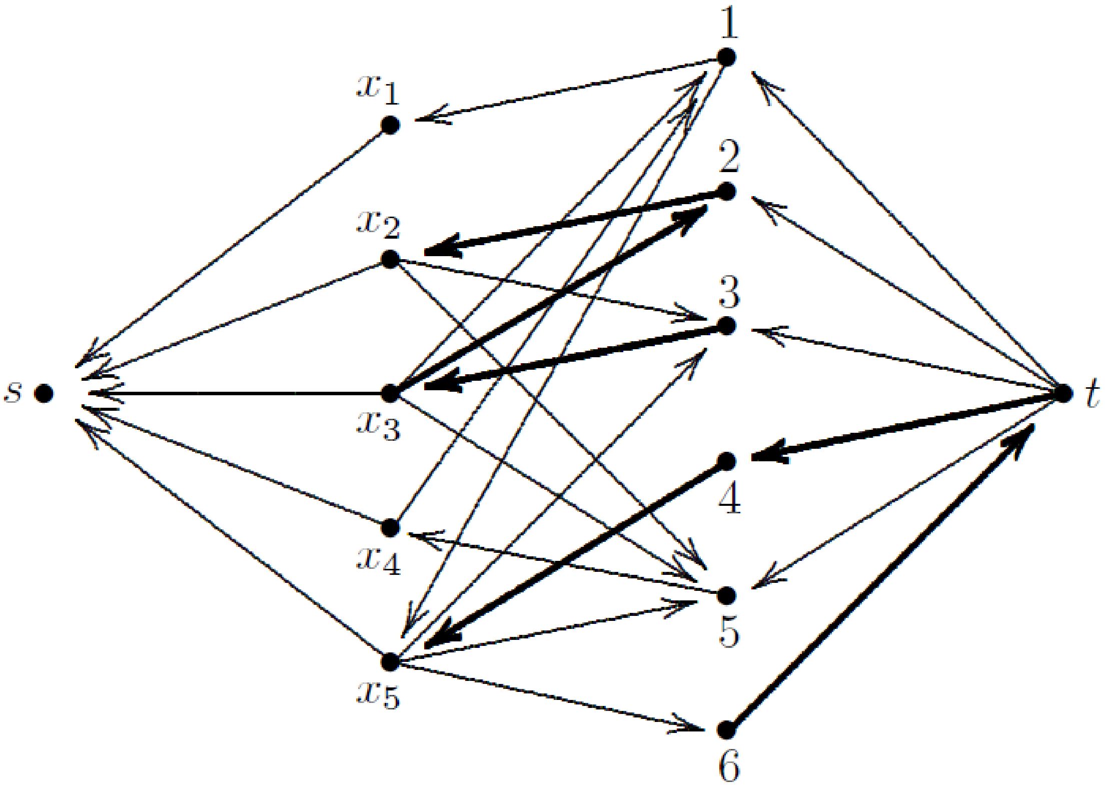


Residual graph for max flow

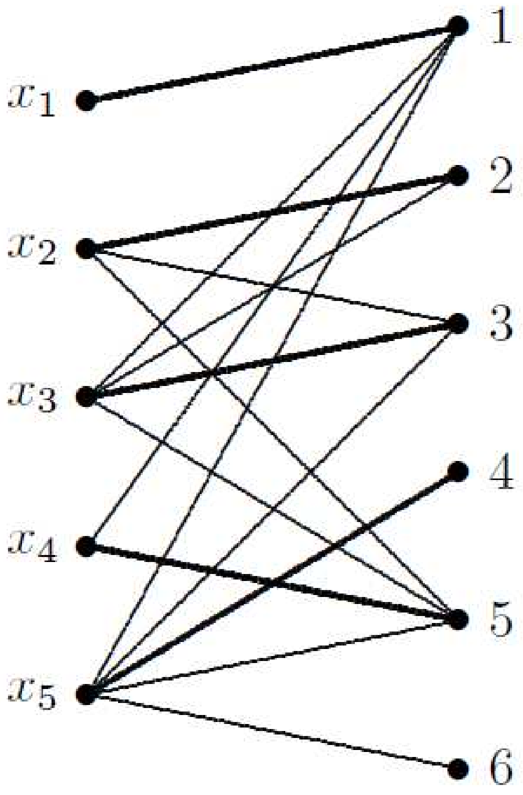


Domain filtering

- To filter domains, fix flow on return arc to 5.
  - Can 3 be removed from domain of x_2? Solve max flow problem from 3 to x_2, treating (x_2,3) as return arc.

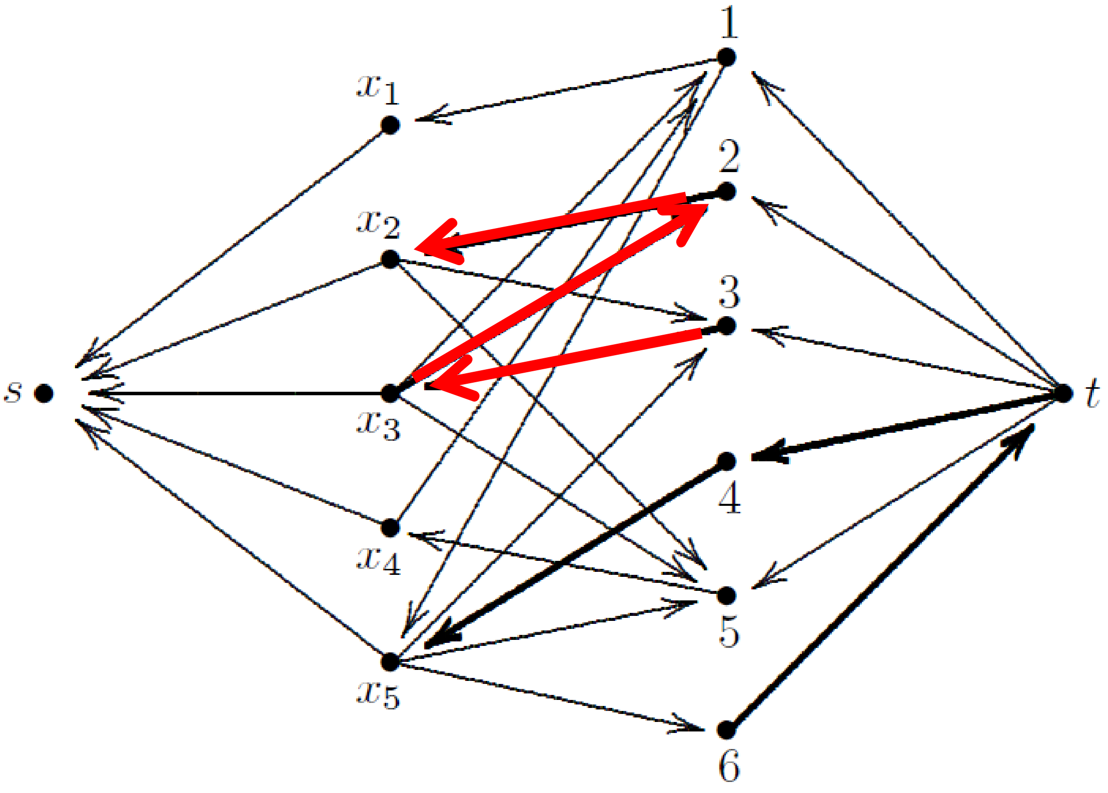


Residual graph for max flow

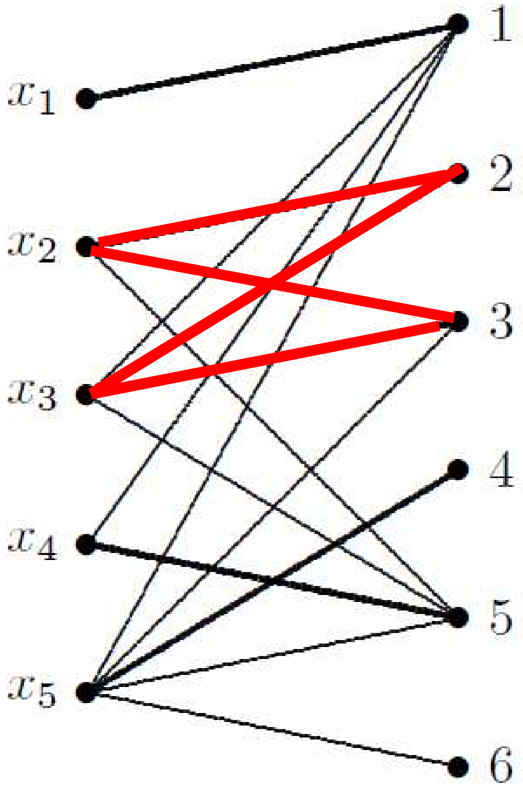


Domain filtering

- To filter domains, fix flow on return arc to 5.
  - Can 3 be removed from domain of x_2? Max flow from 3 to x_2 is 1, due to augmenting path.



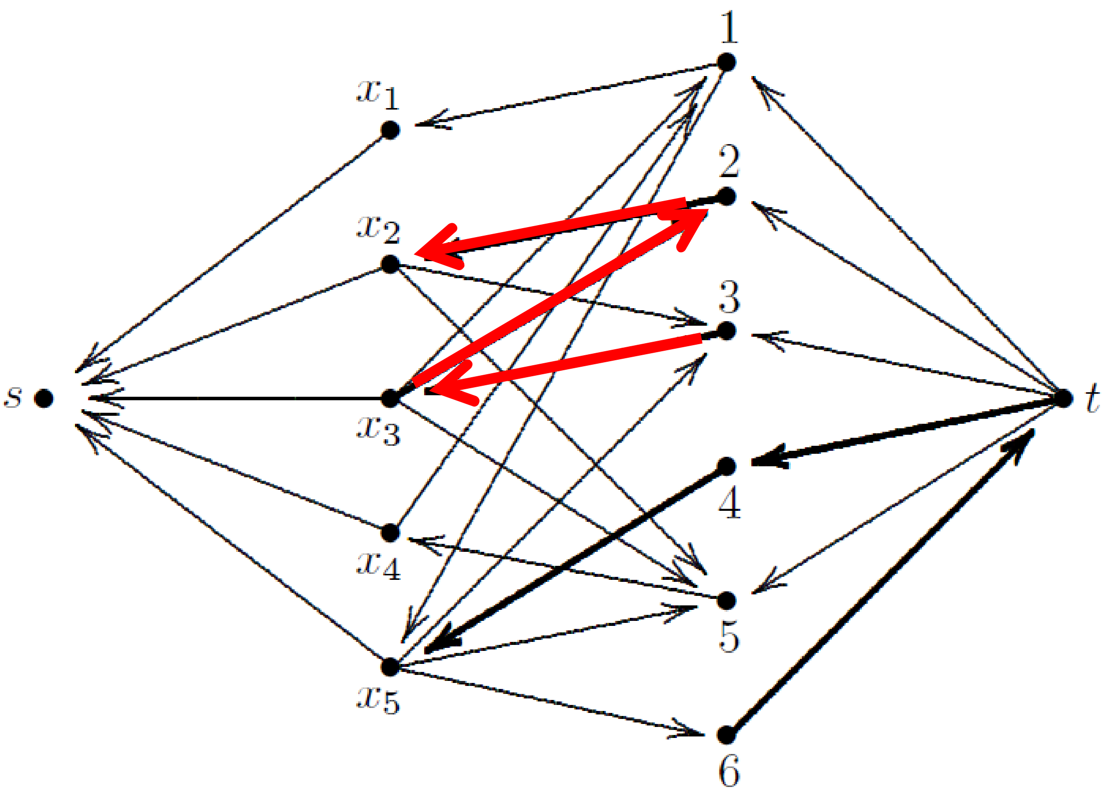
Augmenting path from 3 to x_2



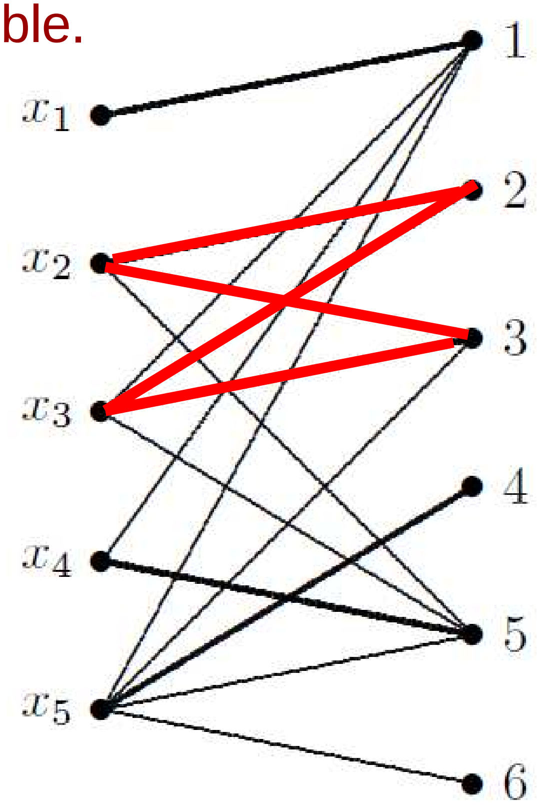
Alternating cycle

Domain filtering

- To filter domains, fix flow on return arc to 5.
    - Can 3 be removed from domain of x_2? Max flow from 3 to x_2 is 1, due to augmenting path. So x_2 = 3 is possible.



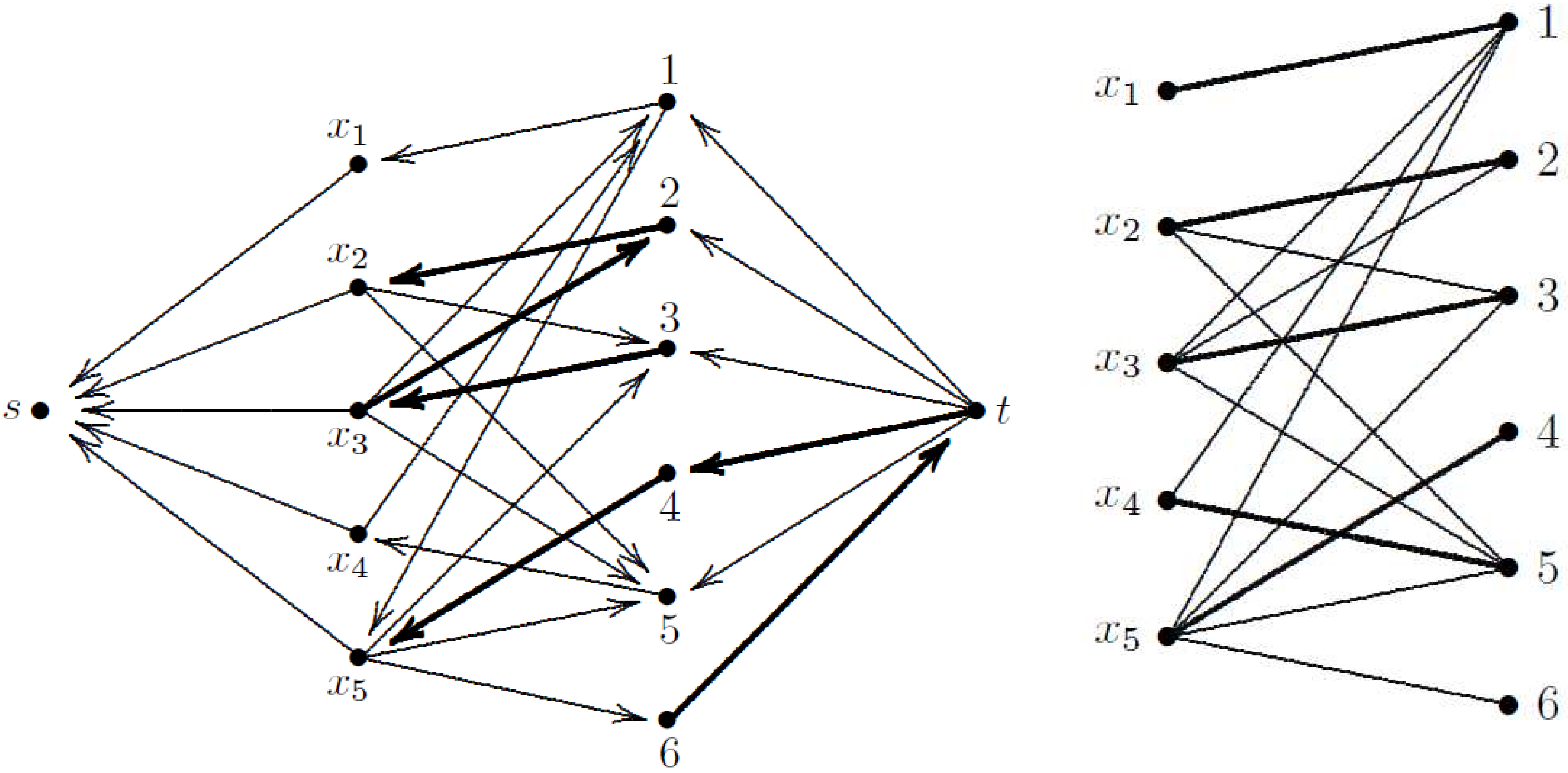
Augmenting path from 3 to x_2



Alternating cycle

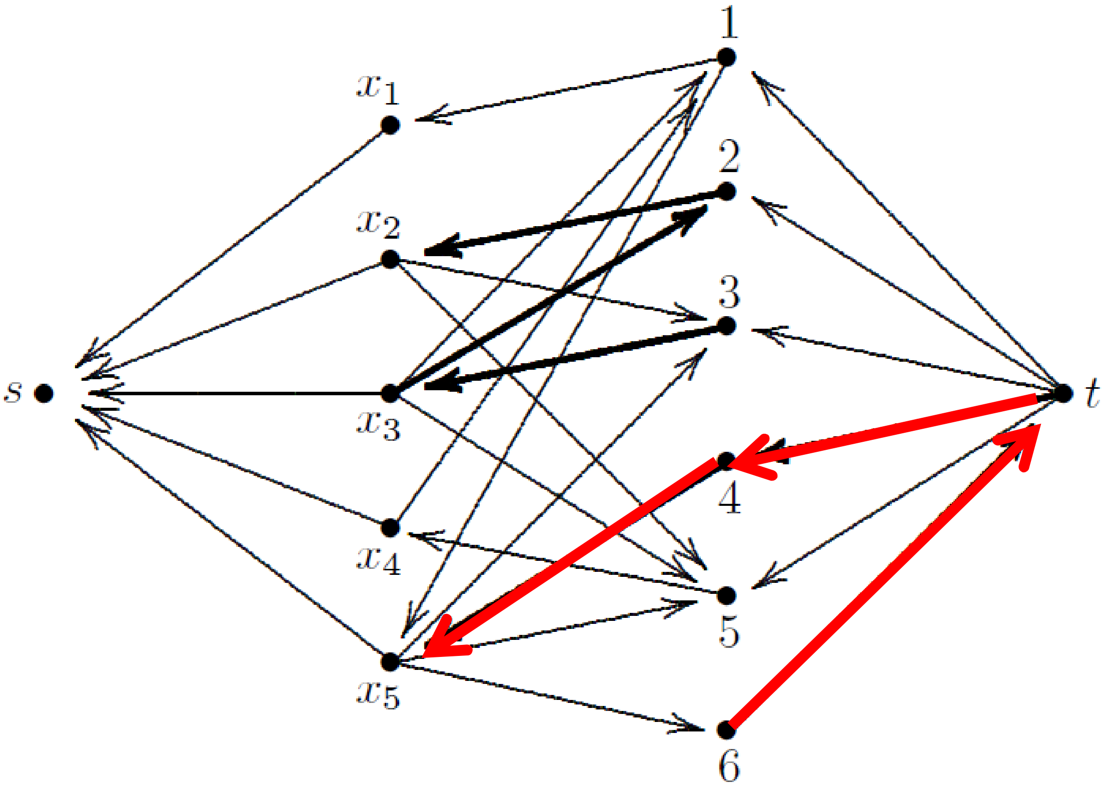
Domain filtering

- Fix flow on return arc in max flow model to 5.
  - Can 6 be removed from domain of x_5?

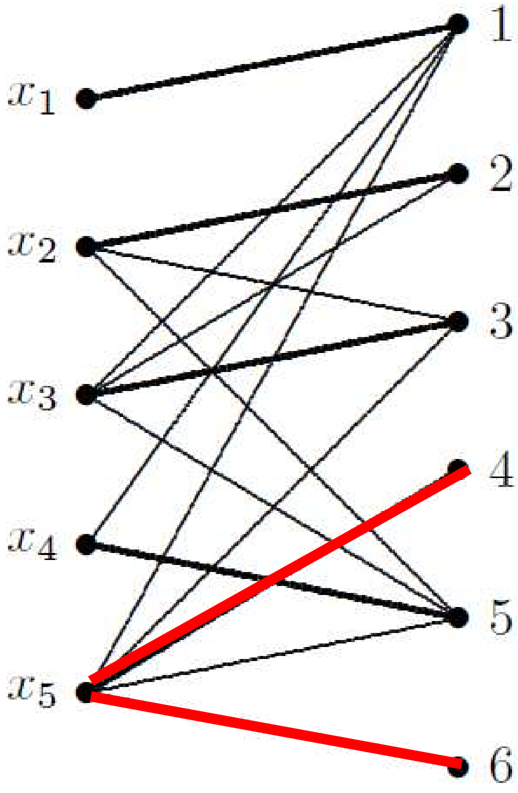


Domain filtering

- Fix flow on return arc in max flow model to 5.
  - Can 6 be removed from domain of x_5? No, because max flow from 6 to x_5 is 1, so that x_5 = 6 is possible.



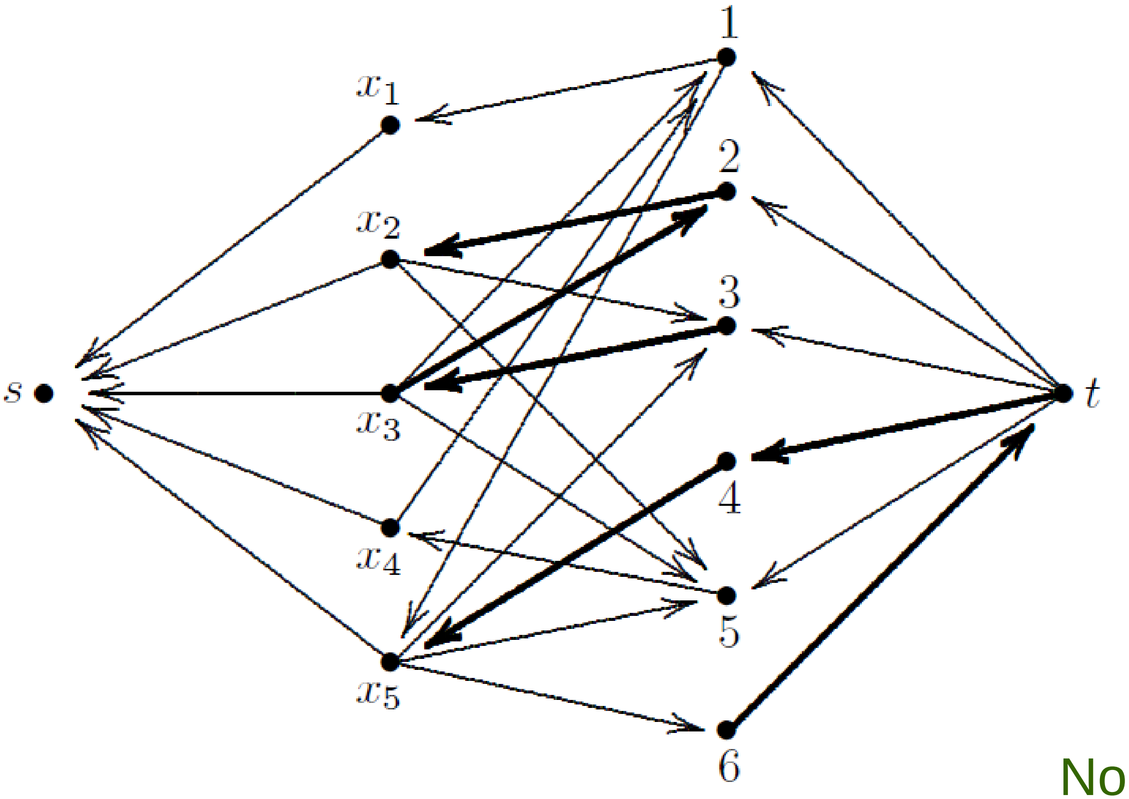
Augmenting path from x_2 to 2

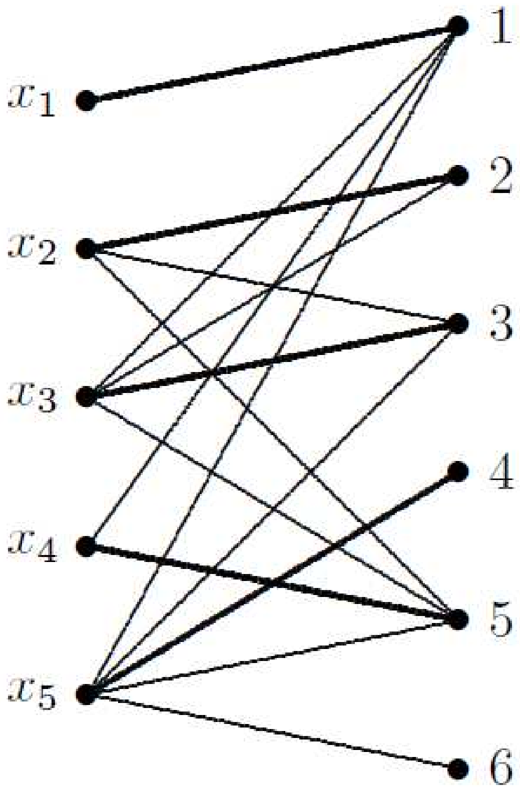


Even alternating path starting at uncovered vertex

Domain filtering

- Fix flow on return arc in max flow model to 5.
  - Can 1 be removed from domain of x_3? Yes, because there is no augmenting path from 1 to x_3.

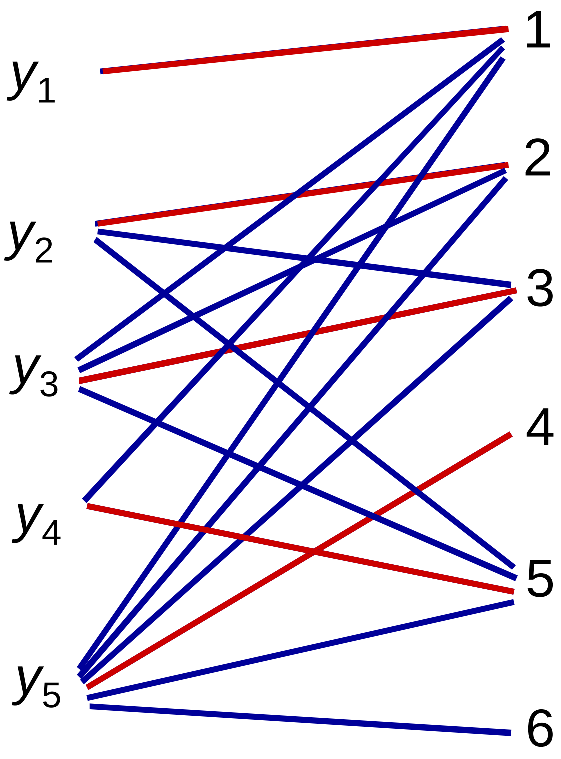




No alternating cycle or even alternating path containing (x_3,1)

Domain filtering

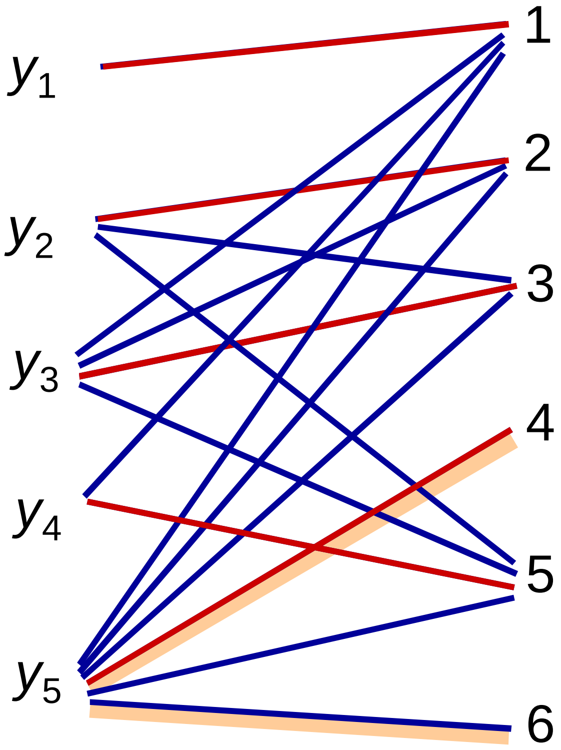
- We can filter $x_i = j$ when (x_i, j) belongs to no alternating cycle or even alternating path.



Mark edges in even alternating paths that start at an uncovered vertex.

Domain filtering

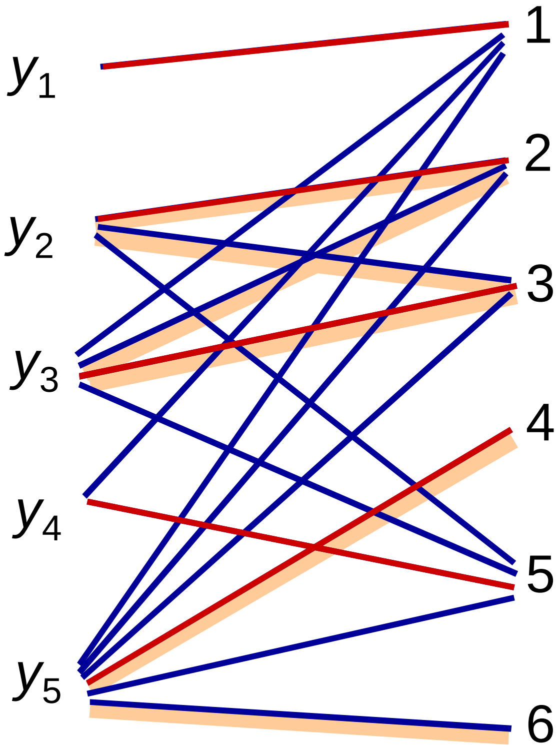
- We can filter $x_i = j$ when (x_i,j) belongs to no alternating cycle or even alternating path.



Mark edges in even alternating paths that start at an uncovered vertex.

Domain filtering

- We can filter $x_i = j$ when (x_i, j) belongs to no alternating cycle or even alternating path.

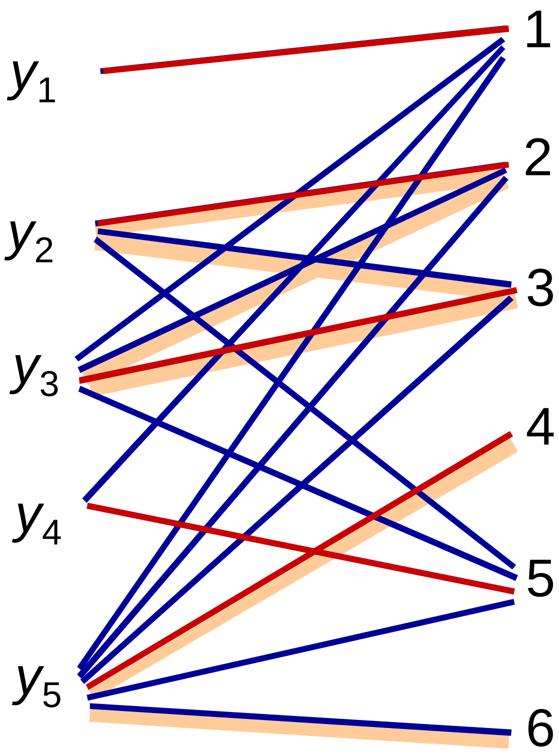


Mark edges in even alternating paths that start at an uncovered vertex.

Mark edges in alternating cycles.

Domain filtering

- We can filter $x_i = j$ when (x_i,j) belongs to no alternating cycle or even alternating path.



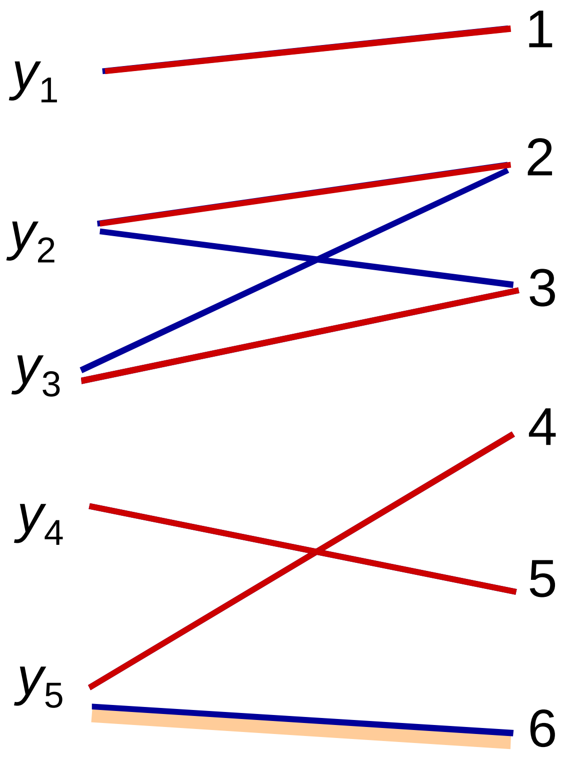
Mark edges in even alternating paths that start at an uncovered vertex.

Mark edges in alternating cycles.

Remove unmarked edges not in matching.

Domain filtering

- We can filter $x_i = j$ when (x_i,j) belongs to no alternating cycle or even alternating path.



Mark edges in alternating paths that start at an uncovered vertex.

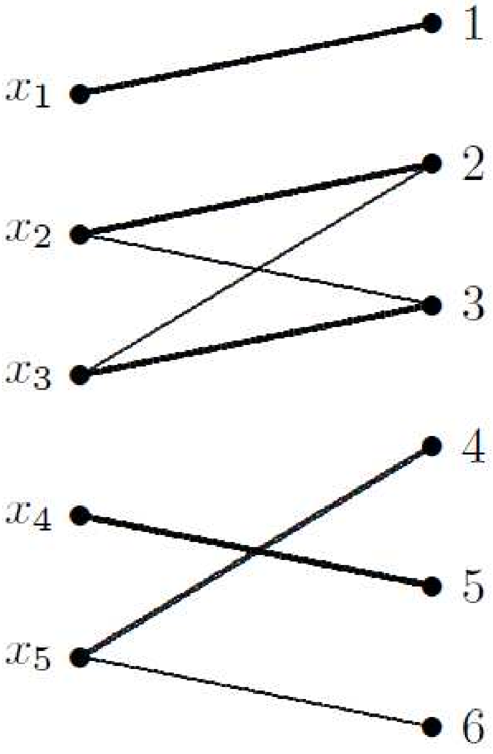
Mark edges in alternating cycles.

Remove unmarked edges not in matching.

Domain filtering

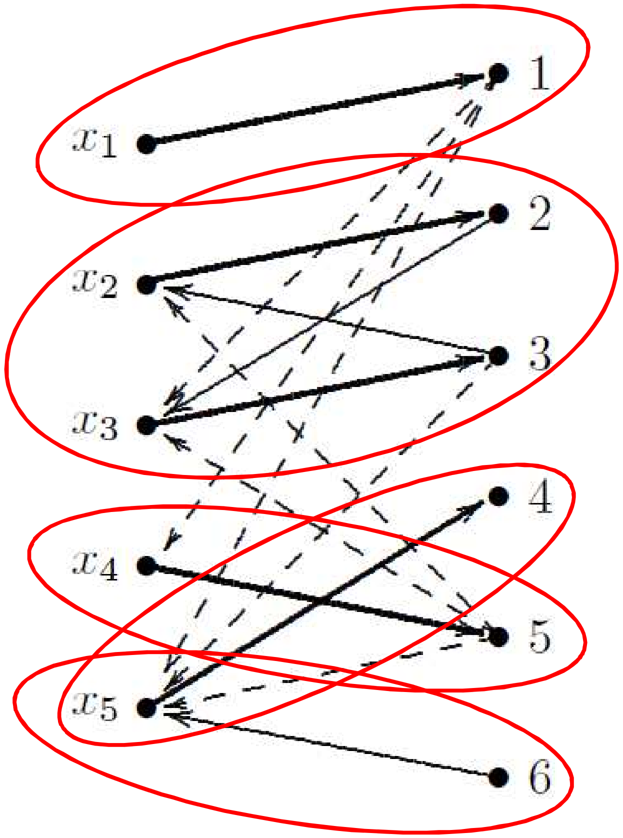
- Filtered domains:

$$x_1 \in \{1\}$$
$$x_2 \in \{2,3\}$$
$$x_3 \in \{2,3\}$$
$$x_4 \in \{5\}$$
$$x_5 \in \{4,6\}$$



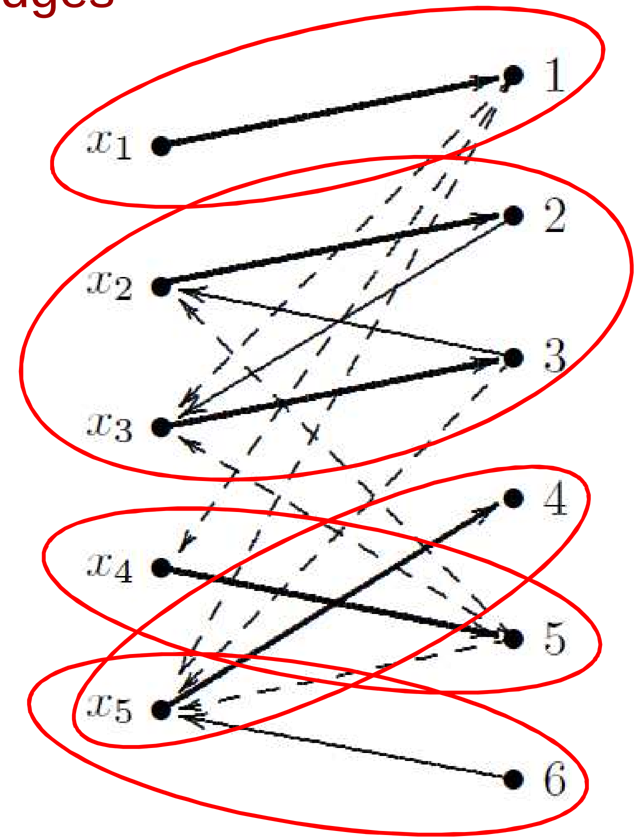
Domain filtering

- Algorithmically, identify strongly connected components of directed bipartite graph.
    - Edge directions are the same as in the residual graph.



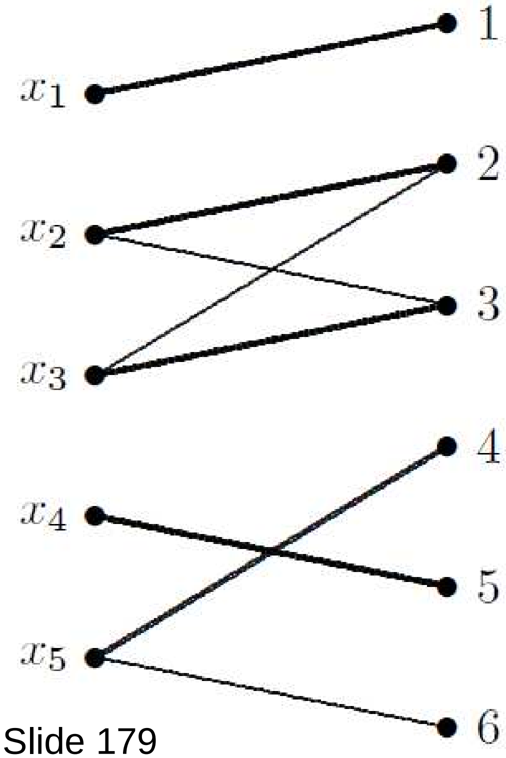
Domain filtering

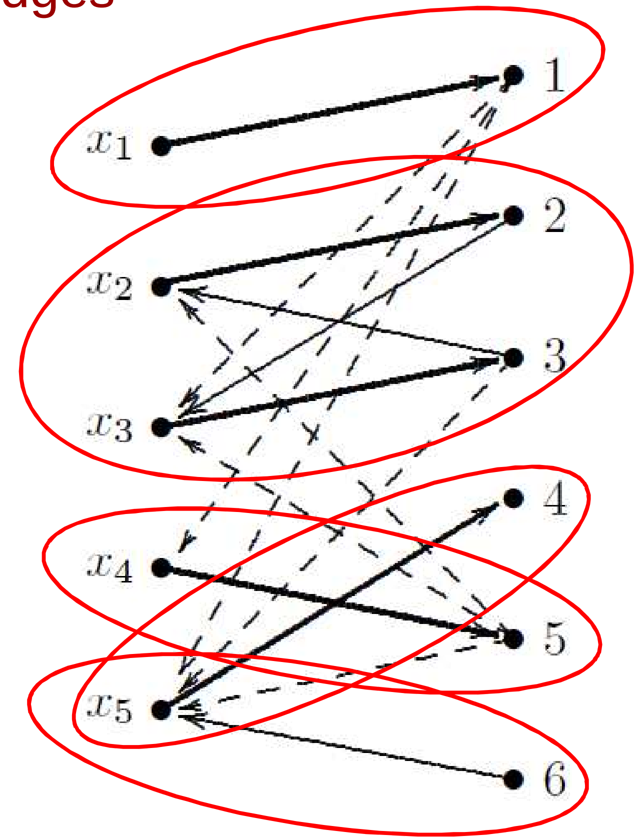
- Algorithmically, identify strongly connected components of directed bipartite graph.
  - Keep edges in matching or on directed paths starting at uncovered vertices, and edges inside a strongly connected component. Remove all other edges



Domain filtering

- Algorithmically, identify strongly connected components of directed bipartite graph.
    - Keep edges in matching or on directed paths starting at uncovered vertices, and edges inside a strongly connected component. Remove all other edges





Bounds Consistency

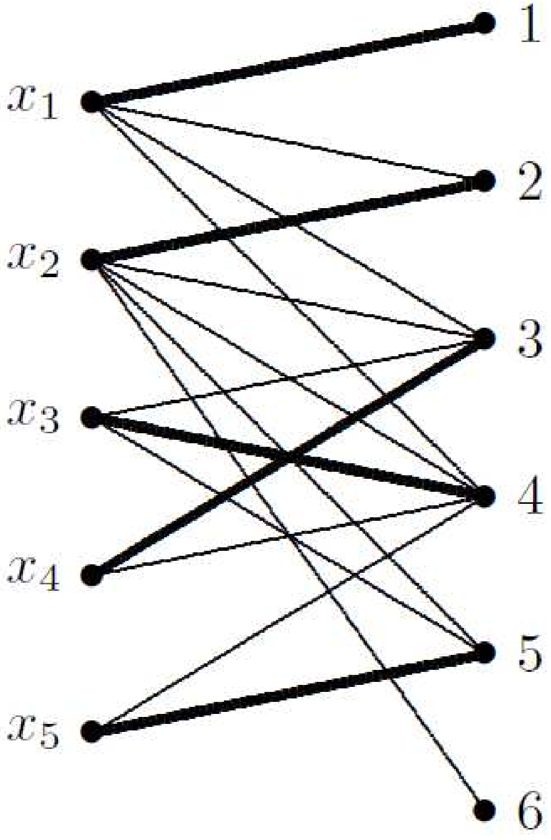
- **Bounds consistency** is easier to achieve for alldiff than domain consistency.
    - Bipartite graph has a convexity property.

Bounds Consistency

- Replace domains with intervals {L_j,,…,U_j}.

$$\text{alldiff}\left(x_1,x_2,x_3,x_4,x_5\right)$$

Domains	Intervals
$x_1 \in \{1,2,4\}$	$x_1 \in \{1,2,3,4\}$
$x_2 \in \{2,3,6\}$	$x_2 \in \{2,3,4,5,6\}$
$x_3 \in \{3,5\}$	$x_3 \in \{3,4,5\}$
$x_4 \in \{3,4\}$	$x_4 \in \{3,4\}$
$x_5 \in \{4,5\}$	$x_5 \in \{4,5\}$



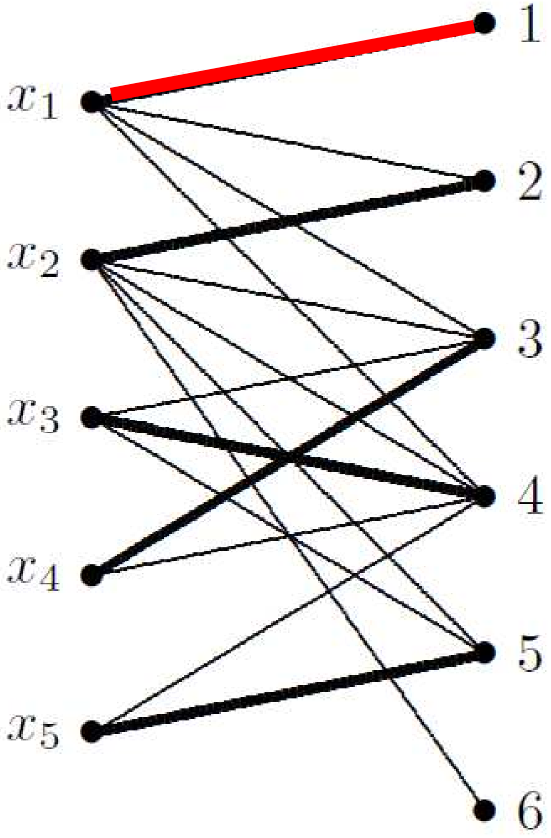
Bipartite graph is “convex.”

Bounds Consistency

- Find initial solution for purposes of achieving bounds consistency.
  - This can be done in O(# variables) time.

Cover 1 using $(x_j,1)$ with smallest U_j.

$U_1 = 4$



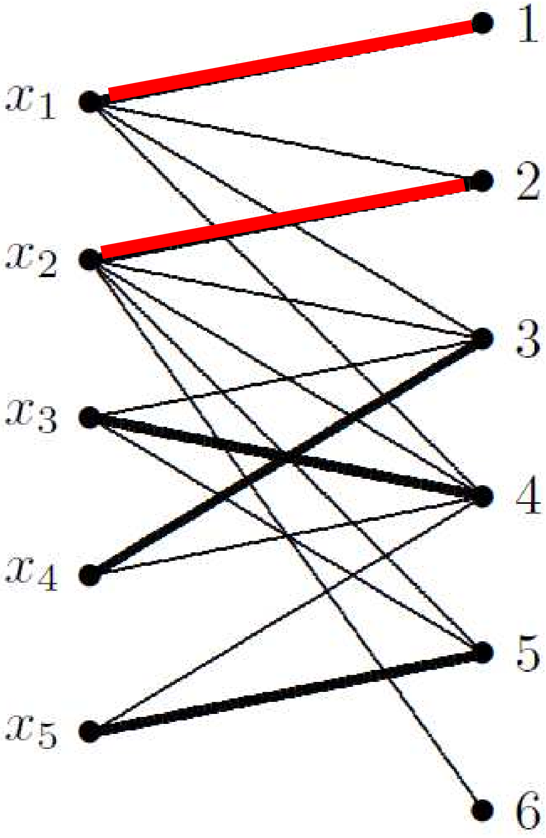
Bounds Consistency

- Find initial solution for purposes of achieving bounds consistency.
    - This can be done in O(# variables) time.

Cover 1 using $(x_j,1)$ with smallest U_j.

Cover 2 using $(x_j,2)$ with smallest U_j.

$U_2 = 6$



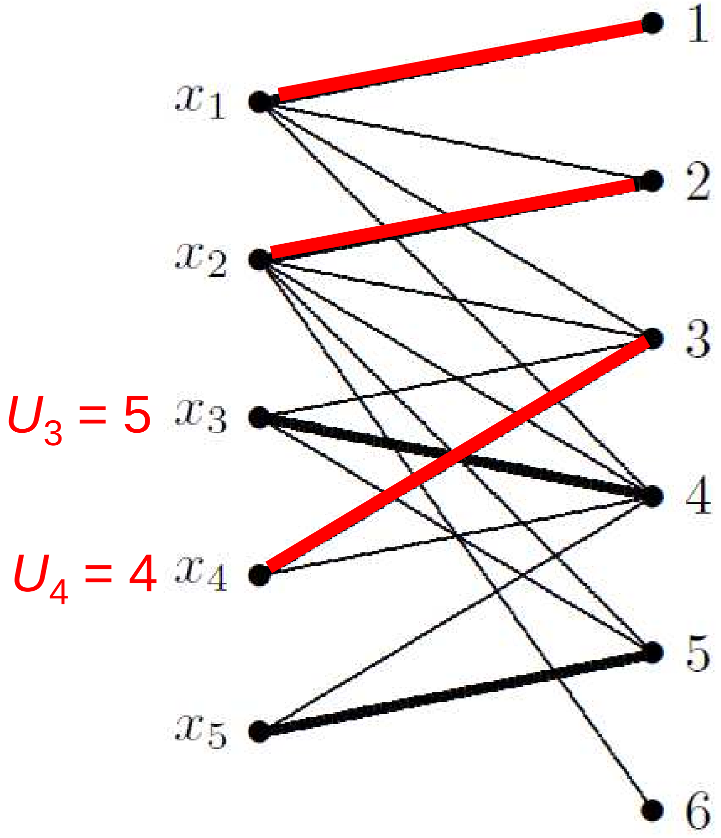
Bounds Consistency

- Find initial solution for purposes of achieving bounds consistency.
  - This can be done in O(# variables) time.

Cover 1 using (x_j,1) with smallest U_j.

Cover 2 using (x_j,2) with smallest U_j.

Cover 3 using (x_j,3) with smallest U_j.



Bounds Consistency

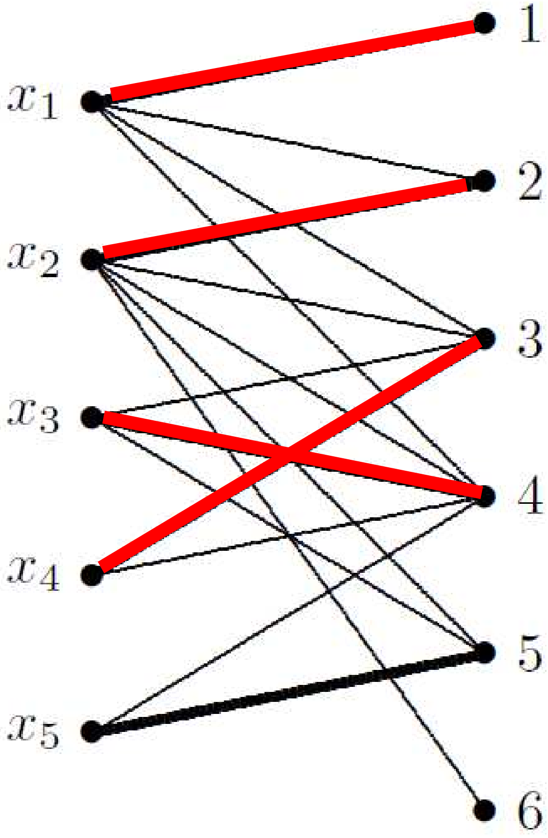
- Find initial solution for purposes of achieving bounds consistency.
  - This can be done in O(# variables) time.

Cover 1 using $(x_j, 1)$ with smallest U_j.

Cover 2 using $(x_j, 2)$ with smallest U_j.

Cover 3 using $(x_j, 3)$ with smallest U_j.

Cover 4 using $(x_j, 4)$ with smallest U_j.



$U_3 = 5$

$U_5 = 5$

Bounds Consistency

- Find initial solution for purposes of achieving bounds consistency.
    - This can be done in O(# variables) time.

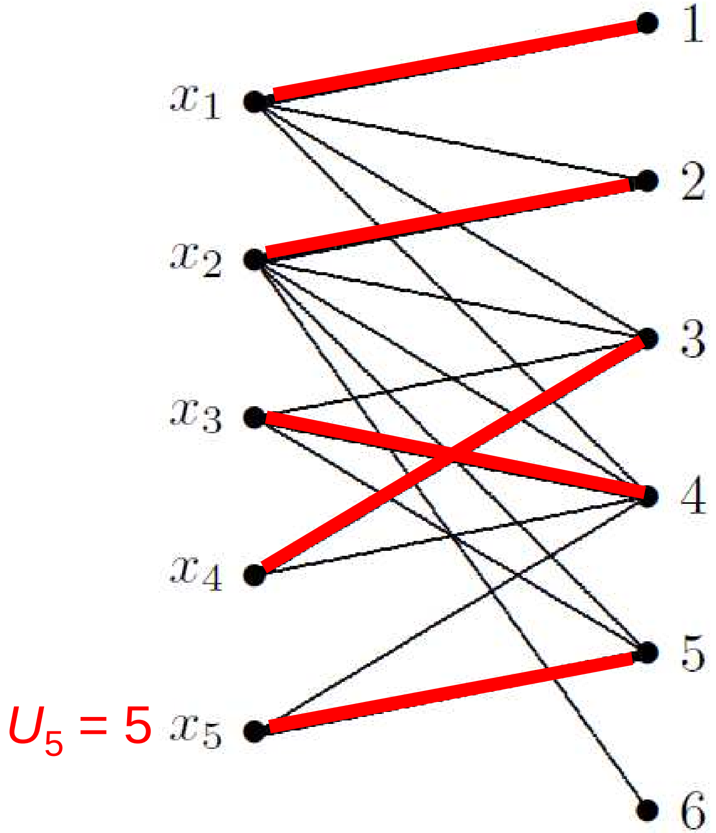
Cover 1 using (x_j,1) with smallest U_j.

Cover 2 using (x_j,2) with smallest U_j.

Cover 3 using (x_j,3) with smallest U_j.

Cover 4 using (x_j,4) with smallest U_j.

Cover 5 using (x_j,5) with smallest U_j.



Bounds Consistency

- Find initial solution for purposes of achieving bounds consistency.
    - This can be done in O(# variables) time.

Cover 1 using (x_j,1) with smallest U_j.

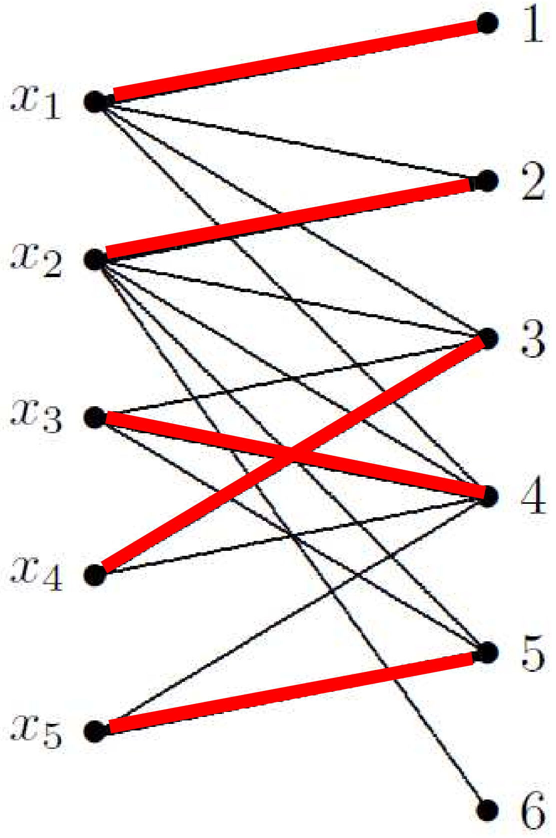
Cover 2 using (x_j,2) with smallest U_j.

Cover 3 using (x_j,3) with smallest U_j.

Cover 4 using (x_j,4) with smallest U_j.

Cover 5 using (x_j,5) with smallest U_j.

(Skip vertices on right that can't be covered.) Now we are done.



Bounds Consistency

- Now filter domains using max flow model as before.

Domains

$x_1 \in \{1,2,4\}$

$x_2 \in \{2,3,6\}$

$x_3 \in \{3,5\}$

$x_4 \in \{3,4\}$

$x_5 \in \{4,5\}$

Reduced domains

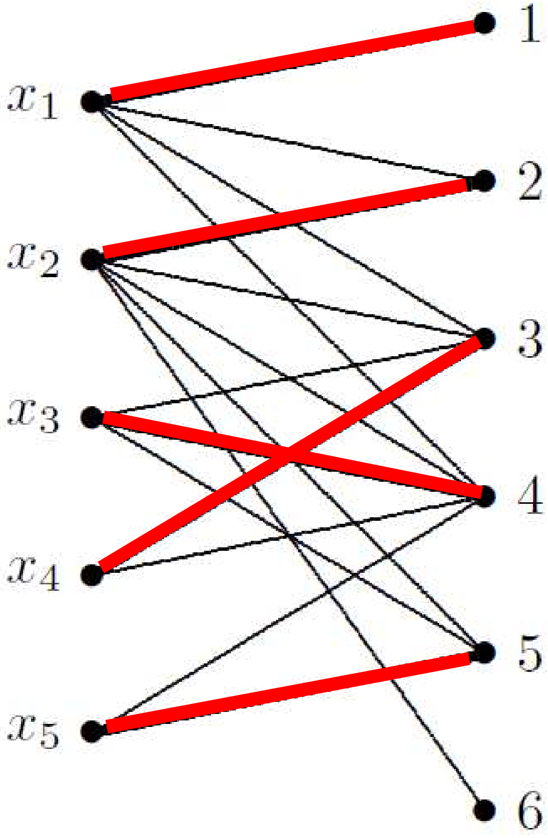
$x_1 \in \{1,2\}$

$x_2 \in \{2,3,6\}$

$x_3 \in \{3,5\}$

$x_4 \in \{3,4\}$

$x_5 \in \{4,5\}$



Cardinality Constraint

Network Flow Model
Domain Consistency
Nvalues Constraint

Cardinality constraint

- The **cardinality constraint** limits the number of variables $x_1, \ldots, x_n$ that take specified values.

$$\text{cardinality}\left((x_1,\ldots,x_n),v,\ell,u\right)$$

  - Requires that $\ell_i \le |\{\, j \mid x_j = v_i\}| \le u_i$ for $i = 1, \ldots, m$, where

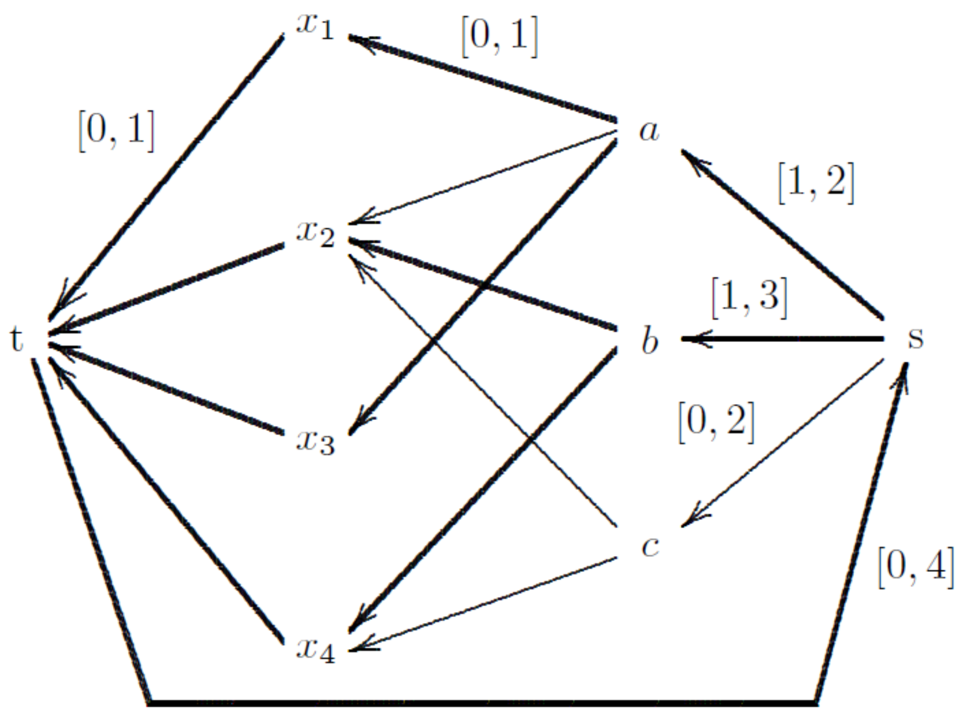
    $v = (v_1, \ldots, v_m)$, $\ell = (\ell_1, \ldots, \ell_m)$, and $u = (u_1, \ldots, u_m)$.

  - Also called **generalized cardinality constraint** or **gcc**.

- **Cardinality** can be filtered using optimality conditions for max flow, similar to **alldiff**.

Cardinality constraint

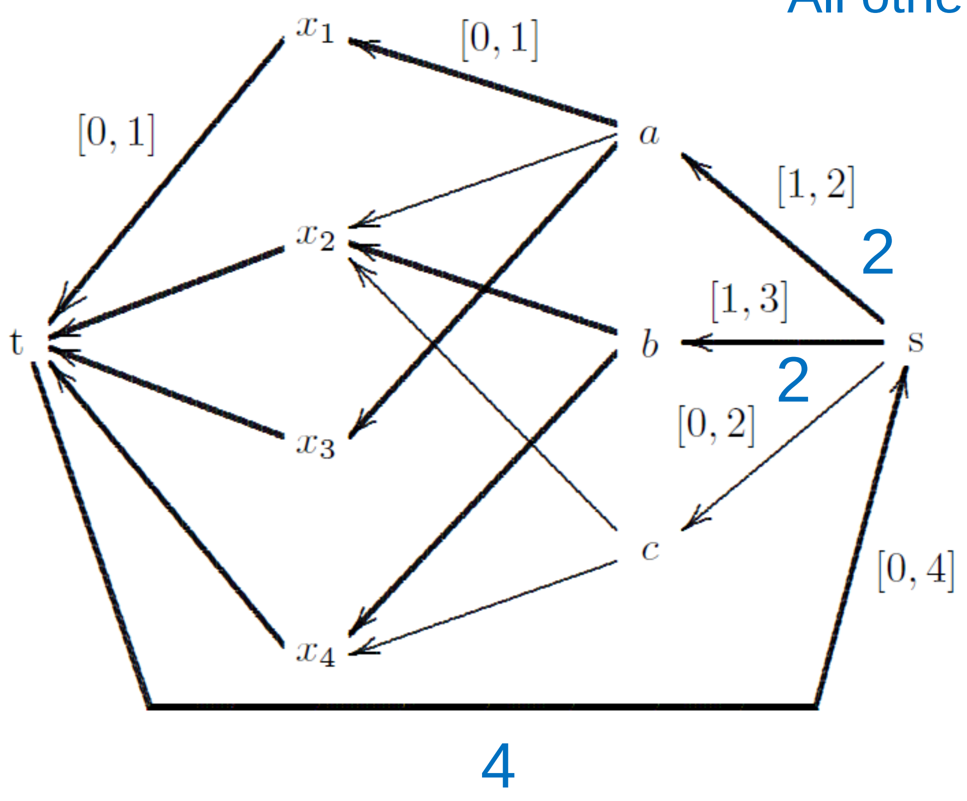
- Example. $\text{cardinality}\big((x_1,x_2,x_3,x_4),(a,b,c),(1,1,0),(2,3,2)\big)$
  - It has a solution if and only if there is a feasible flow:



Cardinality constraint

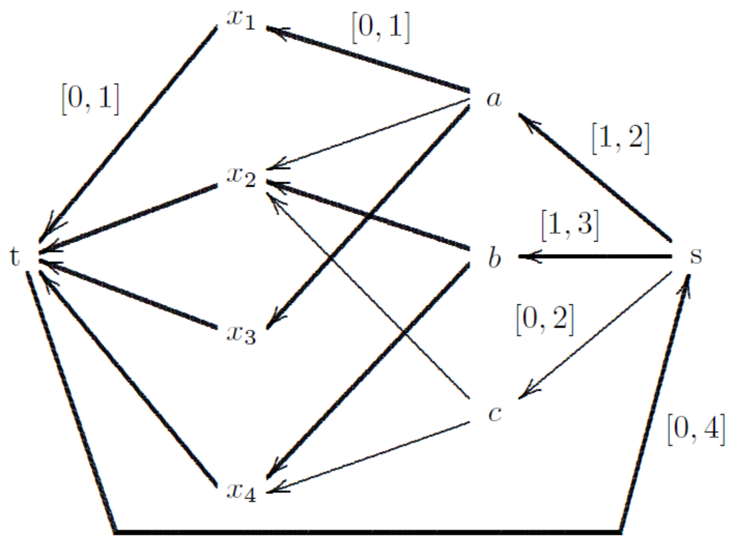
- Example. $\text{cardinality}\big((x_1,x_2,x_3,x_4),(a,b,c),(1,1,0),(2,3,2)\big)$
  - It has a solution if and only if there is a max flow of 4:

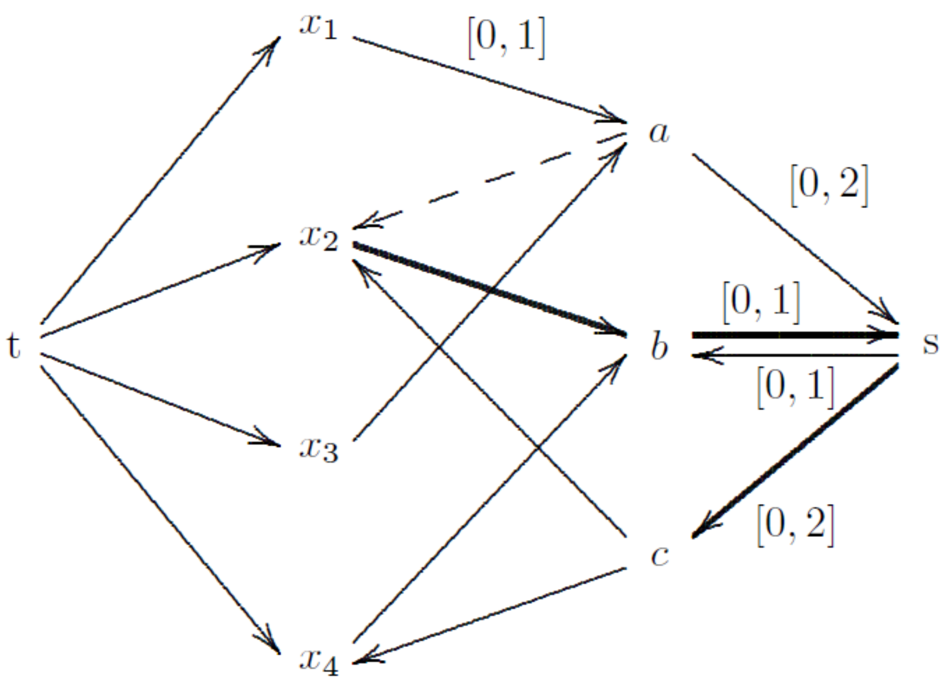
All other flows are 1



Cardinality constraint

- Example. $\text{cardinality}\big((x_1,x_2,x_3,x_4),(a,b,c),(1,1,0),(2,3,2)\big)$
    - Can $x_2 = c$?

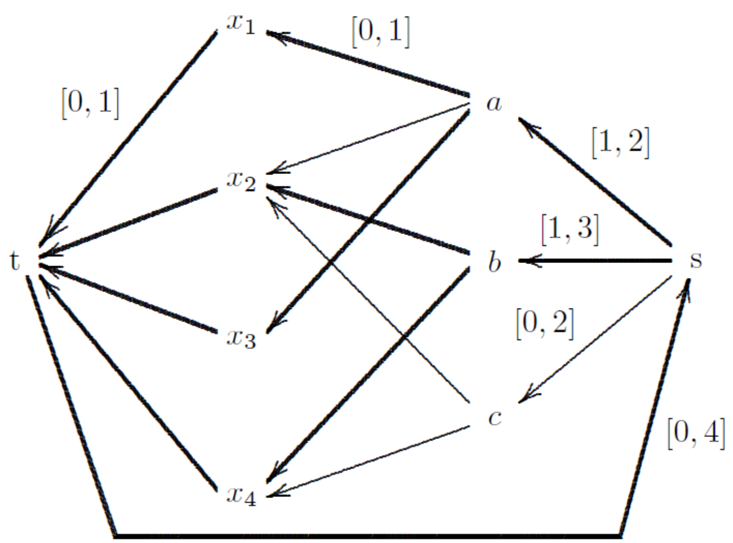


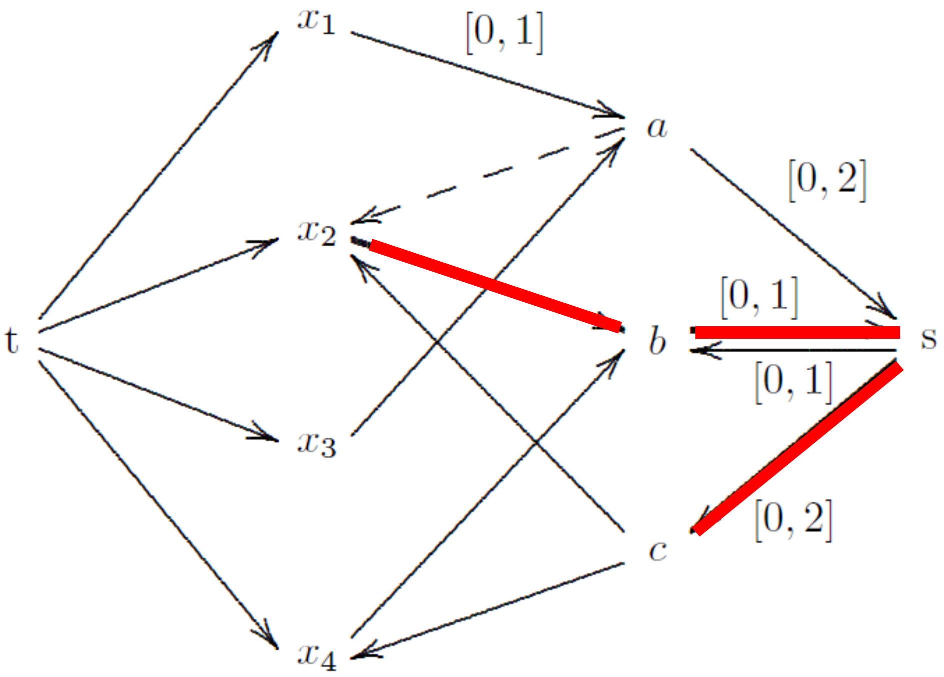


Residual graph

Cardinality constraint

- Example. $\text{cardinality}\big((x_1,x_2,x_3,x_4),(a,b,c),(1,1,0),(2,3,2)\big)$
  - Can $x_2 = c$? Yes, because there is an augmenting path from x_2 to c. We cannot remove c from domain of x_2.

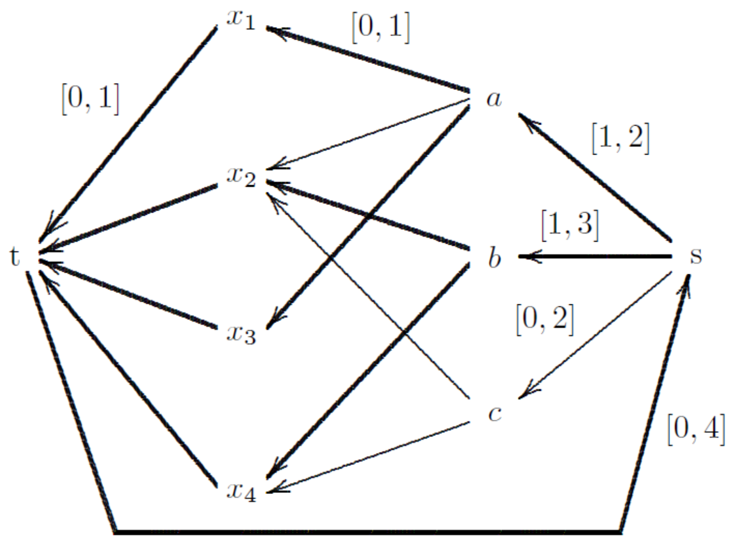


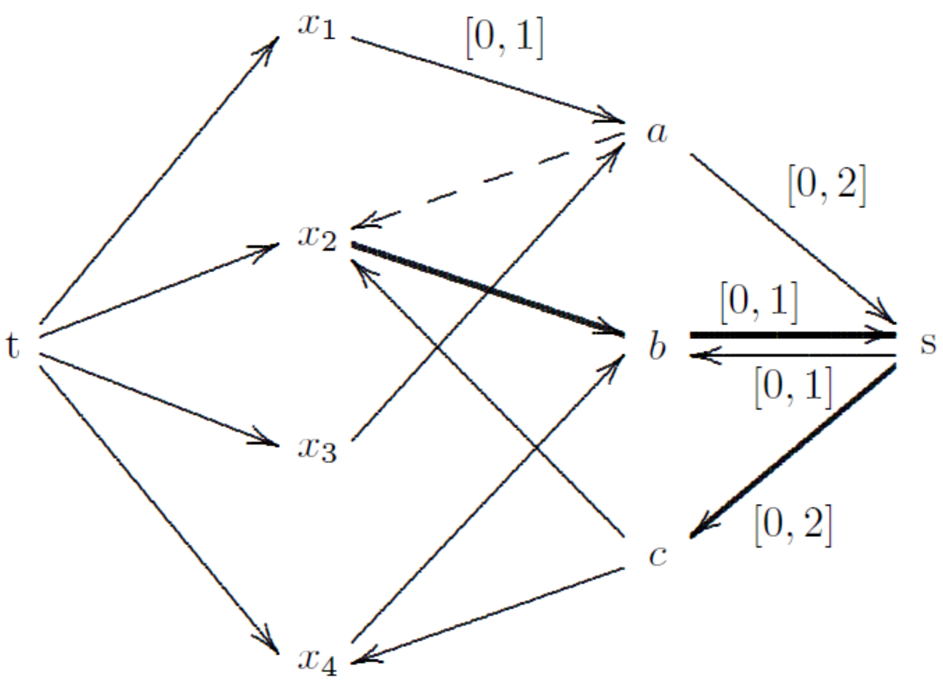


Residual graph

Cardinality constraint

- Example. $\text{cardinality}\left((x_1,x_2,x_3,x_4),(a,b,c),(1,1,0),(2,3,2)\right)$
  - Can $x_2 = a$? No, because there is no augmenting path from x_2 to a. We can remove a from domain of x_2.

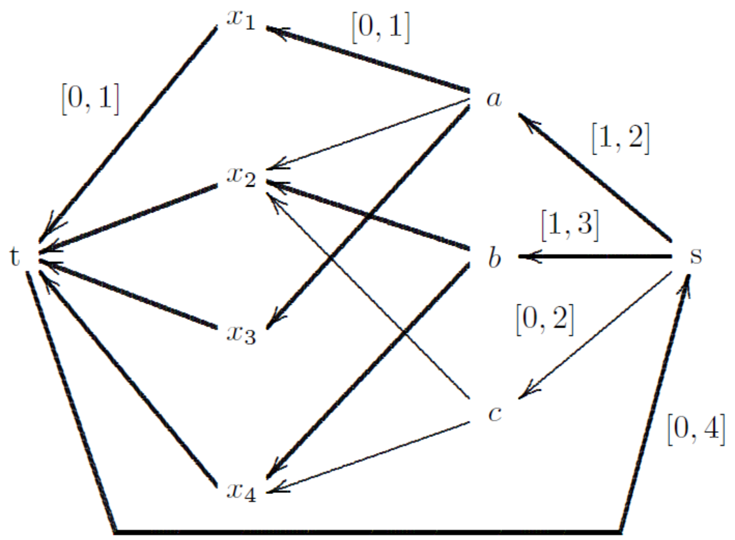


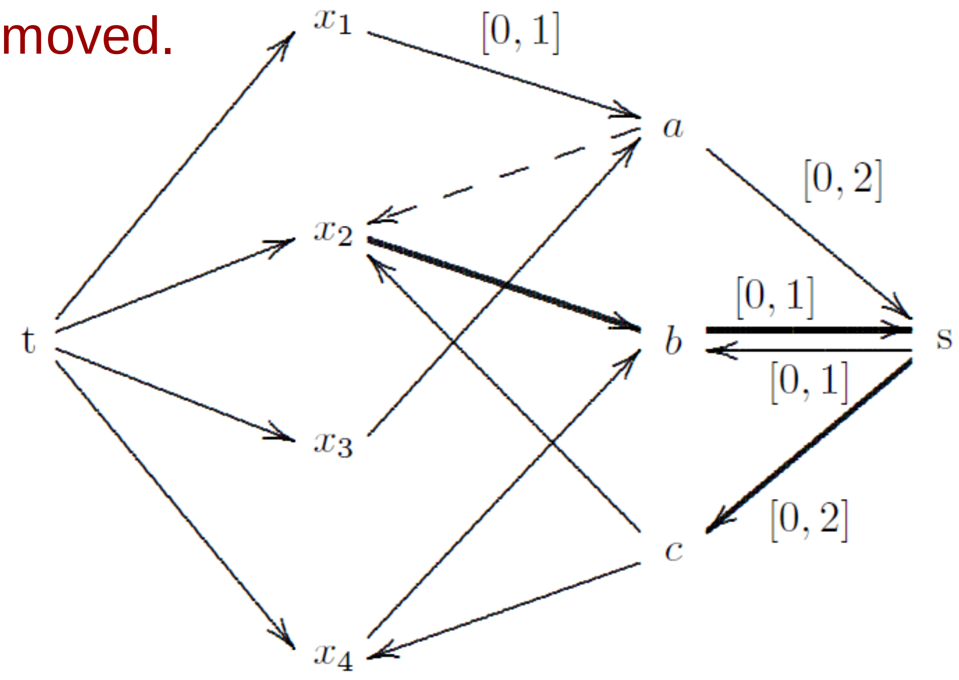


Residual graph

Cardinality constraint

- Example. $\text{cardinality}\big((x_1,x_2,x_3,x_4),(a,b,c),(1,1,0),(2,3,2)\big)$
  - Can $x_2 = a$? No, because there is no augmenting path from x_2 to a. We can remove a from domain of x_2.
  - No other values can be removed.





Residual graph

Nvalues constraint

- The **nvalues constraint** limits the number of different values taken by variables $x_1, \ldots, x_n$.

$$\text{nvalues}\big((x_1,\ldots,x_n),\ell,u\big)$$

  - Requires that $\ell \leq |\{x_1, \ldots, x_n\}| \leq u$
  - Becomes **alldiff** when $\ell = u = n$.

- Has a flow model similar to **cardinality.**

Sequence Constraint

Filtering Based on Cumulative Sums
Filtering Based on Network Flows

Sequence constraint

- The **sequence** constraint limits the number of 1s in each sequence of *q* consecutive binary variables.

$$\text{sequence}\big((y_1,\ldots,y_n),q,\ell,u\big)$$

- Requires that $\ell \le \sum_{i=j}^{j+q-1} y_i \le u, \quad j=1,\ldots,n-q+1$

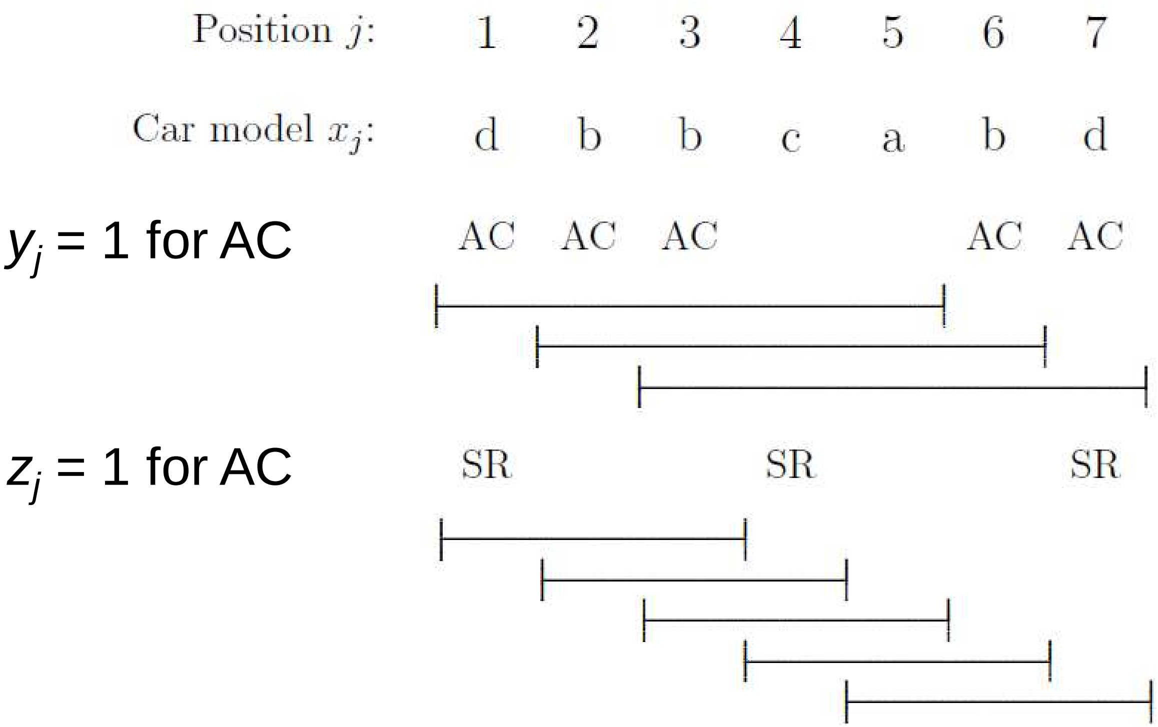
- There is a complete polytime filter (not obvious).

- Used in car sequencing and similar problems.

Sequence constraint

- Recall the car sequencing example.

$$\text{sequence}\big((y_1,\ldots,y_7),5,0,3\big)$$
$$\text{sequence}\big((z_1,\ldots,z_7),3,0,1\big)$$



Position j:	1	2	3	4	5	6	7
Car model x_j:	d	b	b	c	a	b	d
$y_j = 1$ for AC	AC	AC	AC			AC	AC
$z_j = 1$ for AC	SR			SR			SR

Filtering based on cumulative sums

- We first show how to find a feasible solution for **sequence.**
  - We will filter domains by “shaving,” i.e., removing domain elements one at a time and checking whether there is a feasible solution.

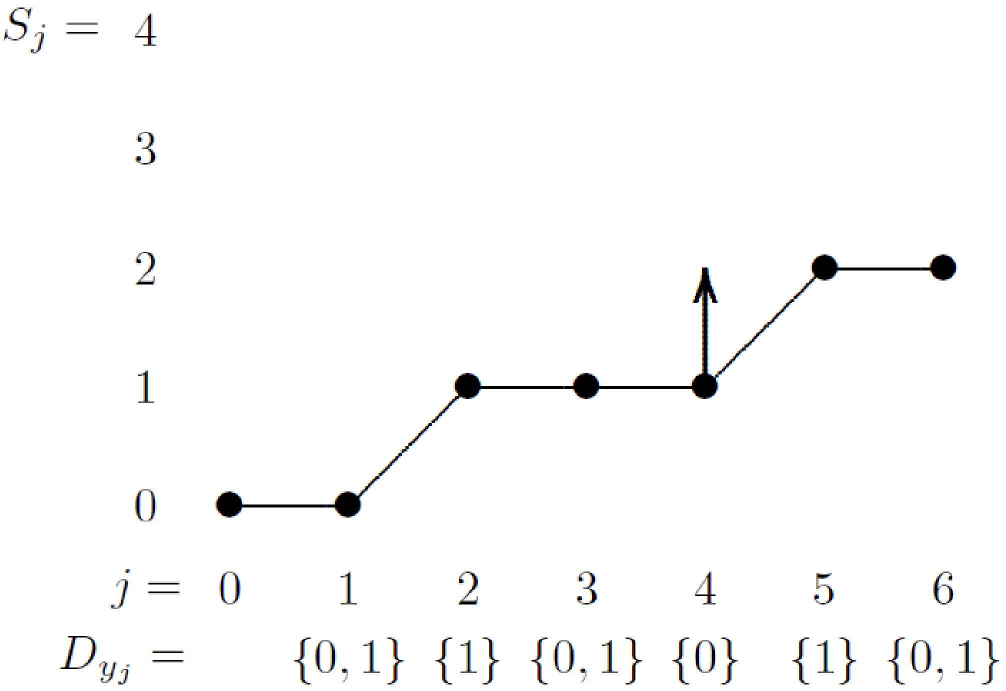
- Define the partial sum $S_j = \sum_{i=1}^{j} y_i$
  - So sequence(y,q,ℓ,u) says $\ell \leq S_j - S_{\mathrm{j-q}} \leq u$ for $j = q,\ldots,n.$

Filtering based on cumulative sums

- Example $\text{sequence}((y_1,\ldots,y_6),4,2,2)$

$y_1 \in \{0,1\}$ $y_2 \in \{1\}$ $y_3 \in \{0,1\}$ $y_4 \in \{0,1\}$ $y_5 \in \{1\}$ $y_6 \in \{0,1\}$

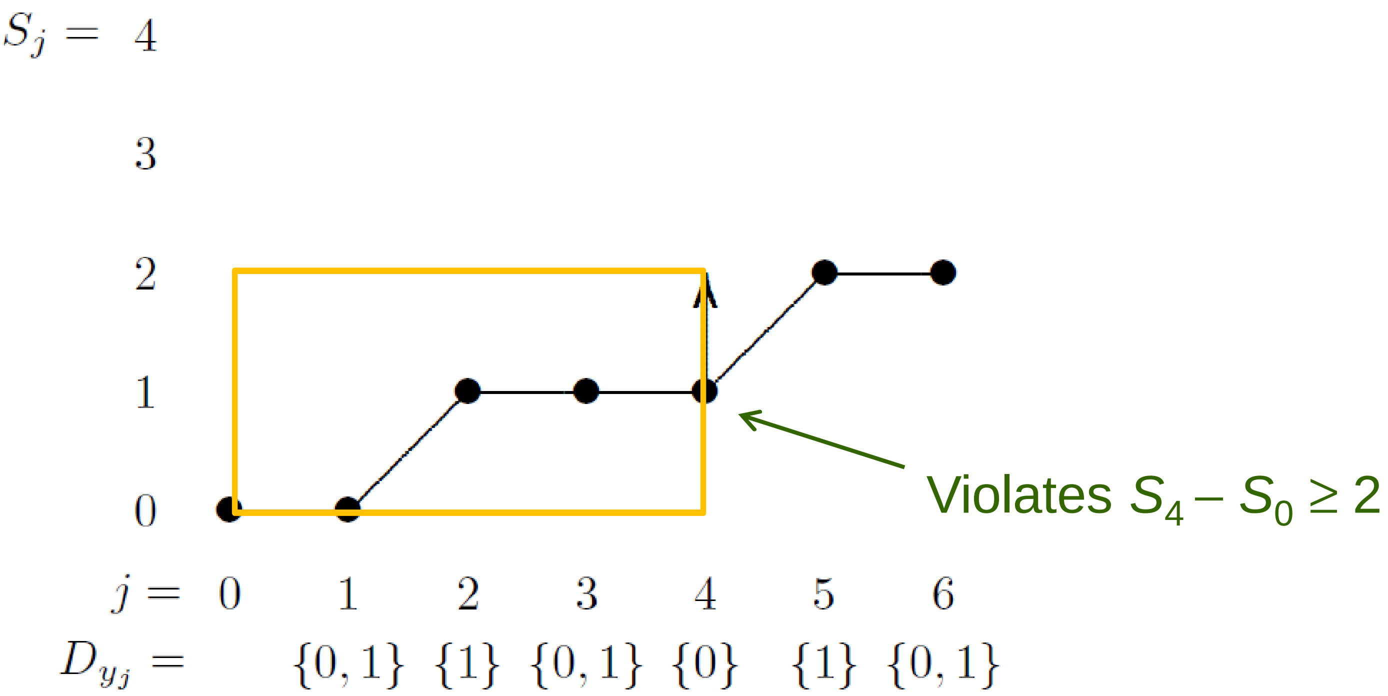
  - First set each y_i to smallest value in its domain.



Filtering based on cumulative sums

- Example $\text{sequence}((y_1,\ldots,y_6),4,2,2)$

$y_1 \in \{0,1\} \quad y_2 \in \{1\} \quad y_3 \in \{0,1\} \quad y_4 \in \{0,1\} \quad y_5 \in \{1\} \quad y_6 \in \{0,1\}$



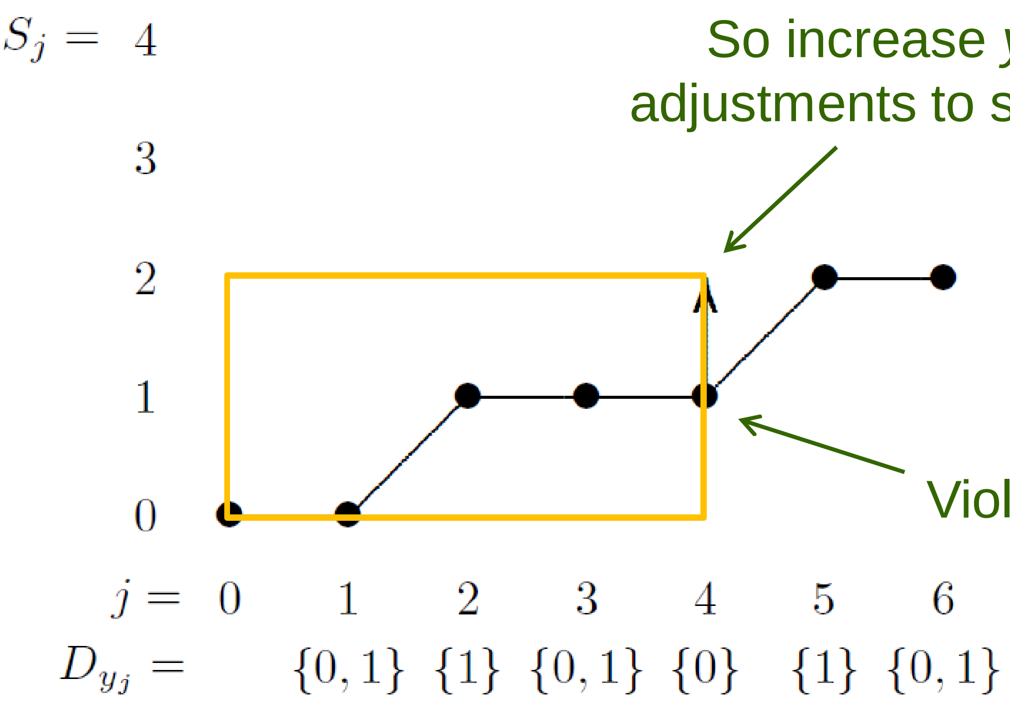
$S_j =$
4
3
2
1
0
$j =$ 0 1 2 3 4 5 6
$D_{y_j} =$ $\{0,1\}$ $\{1\}$ $\{0,1\}$ $\{0\}$ $\{1\}$ $\{0,1\}$

Violates $S_4 - S_0 \geq 2$

Filtering based on cumulative sums

- Example $\text{sequence}\left((y_1,\ldots,y_6),4,2,2\right)$

$y_1 \in \{0,1\} \quad y_2 \in \{1\} \quad y_3 \in \{0,1\} \quad y_4 \in \{0,1\} \quad y_5 \in \{1\} \quad y_6 \in \{0,1\}$



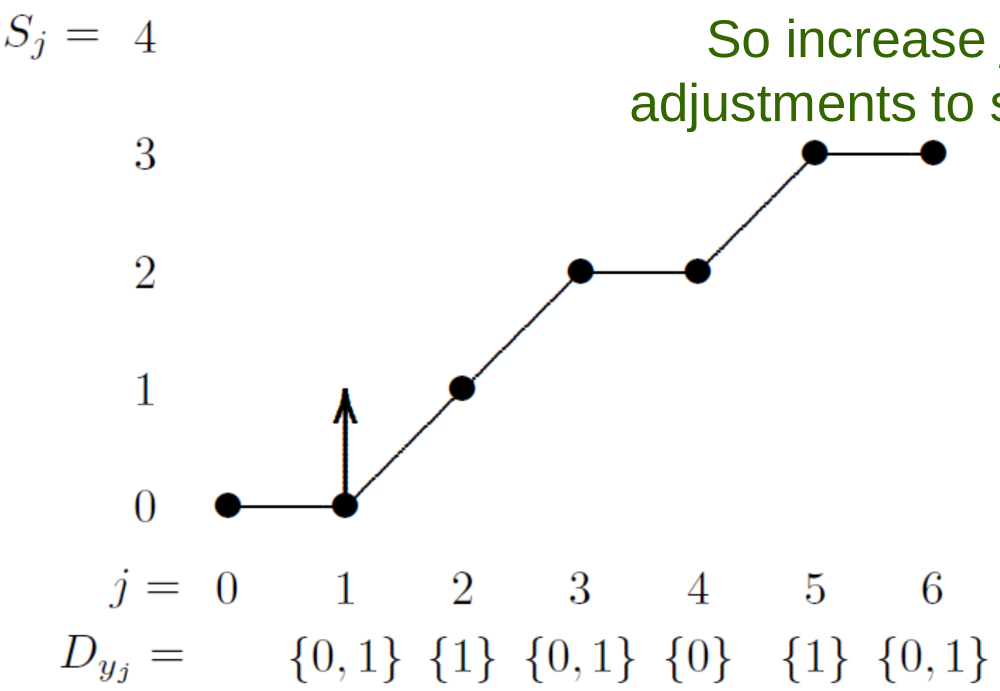
So increase y_4 and make adjustments to stay in domains.

Violates $S_4 - S_0 \geq 2$

Filtering based on cumulative sums

- Example $\text{sequence}\left((y_1,\ldots,y_6),4,2,2\right)$

$y_1 \in \{0,1\} \quad y_2 \in \{1\} \quad y_3 \in \{0,1\} \quad y_4 \in \{0,1\} \quad y_5 \in \{1\} \quad y_6 \in \{0,1\}$

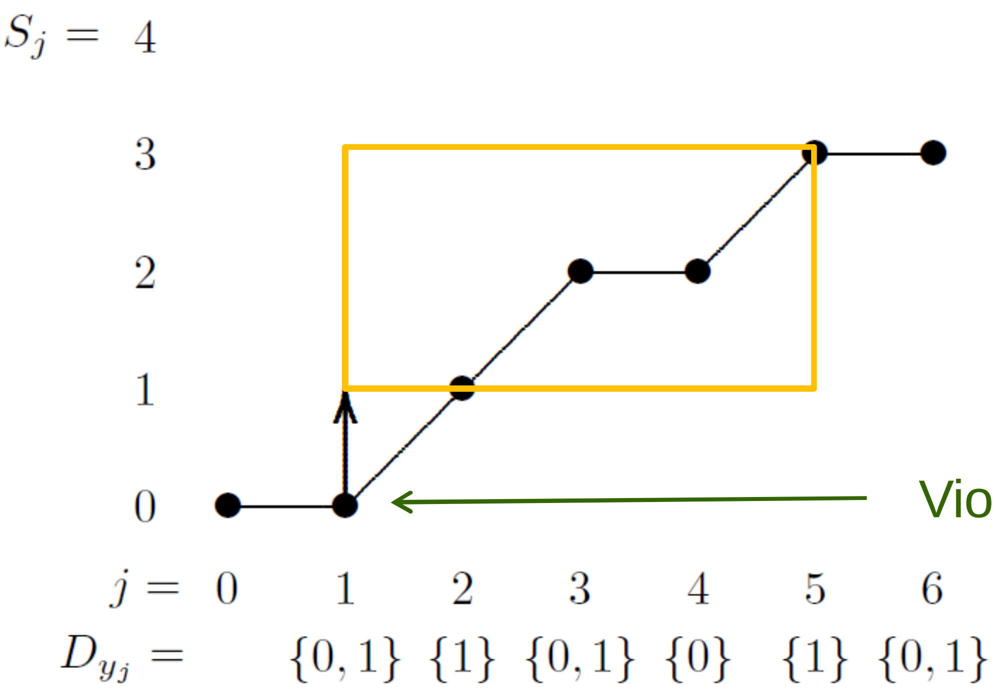


So increase y_4 and make adjustments to stay in domains.

Filtering based on cumulative sums

- Example $\text{sequence}\left((y_1,\ldots,y_6),4,2,2\right)$

$y_1 \in \{0,1\}$ $y_2 \in \{1\}$ $y_3 \in \{0,1\}$ $y_4 \in \{0,1\}$ $y_5 \in \{1\}$ $y_6 \in \{0,1\}$

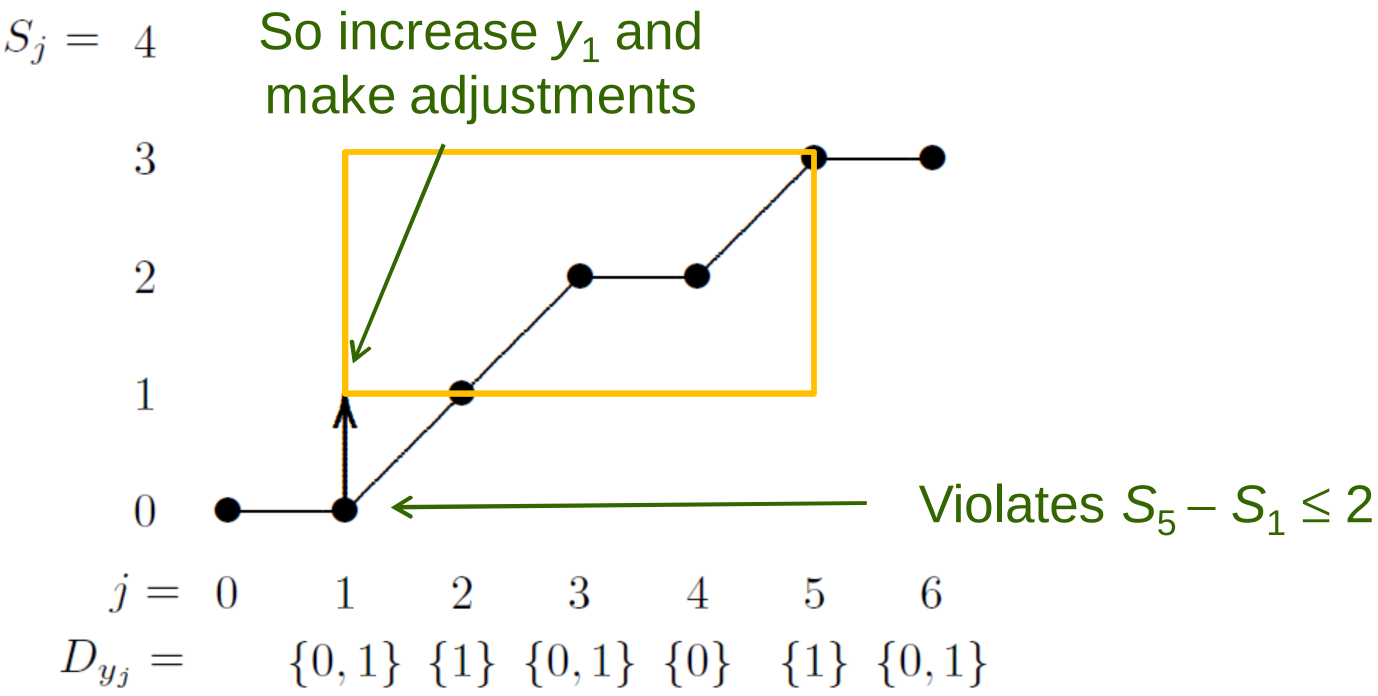


Violates $S_5 - S_1 \leq 2$

Filtering based on cumulative sums

- Example $\text{sequence}\big((y_1,\ldots,y_6),4,2,2\big)$

$y_1 \in \{0,1\} \quad y_2 \in \{1\} \quad y_3 \in \{0,1\} \quad y_4 \in \{0,1\} \quad y_5 \in \{1\} \quad y_6 \in \{0,1\}$


$S_j =$ 4
3
2
1
0
$j =$ 0 1 2 3 4 5 6
$D_{y_j} =$ $\{0,1\}$ $\{1\}$ $\{0,1\}$ $\{0\}$ $\{1\}$ $\{0,1\}$

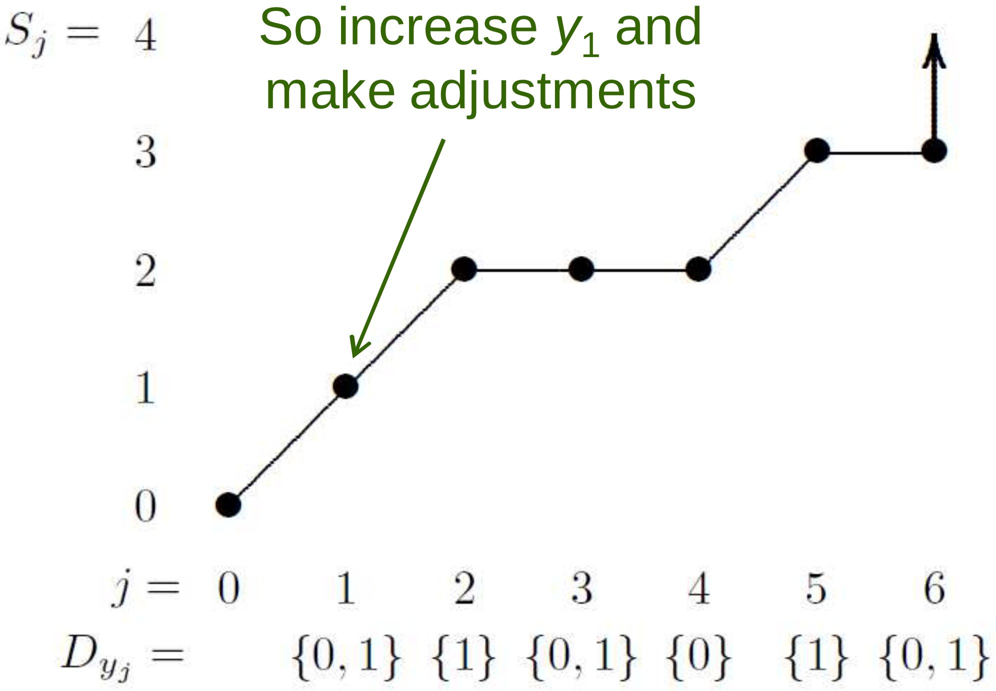
So increase y_1 and make adjustments

Violates $S_5 - S_1 \leq 2$

Filtering based on cumulative sums

- Example $\text{sequence}\big((y_1,\ldots,y_6),4,2,2\big)$

$y_1 \in \{0,1\} \quad y_2 \in \{1\} \quad y_3 \in \{0,1\} \quad y_4 \in \{0,1\} \quad y_5 \in \{1\} \quad y_6 \in \{0,1\}$

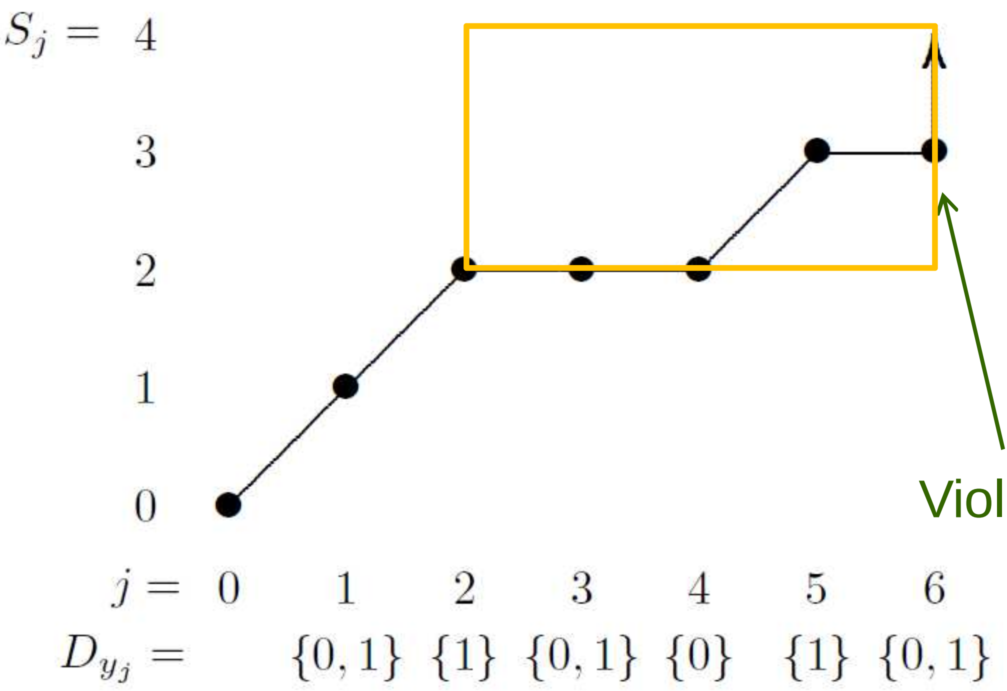


So increase y_1 and make adjustments

Filtering based on cumulative sums

- Example $\text{sequence}\left((y_1,\ldots,y_6),4,2,2\right)$

$y_1 \in \{0,1\} \quad y_2 \in \{1\} \quad y_3 \in \{0,1\} \quad y_4 \in \{0,1\} \quad y_5 \in \{1\} \quad y_6 \in \{0,1\}$

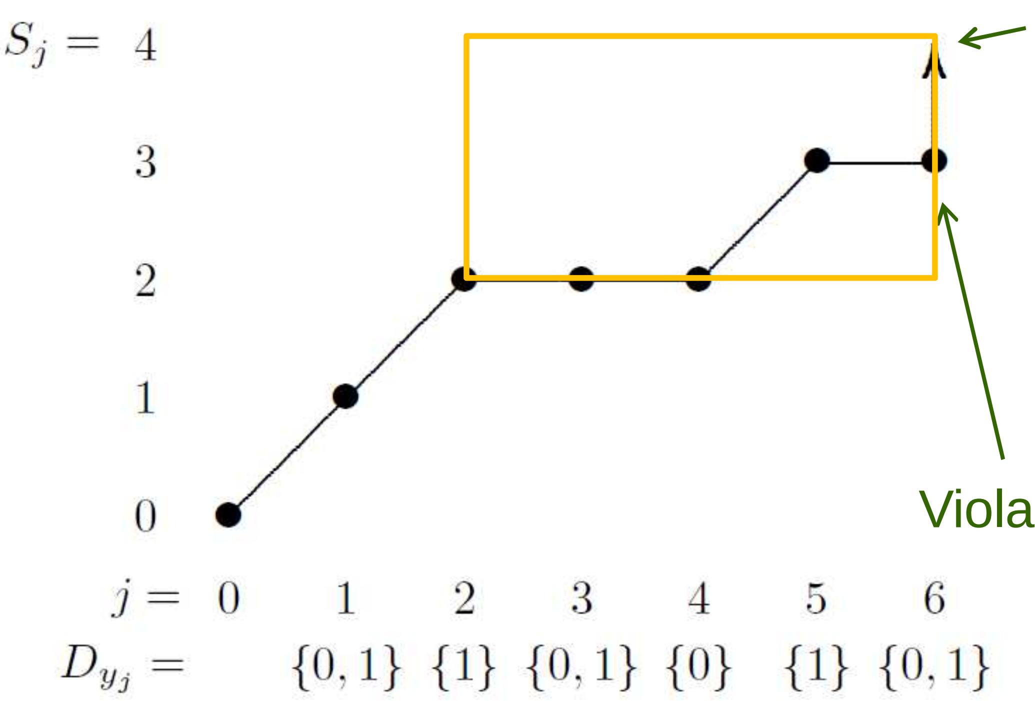


Violates $S_6 - S_2 \geq 2$

Filtering based on cumulative sums

- Example $\text{sequence}\left((y_1,\ldots,y_6),4,2,2\right)$

$y_1 \in \{0,1\} \quad y_2 \in \{1\} \quad y_3 \in \{0,1\} \quad y_4 \in \{0,1\} \quad y_5 \in \{1\} \quad y_6 \in \{0,1\}$



So increase y_6 and make adjustments

Violates $S_6 - S_2 \geq 2$

Filtering based on cumulative sums

- Example $\text{sequence}\left((y_1,\ldots,y_6),4,2,2\right)$

$$y_1 \in \{0,1\} \quad y_2 \in \{1\} \quad y_3 \in \{0,1\} \quad y_4 \in \{0\} \quad y_5 \in \{1\} \quad y_6 \in \{0,1\}$$

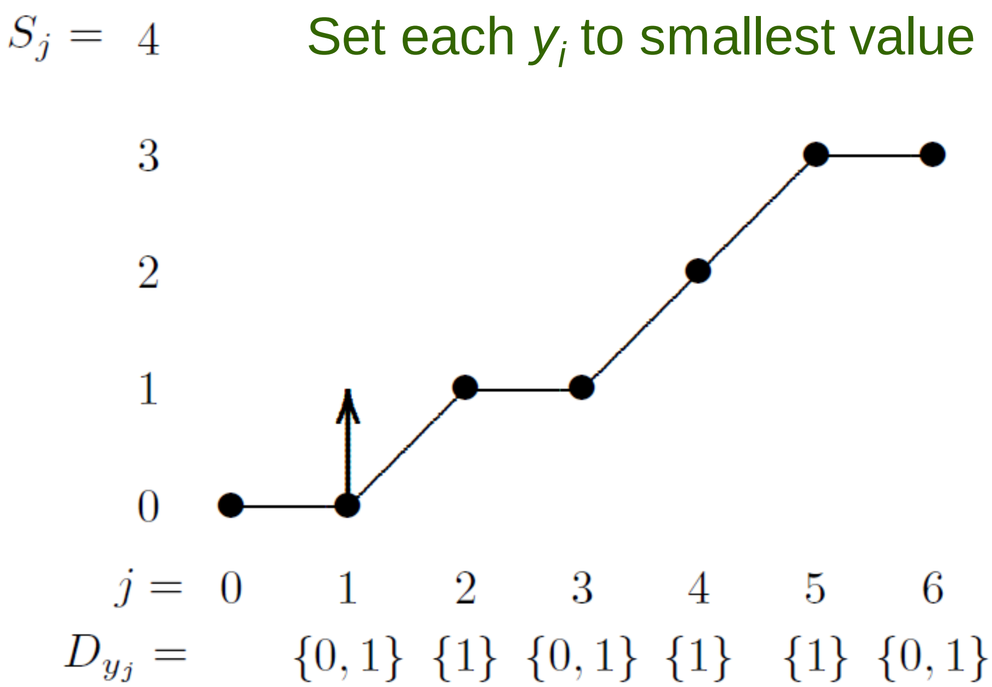
- Check whether 1 can be removed from domain of x_4.
  - Remove the 1 and check for feasibility.

Filtering based on cumulative sums

- Example $\text{sequence}\left((y_1,\ldots,y_6),4,2,2\right)$

$y_1 \in \{0,1\}$ $y_2 \in \{1\}$ $y_3 \in \{0,1\}$ $y_4 \in \{0\}$ $y_5 \in \{1\}$ $y_6 \in \{0,1\}$

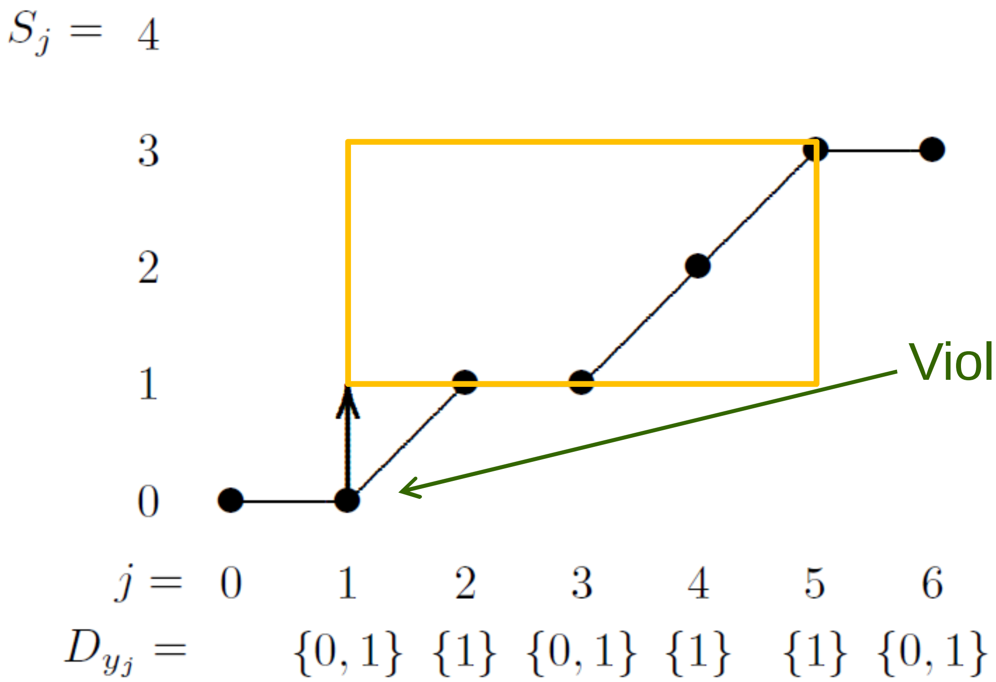
Set each y_i to smallest value in its domain.



Filtering based on cumulative sums

- Example $\text{sequence}\left((y_1,\ldots,y_6),4,2,2\right)$

$y_1 \in \{0,1\}$ $y_2 \in \{1\}$ $y_3 \in \{0,1\}$ $y_4 \in \{0\}$ $y_5 \in \{1\}$ $y_6 \in \{0,1\}$

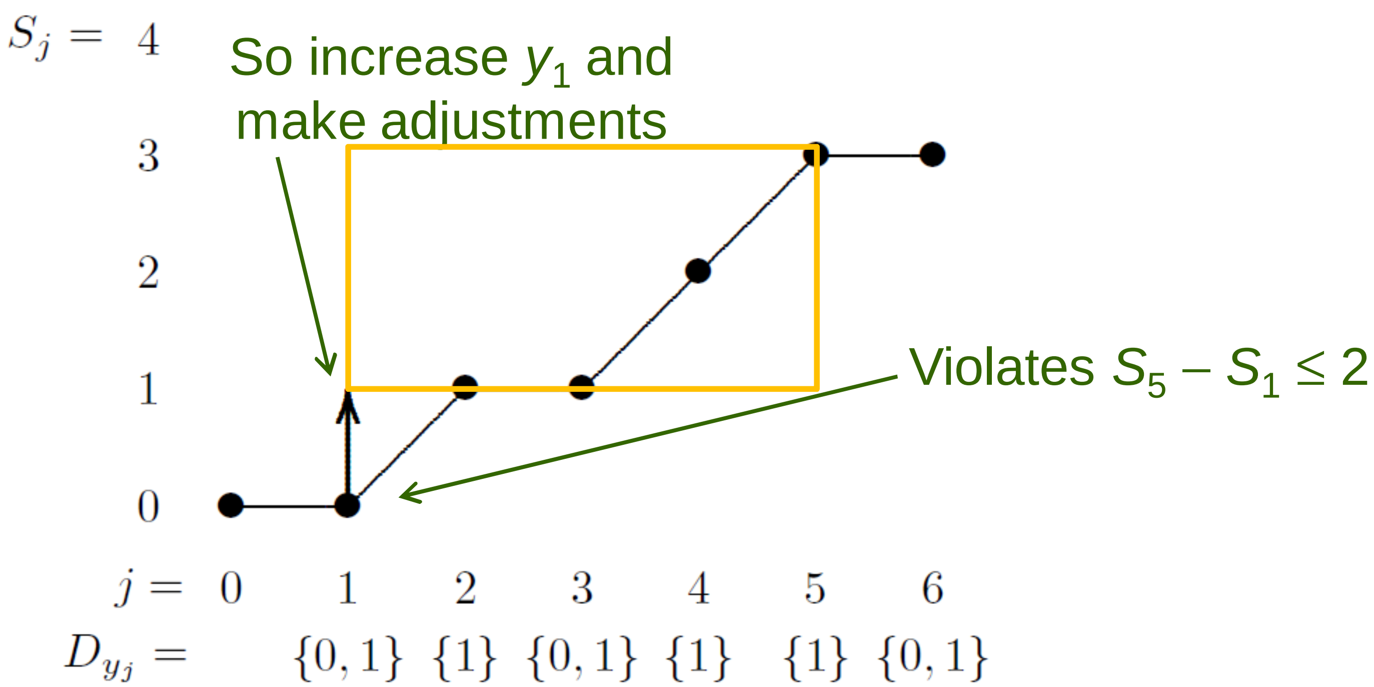


Violates $S_5 - S_1 \le 2$

Filtering based on cumulative sums

- Example $\text{sequence}\left((y_1,\ldots,y_6),4,2,2\right)$

$y_1 \in \{0,1\} \quad y_2 \in \{1\} \quad y_3 \in \{0,1\} \quad y_4 \in \{0\} \quad y_5 \in \{1\} \quad y_6 \in \{0,1\}$

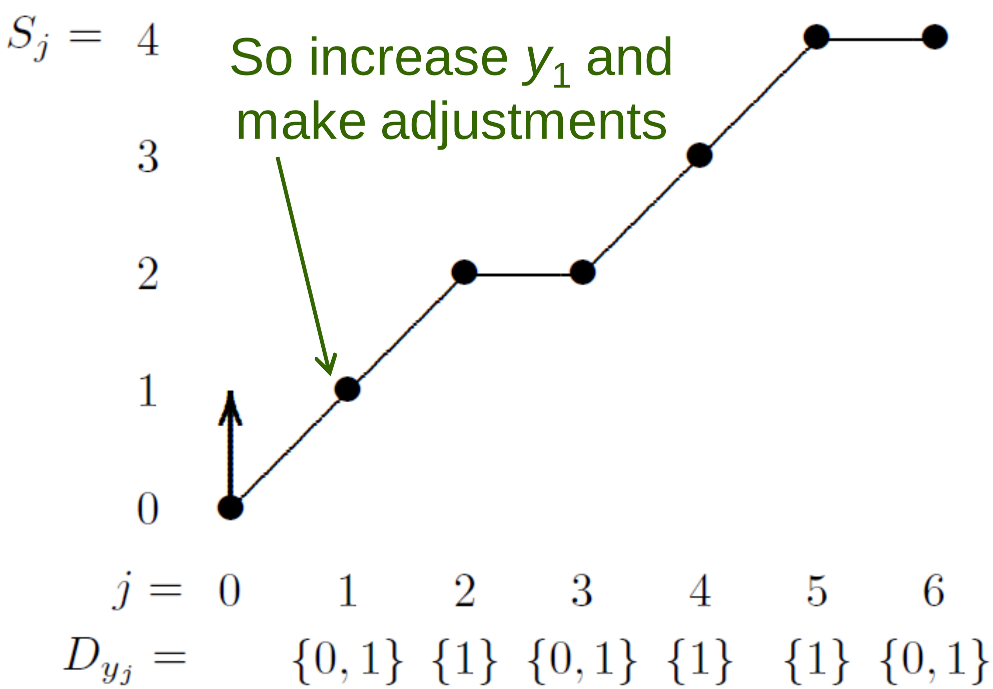


$S_j = 4$
3
2
1
0
So increase y_1 and make adjustments
Violates $S_5 - S_1 \leq 2$
$j =$ 0 1 2 3 4 5 6
$D_{y_j} =$ $\{0,1\}$ $\{1\}$ $\{0,1\}$ $\{1\}$ $\{1\}$ $\{0,1\}$

Filtering based on cumulative sums

- Example $\text{sequence}\big((y_1,\ldots,y_6),4,2,2\big)$

$y_1 \in \{0,1\}$ $y_2 \in \{1\}$ $y_3 \in \{0,1\}$ $y_4 \in \{0\}$ $y_5 \in \{1\}$ $y_6 \in \{0,1\}$

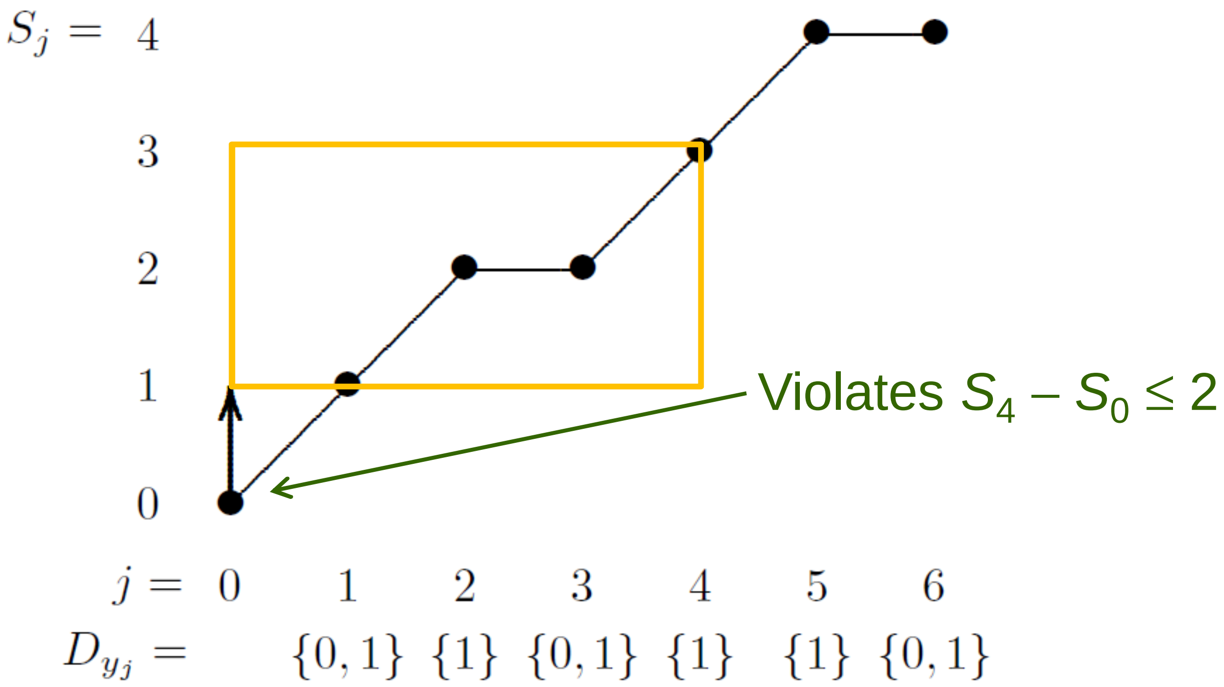


So increase y_1 and make adjustments

Filtering based on cumulative sums

- Example $\text{sequence}\big((y_1,\ldots,y_6),4,2,2\big)$

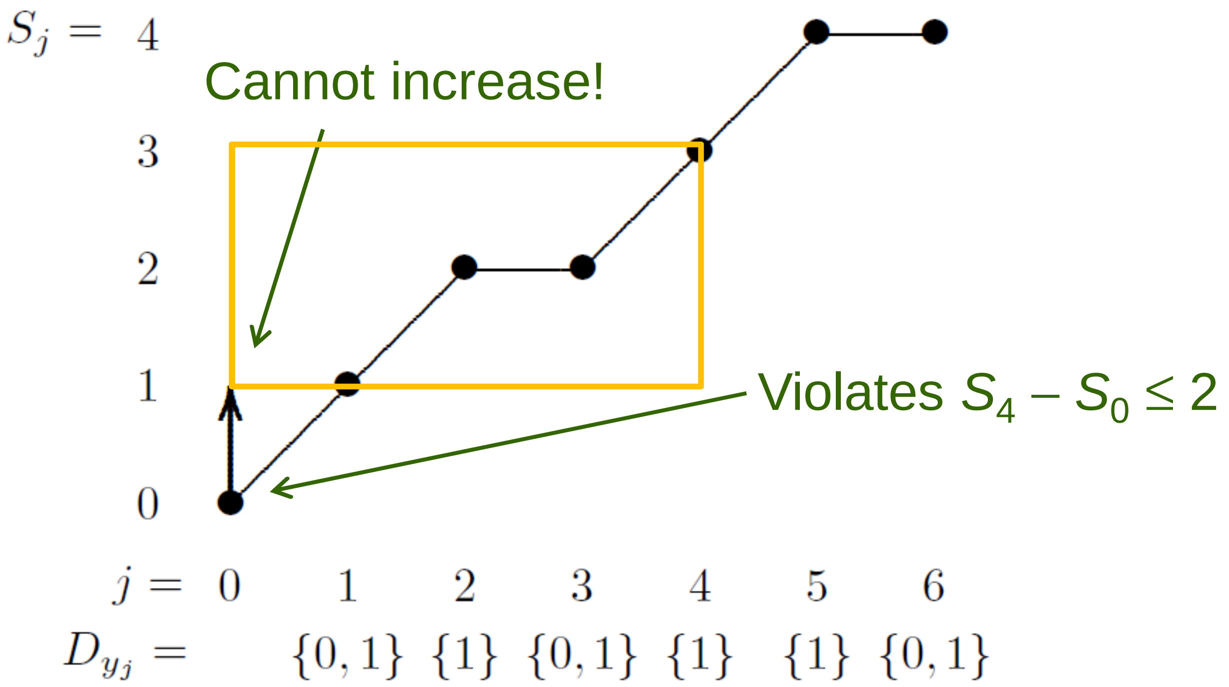
$y_1 \in \{0,1\} \quad y_2 \in \{1\} \quad y_3 \in \{0,1\} \quad y_4 \in \{0\} \quad y_5 \in \{1\} \quad y_6 \in \{0,1\}$



Filtering based on cumulative sums

- Example $\text{sequence}\left((y_1,\ldots,y_6),4,2,2\right)$

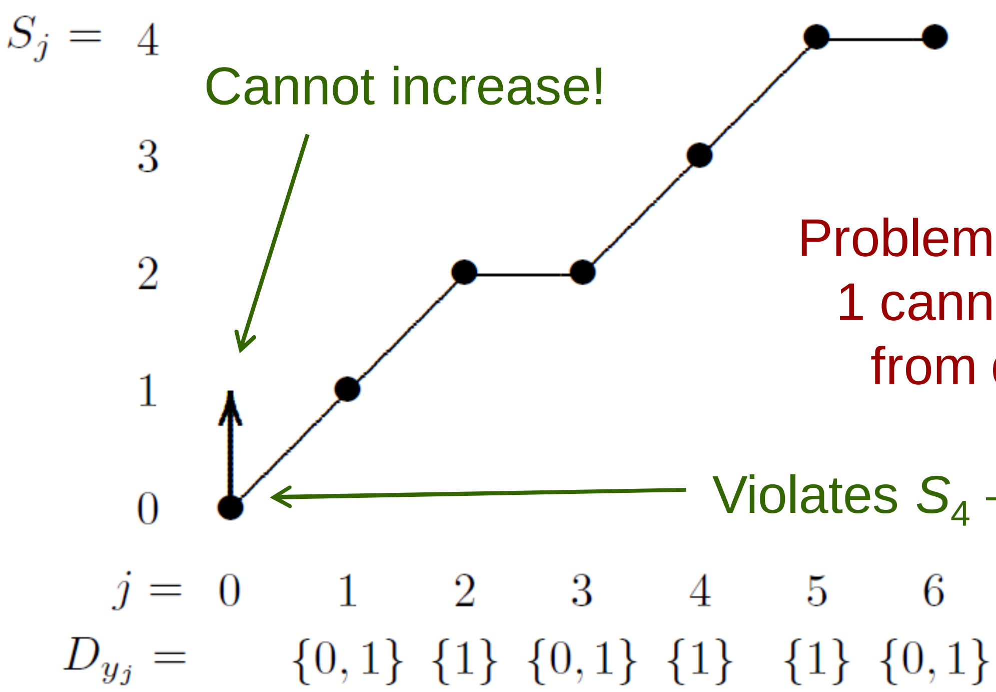
$y_1 \in \{0,1\}$ $y_2 \in \{1\}$ $y_3 \in \{0,1\}$ $y_4 \in \{0\}$ $y_5 \in \{1\}$ $y_6 \in \{0,1\}$


$S_j =$
4
3
2
1
0
Cannot increase!
Violates $S_4 - S_0 \leq 2$
$j =$ 0 1 2 3 4 5 6
$D_{y_j} =$ $\{0,1\}$ $\{1\}$ $\{0,1\}$ $\{1\}$ $\{1\}$ $\{0,1\}$

Filtering based on cumulative sums

- Example $\text{sequence}\left((y_1,\ldots,y_6),4,2,2\right)$

$y_1 \in \{0,1\} \quad y_2 \in \{1\} \quad y_3 \in \{0,1\} \quad y_4 \in \{0,1\} \quad y_5 \in \{1\} \quad y_6 \in \{0,1\}$



Cannot increase!

Problem is infeasible, so 1 cannot be removed from domain of y_4.

Violates $S_4 - S_0 \leq 2$

Filtering based on cumulative sums

- **Theorem.** This method correctly checks for feasibility and runs in $O(n^2)$ time.
    - So filtering requires $O(n^3)$ time (try removing each domain value).

Generalized sequence constraint

- The same method works for the **generalized sequence constraint**.

$$\text{genSequence}\big((X_1,\ldots,X_m),(\ell_1,\ldots,\ell_m),(u_1,\ldots,u_m)\big)$$

  - Each variable set X_i takes value 1 at least ℓ and at most u_i times, where $X = \{x_1,\ldots,x_n\} = X_1 \cup \cdots \cup X_m$.

  - Standard sequence constraint is

$$\text{genSequence}\big((X_1,\ldots,X_{n-q+1}),(\ell,\ldots,\ell),(u,\ldots,u)\big)$$

  where $X_i = \{x_i,\ldots,x_{i+q-1}\}$.

  - Filtering **genSequence** has same complexity as filtering **sequence**.

Filtering based on network flows

- **Sequence** can be formulated as an integer programming problem.
    - Transpose of constraint matrix has **consecutive 1s property**.
    - So feasibility can be checked in polytime.
    - In fact, there is a network flow model.

Filtering based on network flows

- Example. $\text{sequence}\left((y_1,\ldots,y_7),3,\ell,u\right)$
  - Integer programming formulation:

$$\ell \le y_{j-2} + y_{j-1} + y_j \le u$$

Filtering based on network flows

- Example. $\text{sequence}\big((y_1,\ldots,y_7),3,\ell,u\big)$
    - Integer programming formulation:
    - $\ell \le y_{j-2} + y_{j-1} + y_j \le u$
    - Matrix form:

$$\begin{bmatrix}
1 & 1 & 1 & & & & & -1 & & & & & & & & & \\
1 & 1 & 1 & & & & & & 1 & & & & & & & & \\
 & 1 & 1 & 1 & & & & & & -1 & & & & & & & \\
 & 1 & 1 & 1 & & & & & & & 1 & & & & & & \\
 & & 1 & 1 & 1 & & & & & & & -1 & & & & & \\
 & & 1 & 1 & 1 & & & & & & & & 1 & & & & \\
 & & & 1 & 1 & 1 & & & & & & & & -1 & & & \\
 & & & 1 & 1 & 1 & & & & & & & & & 1 & & \\
 & & & & 1 & 1 & 1 & & & & & & & & & -1 & \\
 & & & & 1 & 1 & 1 & & & & & & & & & & 1
\end{bmatrix}
\begin{bmatrix} y_1 \\ \vdots \\ y_7 \\ w_3 \\ z_3 \\ \vdots \\ w_7 \\ z_7 \end{bmatrix}
=
\begin{bmatrix} \ell \\ u \\ \ell \\ u \\ \ell \\ u \\ \ell \\ u \\ \ell \\ u \end{bmatrix}$$

Filtering based on network flows

- Example. $\text{sequence}\big((y_1,\ldots,y_7),3,\ell,u\big)$
  - Integer programming formulation:

$$\ell \le y_{j-2} + y_{j-1} + y_j \le u$$

  - Matrix form:

$$\begin{bmatrix}
1 & 1 & 1 & & & & & -1 & & & & & & & & & \\
1 & 1 & 1 & & & & & & 1 & & & & & & & & \\
 & 1 & 1 & 1 & & & & & & -1 & & & & & & & \\
 & 1 & 1 & 1 & & & & & & & 1 & & & & & & \\
 & & 1 & 1 & 1 & & & & & & & -1 & & & & & \\
 & & 1 & 1 & 1 & & & & & & & & 1 & & & & \\
 & & & 1 & 1 & 1 & & & & & & & & -1 & & & \\
 & & & 1 & 1 & 1 & & & & & & & & & 1 & & \\
 & & & & 1 & 1 & 1 & & & & & & & & & -1 & \\
 & & & & 1 & 1 & 1 & & & & & & & & & & 1
\end{bmatrix}
\begin{bmatrix} y_1 \\ \vdots \\ y_7 \\ \boxed{w_3} \\ z_3 \\ \vdots \\ \boxed{w_7} \\ z_7 \end{bmatrix}
=
\begin{bmatrix} \ell \\ u \\ \ell \\ u \\ \ell \\ u \\ \ell \\ u \\ \ell \\ u \end{bmatrix}$$

Surplus variables

Filtering based on network flows

- Example. $\text{sequence}\big((y_1,\ldots,y_7),3,\ell,u\big)$
  - Integer programming formulation:

$$\ell \le y_{j-2} + y_{j-1} + y_j \le u$$

  - Matrix form:

$$\begin{bmatrix}
1 & 1 & 1 & & & & & -1 & & & & & & & & & \\
1 & 1 & 1 & & & & & & 1 & & & & & & & & \\
 & 1 & 1 & 1 & & & & & & -1 & & & & & & & \\
 & 1 & 1 & 1 & & & & & & & 1 & & & & & & \\
 & & 1 & 1 & 1 & & & & & & & -1 & & & & & \\
 & & 1 & 1 & 1 & & & & & & & & 1 & & & & \\
 & & & 1 & 1 & 1 & & & & & & & & -1 & & & \\
 & & & 1 & 1 & 1 & & & & & & & & & 1 & & \\
 & & & & 1 & 1 & 1 & & & & & & & & & -1 & \\
 & & & & 1 & 1 & 1 & & & & & & & & & & 1
\end{bmatrix}
\begin{bmatrix} y_1 \\ \vdots \\ y_7 \\ w_3 \\ \boxed{z_3} \\ \vdots \\ w_7 \\ \boxed{z_7} \end{bmatrix}
=
\begin{bmatrix} \ell \\ u \\ \ell \\ u \\ \ell \\ u \\ \ell \\ u \\ \ell \\ u \end{bmatrix}$$

Slack variables

Filtering based on network flows

- Example. $\text{sequence}\big((y_1,\ldots,y_7),3,\ell,u\big)$
  - Integer programming formulation:

$$\ell \le y_{j-2} + y_{j-1} + y_j \le u$$

  - Matrix form:

$$\begin{bmatrix}
1&1&1&&&&&-1&&&&&&&&& \\
1&1&1&&&&&&1&&&&&&&& \\
&1&1&1&&&&&&-1&&&&&&& \\
&1&1&1&&&&&&&1&&&&&& \\
&&1&1&1&&&&&&&-1&&&&& \\
&&1&1&1&&&&&&&&1&&&& \\
&&&1&1&1&&&&&&&&-1&&& \\
&&&1&1&1&&&&&&&&&1&& \\
&&&&1&1&1&&&&&&&&&-1& \\
&&&&1&1&1&&&&&&&&&&1
\end{bmatrix}
\begin{bmatrix} y_1 \\ \vdots \\ y_7 \\ w_3 \\ z_3 \\ \vdots \\ w_7 \\ z_7 \end{bmatrix}
=
\begin{bmatrix} \ell \\ u \\ \ell \\ u \\ \ell \\ u \\ \ell \\ u \\ \ell \\ u \end{bmatrix}$$

Transpose of matrix has consecutive 1s property.

Filtering based on network flows

- Row operations convert it to network flow matrix.

Subtract each row from the next (after adding row of 0s to the bottom)

$$\begin{bmatrix}
1&1&1&&&&&-1&&&&&&&&&\\
1&1&1&&&&&&1&&&&&&&&\\
&1&1&1&&&&&&-1&&&&&&&\\
&1&1&1&&&&&&&1&&&&&&\\
&&1&1&1&&&&&&&-1&&&&&\\
&&1&1&1&&&&&&&&1&&&&\\
&&&1&1&1&&&&&&&&-1&&&\\
&&&1&1&1&&&&&&&&&1&&\\
&&&&1&1&1&&&&&&&&&-1&\\
&&&&1&1&1&&&&&&&&&&1
\end{bmatrix}
\begin{bmatrix} y_1\\ \vdots\\ y_7\\ w_3\\ z_3\\ \vdots\\ w_7\\ z_7 \end{bmatrix}
=
\begin{bmatrix} \ell\\ u\\ \ell\\ u\\ \ell\\ u\\ \ell\\ u\\ \ell\\ u \end{bmatrix}$$

$$\begin{bmatrix}
1&1&1&&&&&-1&&&&&&&&&\\
&&&&&&&1&1&&&&&&&&\\
-1&&&1&&&&&-1&-1&&&&&&&\\
&&&&&&&&&1&1&&&&&&\\
&-1&&&1&&&&&&-1&-1&&&&&\\
&&&&&&&&&&&1&1&&&&\\
&&-1&&&1&&&&&&&-1&-1&&&\\
&&&&&&&&&&&&&1&1&&\\
&&&-1&&&1&&&&&&&&-1&-1&\\
&&&&&&&&&&&&&&&1&1\\
&&&&-1&-1&-1&&&&&&&&&&-1
\end{bmatrix}
\begin{bmatrix} y_1\\ \vdots\\ y_7\\ w_3\\ z_3\\ \vdots\\ w_7\\ z_7 \end{bmatrix}
=
\begin{bmatrix} \ell\\ u-\ell\\ \ell-u\\ u-\ell\\ \ell-u\\ u-\ell\\ \ell-u\\ u-\ell\\ \ell-u\\ u-\ell\\ -u \end{bmatrix}
\begin{matrix} (b_3)\\ (a_3)\\ (b_4)\\ (a_4)\\ (b_5)\\ (a_5)\\ (b_6)\\ (a_6)\\ (b_7)\\ (a_7)\\ (b_8) \end{matrix}$$

Filtering based on network flows

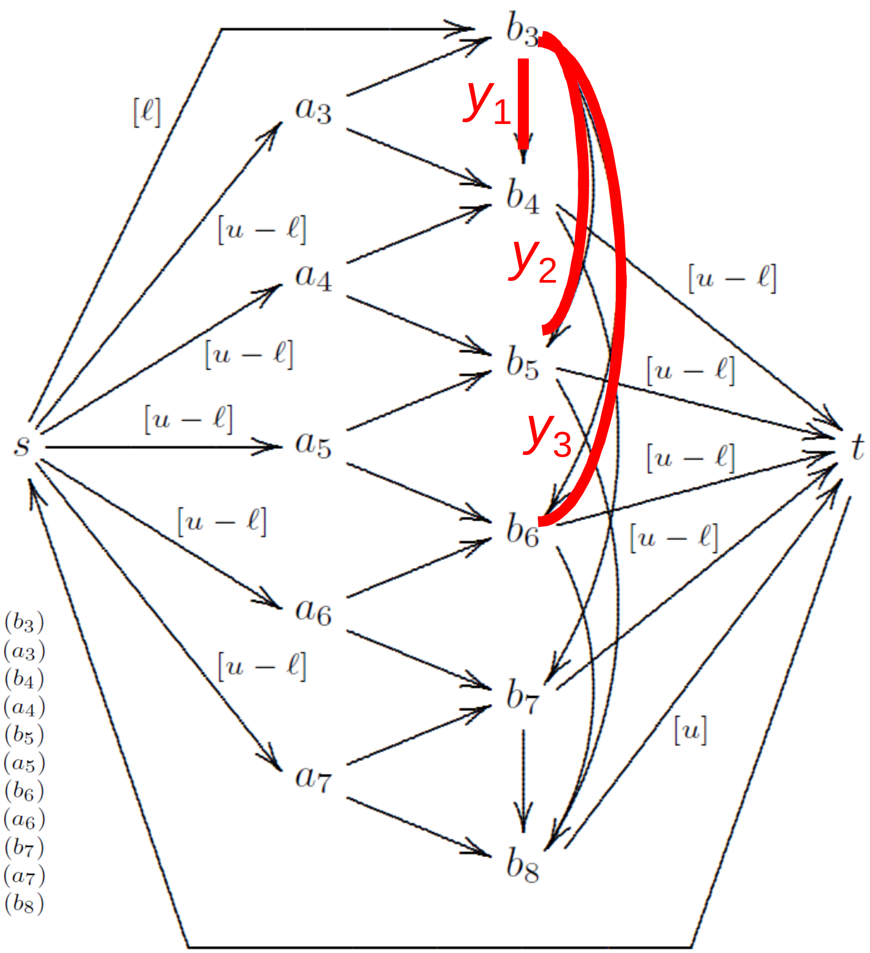
- Corresponding network flow problem.

Flow on labeled edges is fixed to label.

y_{j-q} = flow on arc (b_q, b_j) for $j = q+1, \ldots, 2q$

y_j = flow on arc (b_j, b_{j+q}) for $j = q+1, \ldots, n-q$

y_j = flow on arc (b_j, b_{n+1}) for $j = n-q+1, \ldots, n$



$$\begin{bmatrix}
1 & 1 & 1 & & & & & -1 & & & & & & & & & \\
& & & & & & & 1 & 1 & & & & & & & & \\
-1 & & & 1 & & & & & -1 & -1 & & & & & & & \\
& & & & & & & & & 1 & 1 & & & & & & \\
& -1 & & & 1 & & & & & & -1 & -1 & & & & & \\
& & & & & & & & & & & 1 & 1 & & & & \\
& & -1 & & & 1 & & & & & & & -1 & -1 & & & \\
& & & & & & & & & & & & & 1 & 1 & & \\
& & & -1 & & & 1 & & & & & & & & -1 & -1 & \\
& & & & & & & & & & & & & & & 1 & 1 \\
& & & & -1 & -1 & -1 & & & & & & & & & & -1
\end{bmatrix}
\begin{bmatrix} y_1 \\ \vdots \\ y_7 \\ w_3 \\ z_3 \\ \vdots \\ w_7 \\ z_7 \end{bmatrix}
=
\begin{bmatrix} \ell \\ u-\ell \\ \ell-u \\ u-\ell \\ \ell-u \\ u-\ell \\ \ell-u \\ u-\ell \\ \ell-u \\ u-\ell \\ -u \end{bmatrix}
\begin{matrix} (b_3) \\ (a_3) \\ (b_4) \\ (a_4) \\ (b_5) \\ (a_5) \\ (b_6) \\ (a_6) \\ (b_7) \\ (a_7) \\ (b_8) \end{matrix}$$

Filtering based on network flows

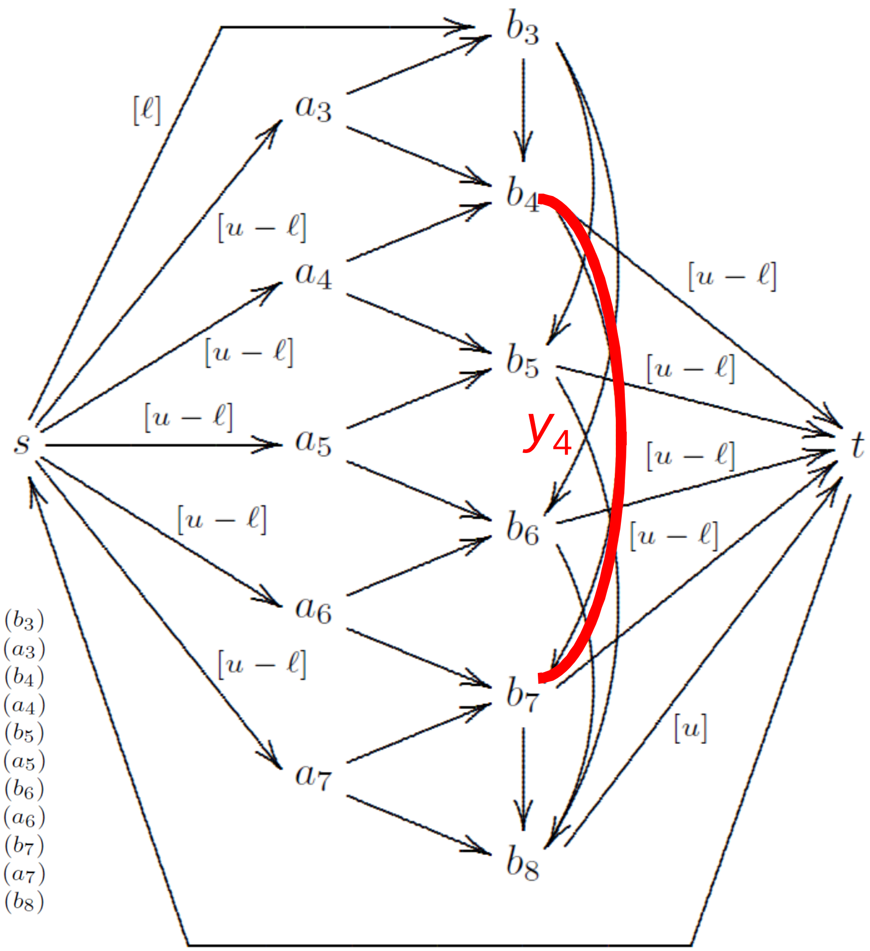
- Corresponding network flow problem.

Flow on labeled edges is fixed to label.

y_{j-q} = flow on arc (b_q,b_j) for $j = q+1,\ldots,2q$

y_j = flow on arc (b_j,b_{j+q}) for $j = q+1,\ldots,n-q$

y_j = flow on arc (b_j,b_{n+1}) for $j = n-q+1,\ldots,n$



$$\begin{bmatrix}
1 & 1 & 1 & & & & & -1 & & & & & & & & & \\
 & & & & & & & 1 & 1 & & & & & & & & \\
-1 & & & 1 & & & & & -1 & -1 & & & & & & & \\
 & & & & & & & & & 1 & 1 & & & & & & \\
 & -1 & & & 1 & & & & & & -1 & -1 & & & & & \\
 & & & & & & & & & & & 1 & 1 & & & & \\
 & & -1 & & & 1 & & & & & & & -1 & -1 & & & \\
 & & & & & & & & & & & & & 1 & 1 & & \\
 & & & -1 & & & 1 & & & & & & & & -1 & -1 & \\
 & & & & & & & & & & & & & & & 1 & 1 \\
 & & & & -1 & -1 & -1 & & & & & & & & & & -1
\end{bmatrix}
\begin{bmatrix} y_1 \\ \vdots \\ y_7 \\ w_3 \\ z_3 \\ \vdots \\ w_7 \\ z_7 \end{bmatrix}
=
\begin{array}{c}
\begin{bmatrix} \ell \\ u-\ell \\ \ell-u \\ u-\ell \\ \ell-u \\ u-\ell \\ \ell-u \\ u-\ell \\ \ell-u \\ u-\ell \\ -u \end{bmatrix}
\end{array}
\begin{array}{c} (b_3) \\ (a_3) \\ (b_4) \\ (a_4) \\ (b_5) \\ (a_5) \\ (b_6) \\ (a_6) \\ (b_7) \\ (a_7) \\ (b_8) \end{array}$$

Filtering based on network flows

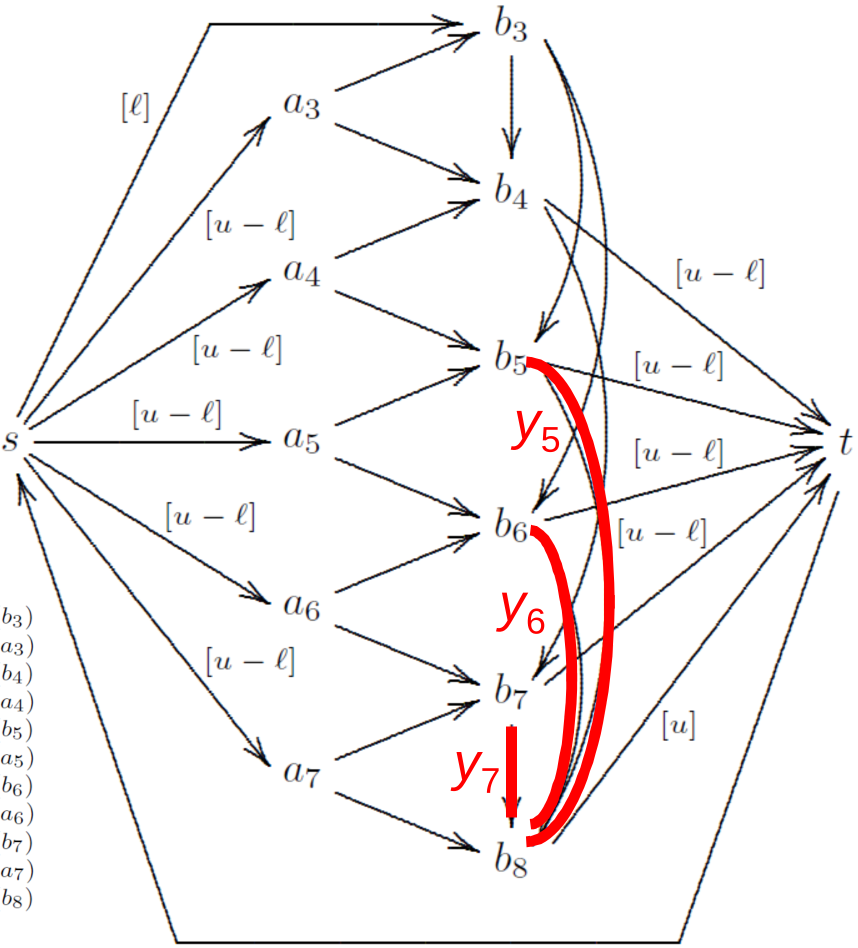
- Corresponding network flow problem.

Flow on labeled edges is fixed to label.

y_{j-q} = flow on arc (b_q,b_j) for $j = q+1,\ldots,2q$

y_j = flow on arc (b_j,b_{j+q}) for $j = q+1,\ldots,n-q$

y_j = flow on arc (b_j,b_{n+1}) for $j = n-q+1,\ldots,n$



$$
\begin{bmatrix}
1 & 1 & 1 & & & & & -1 & & & & & & & & & \\
 & & & & & & & 1 & 1 & & & & & & & & \\
-1 & & & 1 & & & & & -1 & -1 & & & & & & & \\
 & & & & & & & & & 1 & 1 & & & & & & \\
 & -1 & & & 1 & & & & & & -1 & -1 & & & & & \\
 & & & & & & & & & & & 1 & 1 & & & & \\
 & & -1 & & & 1 & & & & & & & -1 & -1 & & & \\
 & & & & & & & & & & & & & 1 & 1 & & \\
 & & & -1 & & & 1 & & & & & & & & -1 & -1 & \\
 & & & & & & & & & & & & & & & 1 & 1 \\
 & & & & -1 & -1 & -1 & & & & & & & & & & -1
\end{bmatrix}
\begin{bmatrix} y_1 \\ \vdots \\ y_7 \\ w_3 \\ z_3 \\ \vdots \\ w_7 \\ z_7 \end{bmatrix}
=
\begin{bmatrix} \ell \\ u-\ell \\ \ell-u \\ u-\ell \\ \ell-u \\ u-\ell \\ \ell-u \\ u-\ell \\ \ell-u \\ u-\ell \\ -u \end{bmatrix}
\begin{matrix} (b_3) \\ (a_3) \\ (b_4) \\ (a_4) \\ (b_5) \\ (a_5) \\ (b_6) \\ (a_6) \\ (b_7) \\ (a_7) \\ (b_8) \end{matrix}
$$

Filtering based on network flows

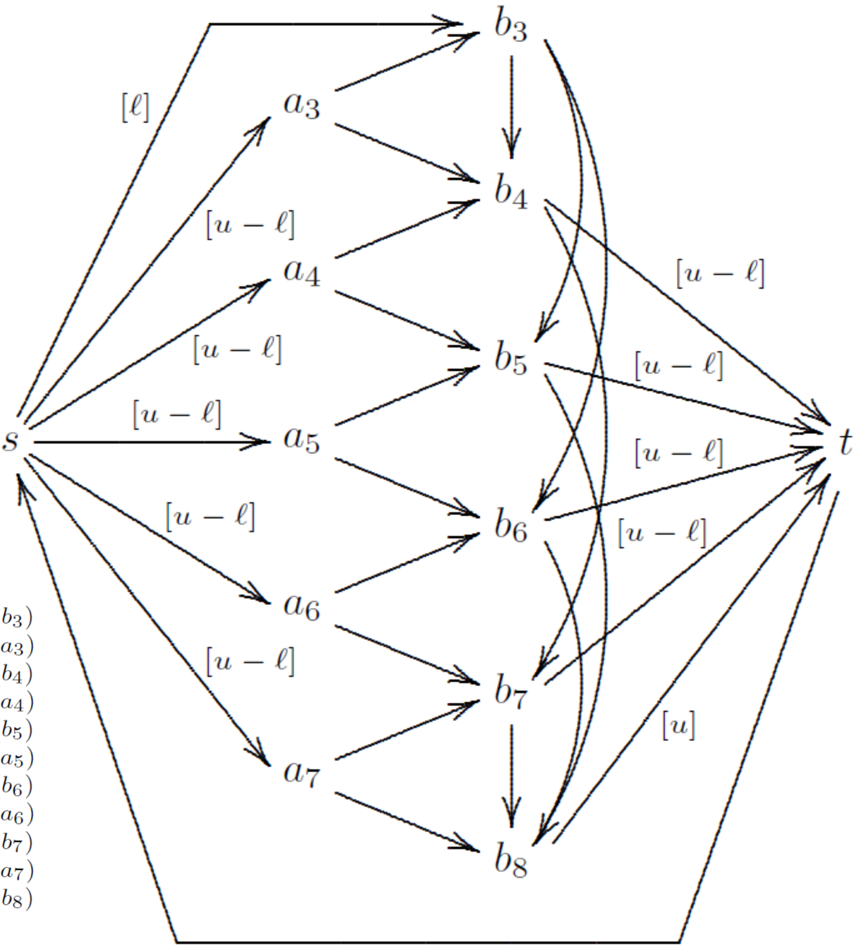
- Can now filter using optimality conditions for max flow

Flow on labeled edges is fixed to label.

y_{j-q} = flow on arc (b_q, b_j) for $j = q+1,\ldots,2q$

y_j = flow on arc (b_j, b_{j+q}) for $j = q+1,\ldots,n-q$

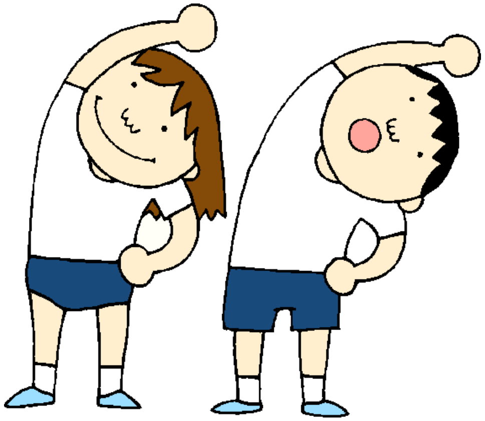
y_j = flow on arc (b_j, b_{n+1}) for $j = n-q+1,\ldots,n$



$$\begin{bmatrix}
1 & 1 & 1 & & & & & -1 & & & & & & & & & \\
 & & & & & & & 1 & 1 & & & & & & & & \\
-1 & & & 1 & & & & & -1 & -1 & & & & & & & \\
 & & & & & & & & & 1 & 1 & & & & & & \\
 & -1 & & & 1 & & & & & & -1 & -1 & & & & & \\
 & & & & & & & & & & & 1 & 1 & & & & \\
 & & -1 & & & 1 & & & & & & & -1 & -1 & & & \\
 & & & & & & & & & & & & & 1 & 1 & & \\
 & & & -1 & & & 1 & & & & & & & & -1 & -1 & \\
 & & & & & & & & & & & & & & & 1 & 1 \\
 & & & & -1 & -1 & -1 & & & & & & & & & & -1
\end{bmatrix}
\begin{bmatrix} y_1 \\ \vdots \\ y_7 \\ w_3 \\ z_3 \\ \vdots \\ w_7 \\ z_7 \end{bmatrix}
=
\begin{bmatrix} \ell \\ u-\ell \\ \ell-u \\ u-\ell \\ \ell-u \\ u-\ell \\ \ell-u \\ u-\ell \\ \ell-u \\ u-\ell \\ -u \end{bmatrix}
\begin{array}{c} (b_3) \\ (a_3) \\ (b_4) \\ (a_4) \\ (b_5) \\ (a_5) \\ (b_6) \\ (a_6) \\ (b_7) \\ (a_7) \\ (b_8) \end{array}$$

Generalized sequence constraint

- The genSequence constraint may not have a network flow model.
  - Can check in $O(m + n + r)$ time whether rows can be permuted to yield a matrix whose transpose has the consecutive 1s property, in which case there is a network flow model.
    - m x n = size of matrix, r = number of nonzeros in matrix.
  - If not, can still check in $O(mr)$ time if there is an equivalent network matrix.
  - If not, can still check feasibility by linear programming.
    - y_i portion of matrix has consecutive 1s property, and remaining columns are ±unit vectors.
    - So problem is totally unimodular, and LP has integral solution.



Stretch Constraint

Filtering Based on Dynamic Programming

Stretch constraint

- The **stretch constraint** controls the length of stretches (consecutive subsequences) of variables that take the same value.
  - It also includes a **pattern constraint**, which restricts value changes from one variable to the next.
- Can be filtered using **dynamic programming**.

Stretch constraint

- Example $\text{stretch}\big((x_1,\ldots,x_7),(a,b,c),(2,2,2),(3,3,3),P\big)$

  $P=\{(a,b),(b,a),(b,c),(c,b)\}$

  - x_i = shift worked on day *i*.
  - Stretch of shift *a* must contain 2 or 3 *a*'s, similarly for shift *b* and *c*.
  - Can transition only between shifts *a* & *b*, or *b* & *c*.

- Domains:

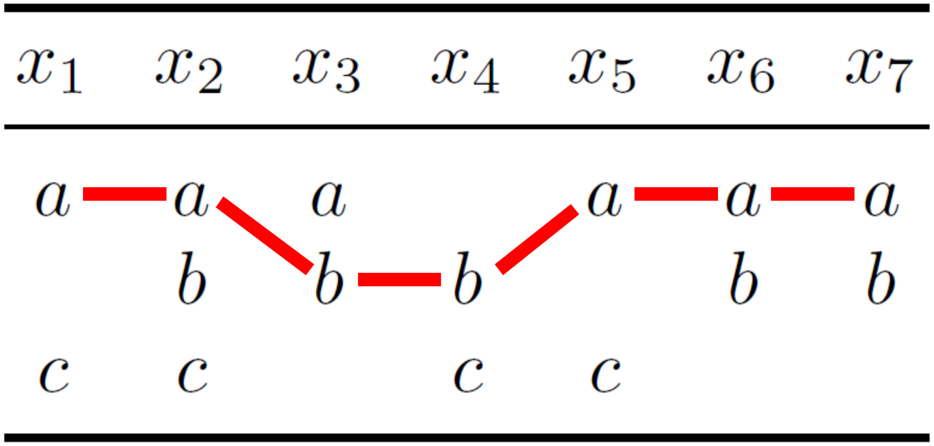
x_1	x_2	x_3	x_4	x_5	x_6	x_7
a	a	a		a	a	a
	b	b	b		b	b
c	c		c	c		

Stretch constraint

- Example $\text{stretch}\big((x_1,\ldots,x_7),(a,b,c),(2,2,2),(3,3,3),P\big)$

  $P=\{(a,b),(b,a),(b,c),(c,b)\}$

- There are 2 solutions.
  - Solution 1:

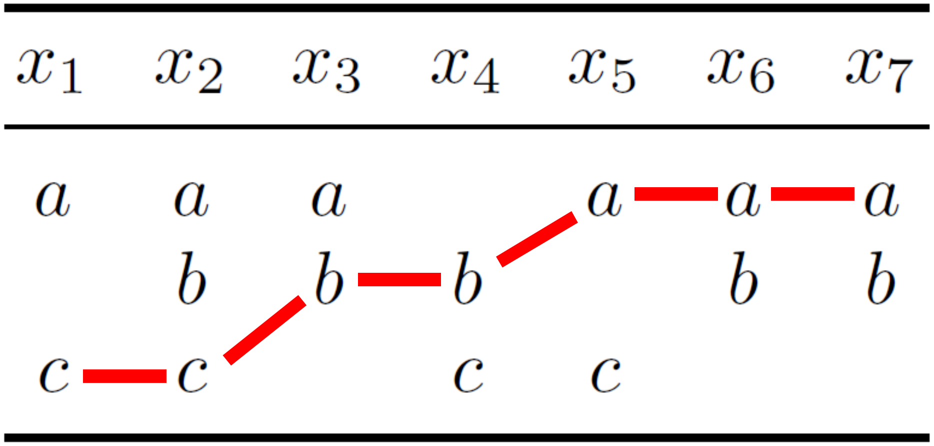


x_1	x_2	x_3	x_4	x_5	x_6	x_7
a	a	a		a	a	a
	b	b	b		b	b
c	c		c	c		

Stretch constraint

- Example $$\text{stretch}\left((x_1,\ldots,x_7),(a,b,c),(2,2,2),(3,3,3),P\right)$$ $$P=\{(a,b),(b,a),(b,c),(c,b)\}$$

- There are 2 solutions.
  - Solution 2:



x_1	x_2	x_3	x_4	x_5	x_6	x_7
a	a	a		a	a	a
	b	b	b		b	b
c	c		c	c		

Stretch constraint

- In general,

$$\text{stretch}(x,v,\ell,u,P)$$
$$P=\{(v_j,v_k)\,|\,(j,k)\in E\}$$

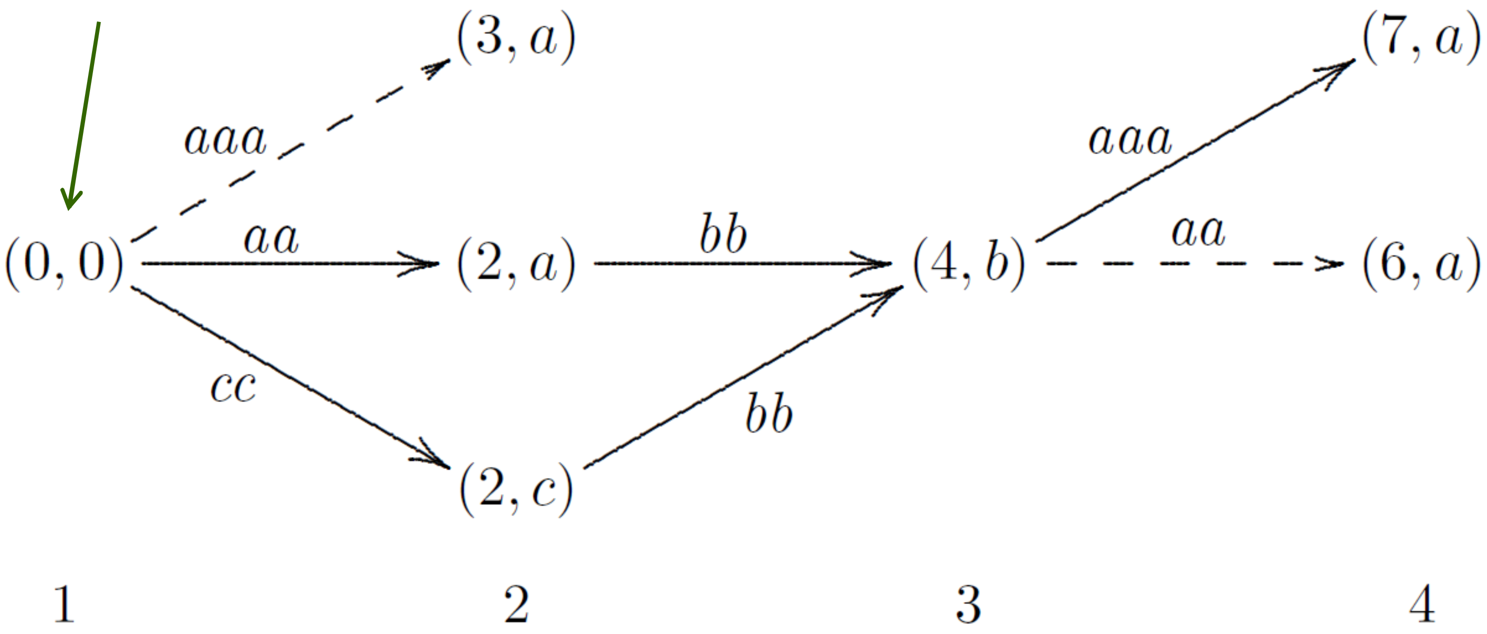
  - where $x = (x_1,\ldots,x_n)$, $v = (v_1,\ldots,v_m)$, $\ell = (\ell_1,\ldots, \ell_m)$, $u = (u_1,\ldots,u_m)$.
  - Requires that for $i = 1,\ldots,m$, any stretch of value v_i has length in the interval $[\ell_i,u_i]$.
    - A **stretch** is a maximal sequence of consecutive variables x_i that take the same value.
  - Requires that $(x_i,x_{i+1}) \in P$, for all i.

Filter based on dynamic programming

- Example $\text{stretch}\left((x_1,\ldots,x_7),(a,b,c),(2,2,2),(3,3,3),P\right)$

  $P=\{(a,b),(b,a),(b,c),(c,b)\}$

State = (day,shift)



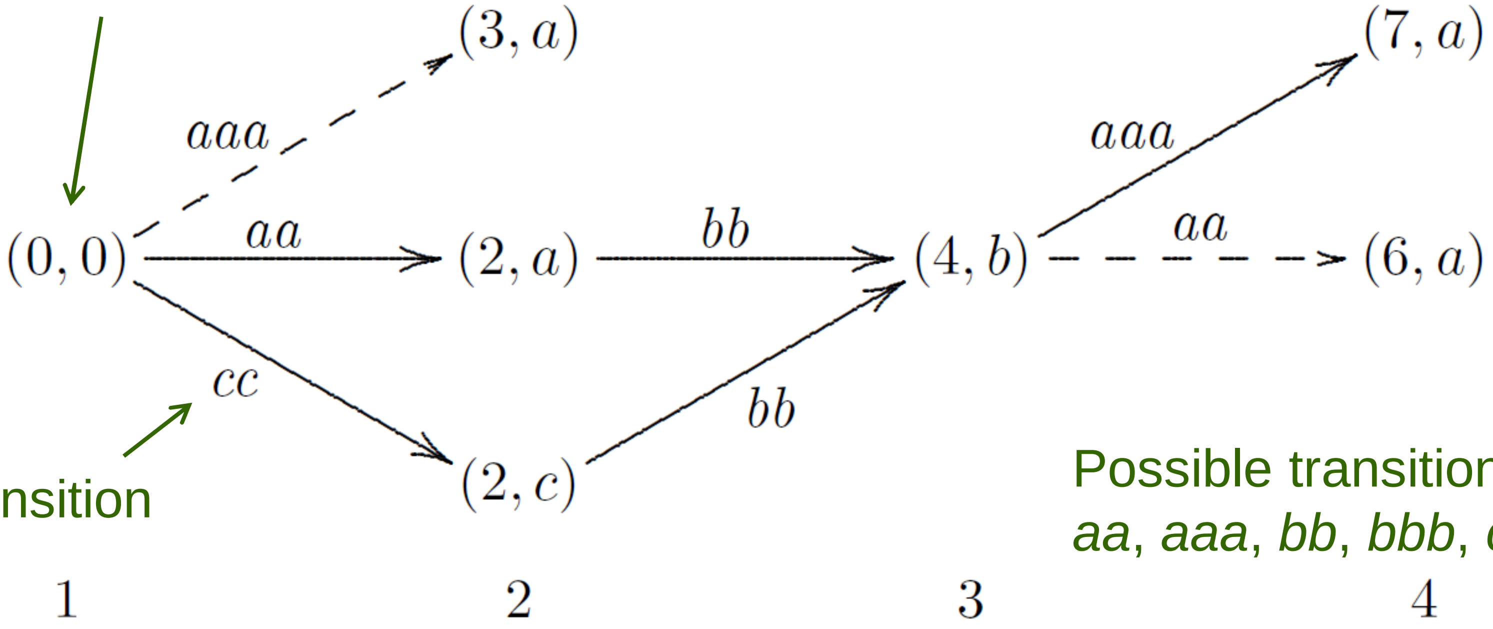
Filter based on dynamic programming

- Example

$$\text{stretch}\left((x_1,\ldots,x_7),(a,b,c),(2,2,2),(3,3,3),P\right)$$

$$P=\{(a,b),(b,a),(b,c),(c,b)\}$$

State = (day,shift)



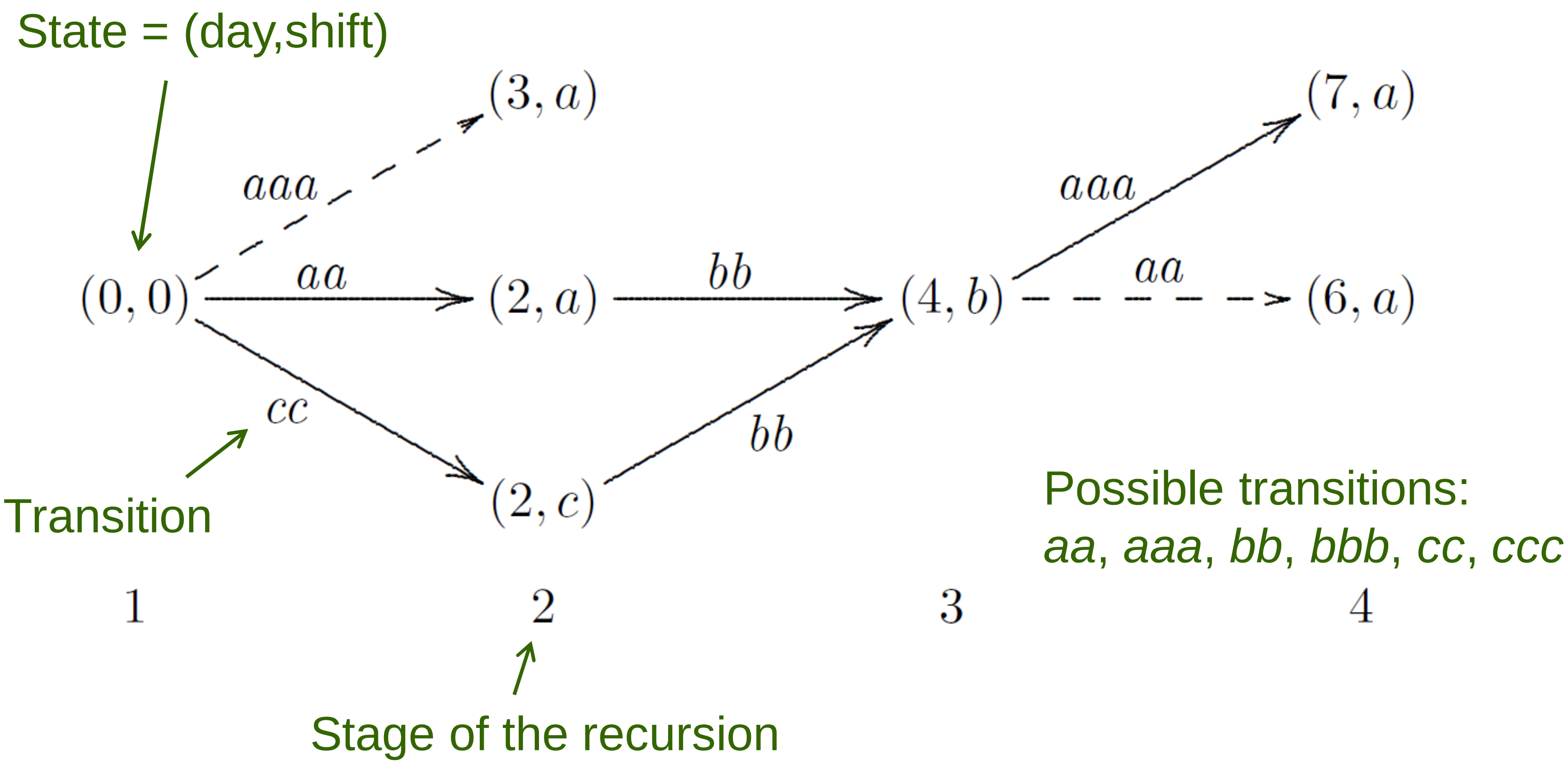
Transition

Possible transitions:
aa, *aaa*, *bb*, *bbb*, *cc*, *ccc*

Filter based on dynamic programming

- Example $\text{stretch}\big((x_1,\ldots,x_7),(a,b,c),(2,2,2),(3,3,3),P\big)$

$$P=\{(a,b),(b,a),(b,c),(c,b)\}$$



State = (day,shift)

Transition

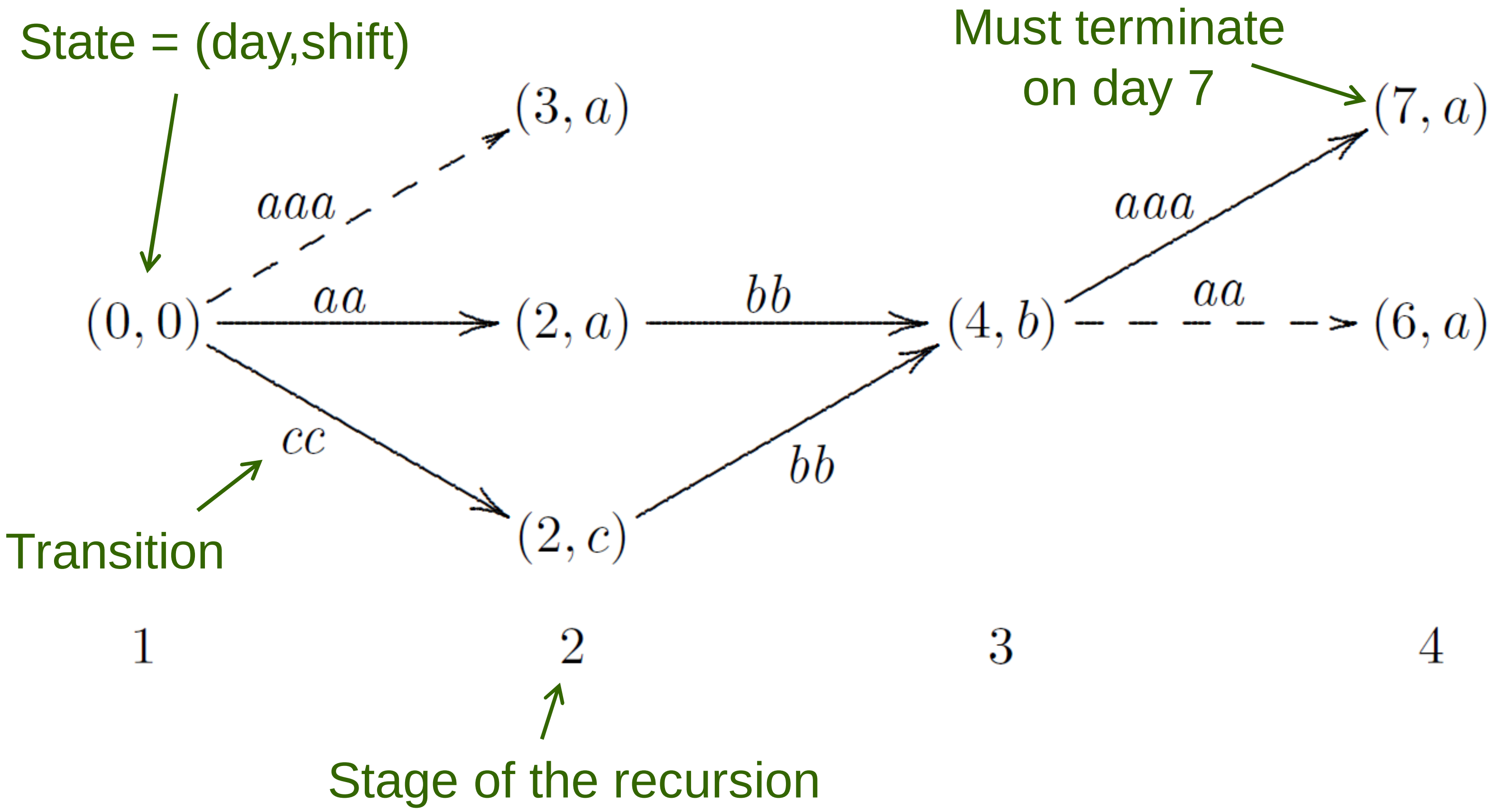
Possible transitions:
aa, *aaa*, *bb*, *bbb*, *cc*, *ccc*

Stage of the recursion

Filter based on dynamic programming

- Example $\text{stretch}\left((x_1,\ldots,x_7),(a,b,c),(2,2,2),(3,3,3),P\right)$

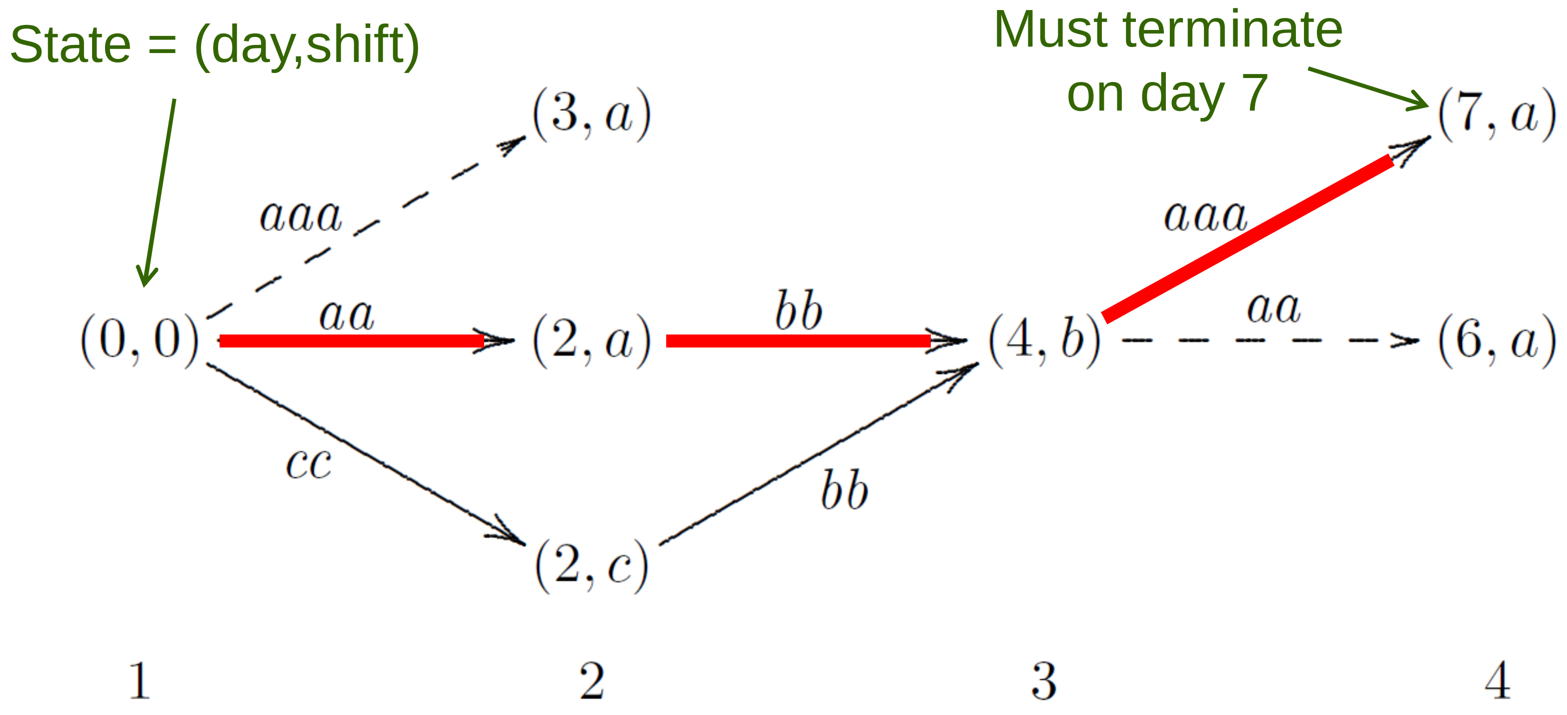
  $P=\{(a,b),(b,a),(b,c),(c,b)\}$



Filter based on dynamic programming

- Example $\text{stretch}\left((x_1,\ldots,x_7),(a,b,c),(2,2,2),(3,3,3),P\right)$

$$P = \left\{(a,b),(b,a),(b,c),(c,b)\right\}$$



State = (day,shift)

Must terminate on day 7

Solution: *aabbaaa*

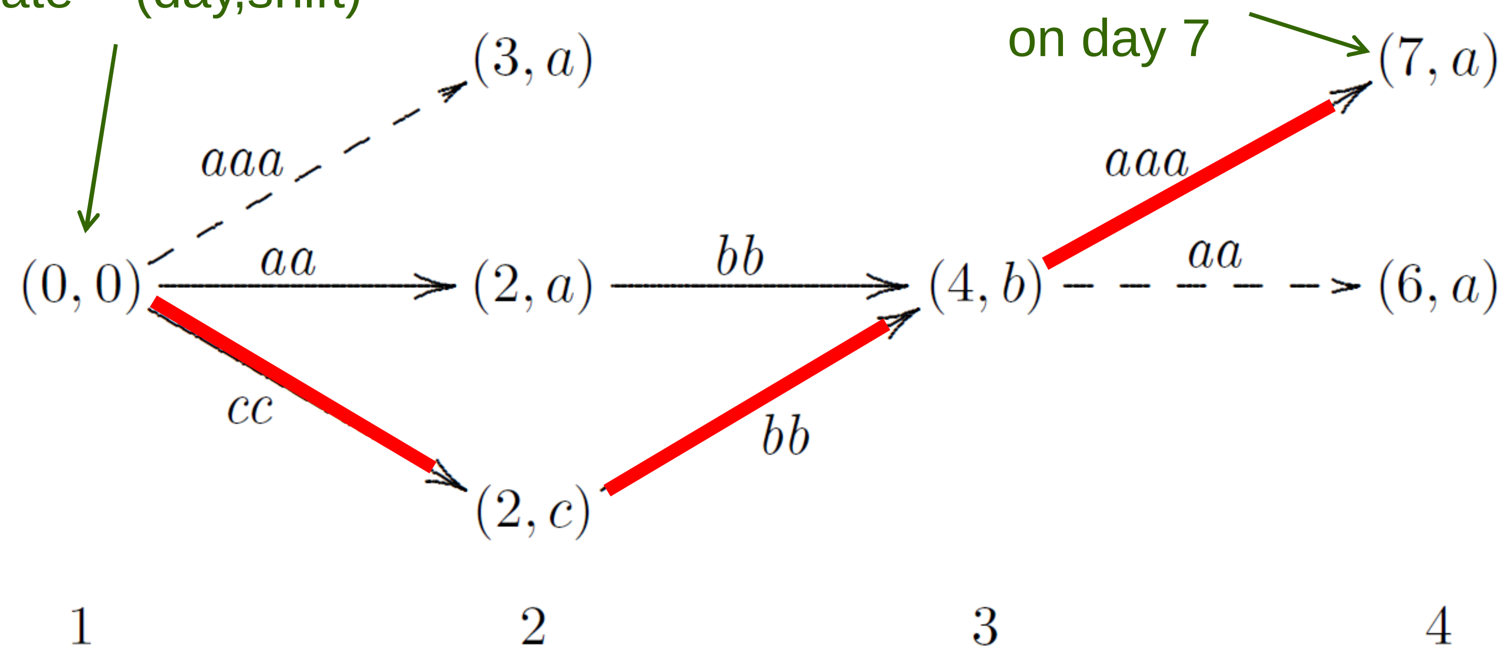
Filter based on dynamic programming

- Example $\text{stretch}\left((x_1,\ldots,x_7),(a,b,c),(2,2,2),(3,3,3),P\right)$

$$P=\{(a,b),(b,a),(b,c),(c,b)\}$$

State = (day,shift)

Must terminate on day 7



Solution: *ccbbaaa*

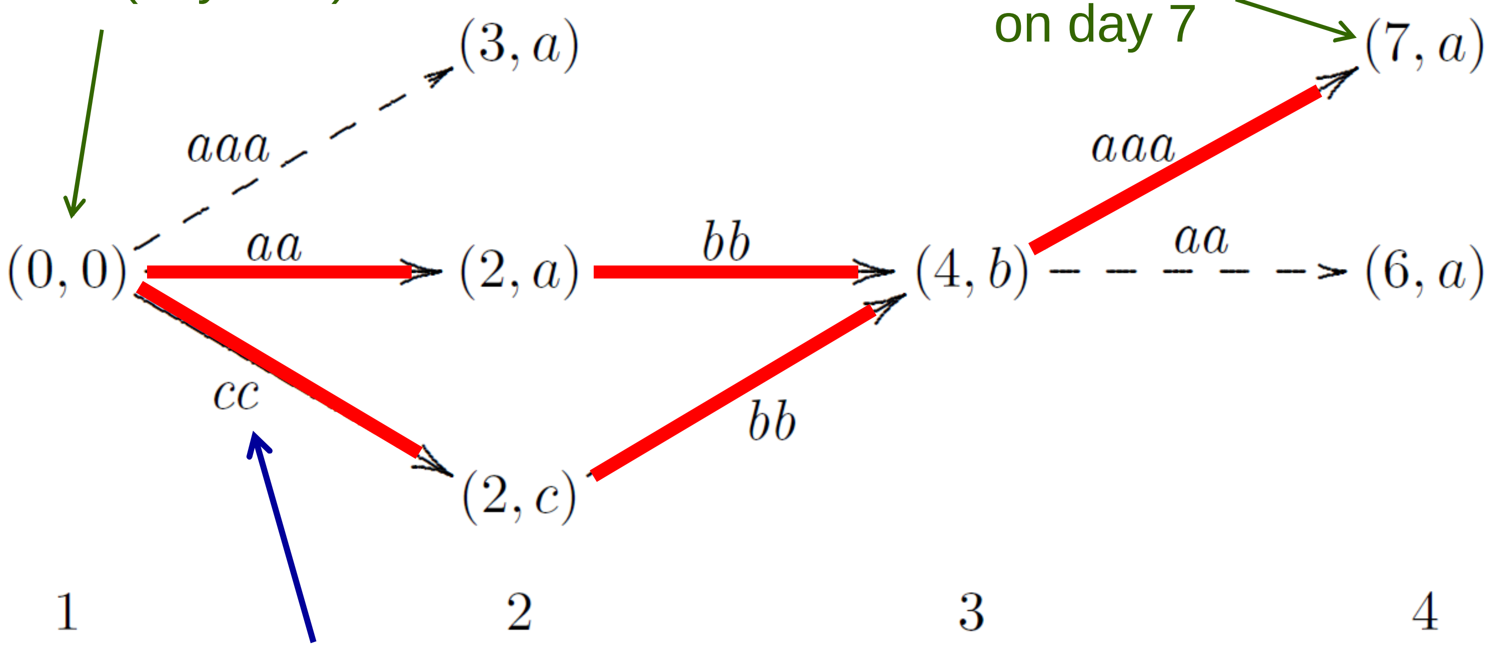
Filter based on dynamic programming

- Example $\text{stretch}\left((x_1,\ldots,x_7),(a,b,c),(2,2,2),(3,3,3),P\right)$

$$P=\{(a,b),(b,a),(b,c),(c,b)\}$$

State = (day,shift)

Must terminate on day 7

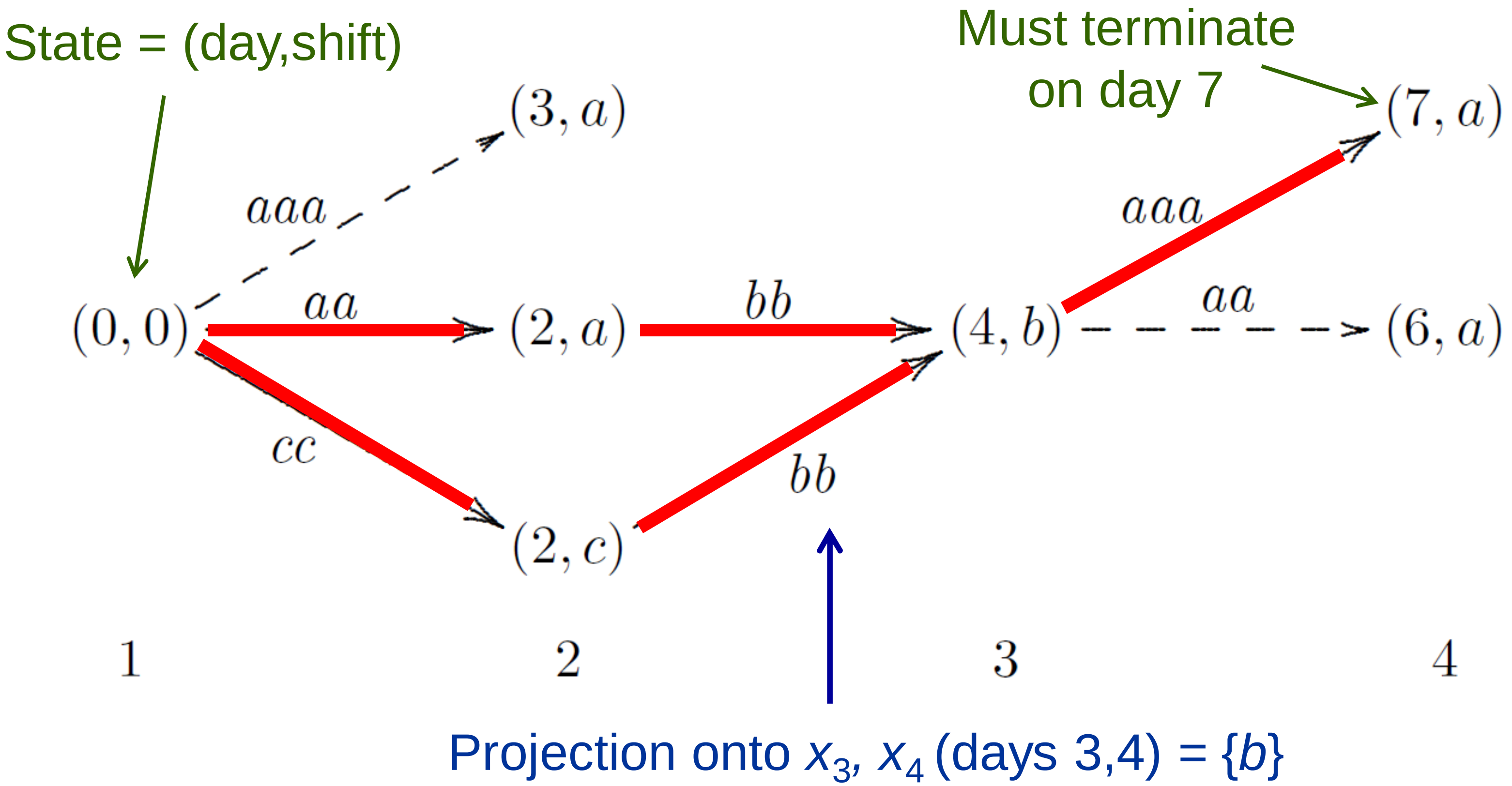


Projection onto x_1, x_2 (days 1, 2) = {a,c}

Filter based on dynamic programming

- Example $\text{stretch}\left((x_1,\ldots,x_7),(a,b,c),(2,2,2),(3,3,3),P\right)$

$$P=\{(a,b),(b,a),(b,c),(c,b)\}$$



State = (day,shift)

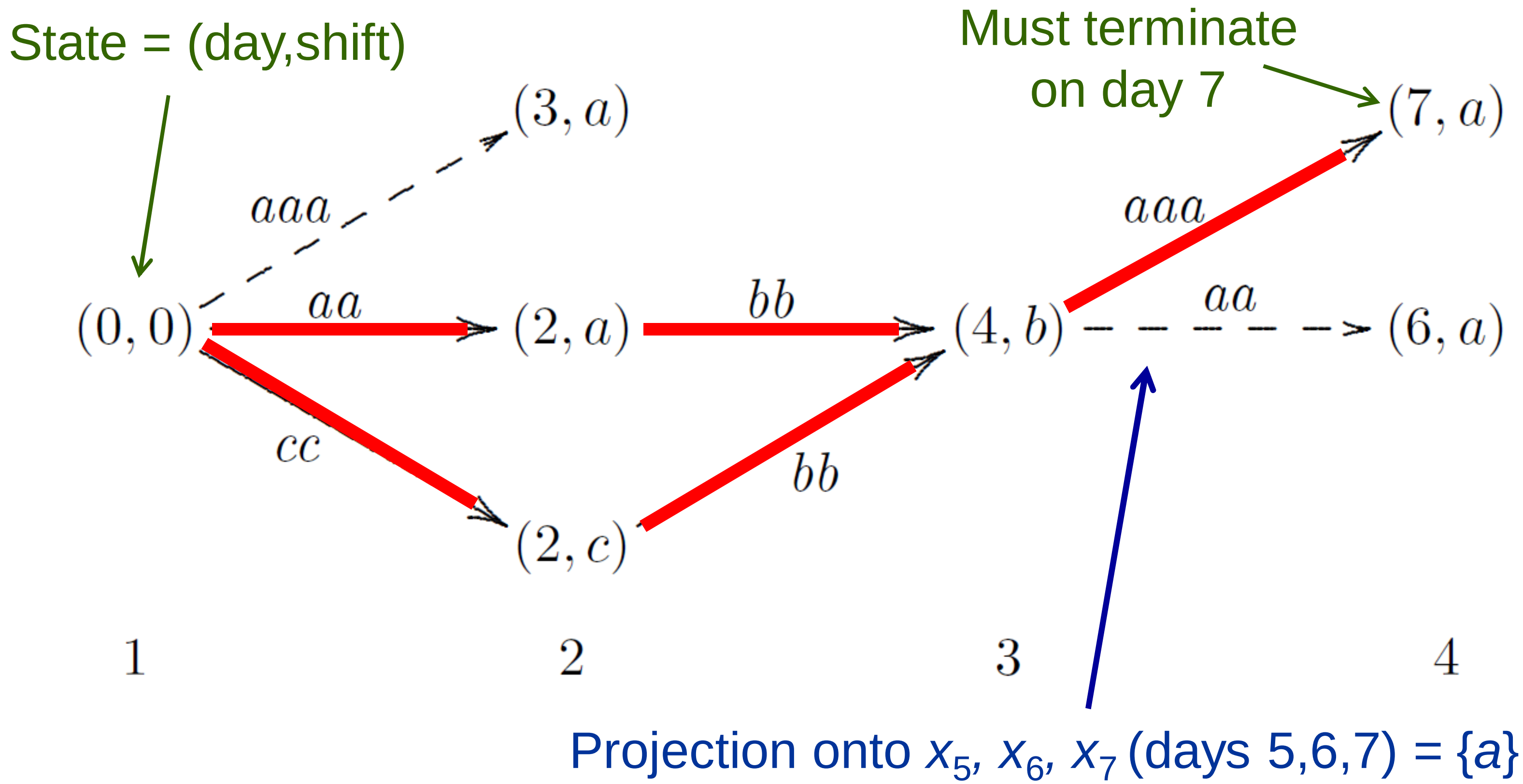
Must terminate on day 7

Projection onto x_3, x_4 (days 3,4) = $\{b\}$

Filter based on dynamic programming

- Example $\text{stretch}\left((x_1,\ldots,x_7),(a,b,c),(2,2,2),(3,3,3),P\right)$

$$P=\{(a,b),(b,a),(b,c),(c,b)\}$$



Projection onto x_5, x_6, x_7 (days 5,6,7) = {a}

Filter based on dynamic programming

- Example $\text{stretch}\left((x_1,\ldots,x_7),(a,b,c),(2,2,2),(3,3,3),P\right)$

$$P=\{(a,b),(b,a),(b,c),(c,b)\}$$

Original domains

x_1	x_2	x_3	x_4	x_5	x_6	x_7
a	a	a		a	a	a
	b	b	b		b	b
c	c		c	c		

Filtered domains

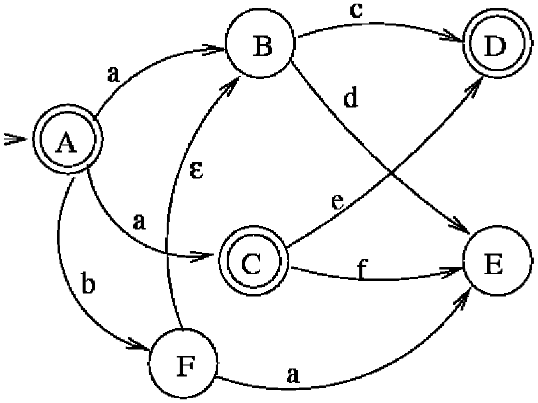
x_1	x_2	x_3	x_4	x_5	x_6	x_7
a	a			a	a	a
		b	b			
c	c					

Filter based on dynamic programming

- The filter is complete (achieves domain consistency).
- There is a clever way to speed up the dynamic programming algorithm.
    - Too complicated to present here.

Stretch-cycle

- The **stretch-cycle** constraint applies to a cycle rather than a linear sequence.
    - Useful for cyclic schedules (e.g., same schedule every week).
    - Dynamic programming filter can be modified for **stretch-cycle**.



Regular Constraint

Finite Automaton Model

Filtering Based on Dynamic Programming

Regular Constraint

- Based on **regular expressions** in Chomsky hierarchy.
    - Deals with any sequencing constraint that can be captured by a **deterministic finite automaton**.
    - …or by a regular expression.
- Used in sequencing and scheduling problems.
    - More general than **stretch**.
- Also filtered by dynamic programming.
    - Or by decomposition

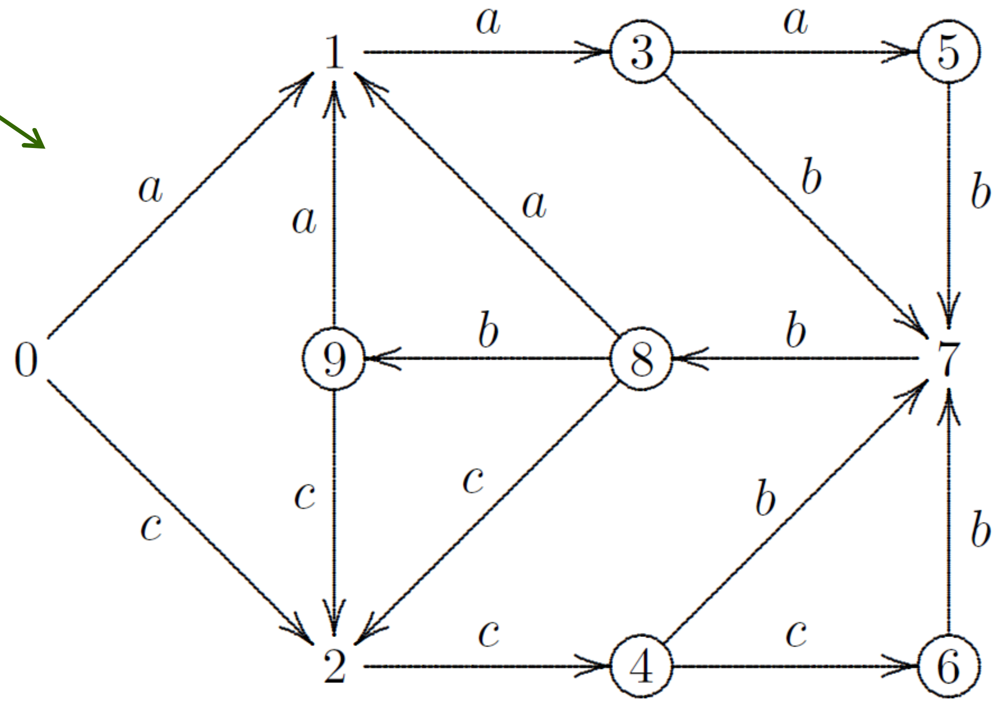
Regular constraint

- Use same stretch example

$$\text{regular}\left((x_1,\ldots,x_7),A\right)$$

x_1	x_2	x_3	x_4	x_5	x_6	x_7
a	a	a		a	a	a
	b	b	b		b	b
c	c		c	c		

Deterministic finite automaton



Initial state

Absorbing states in circles.
State labels are arbitrary.

Regular constraint

- Use same stretch example

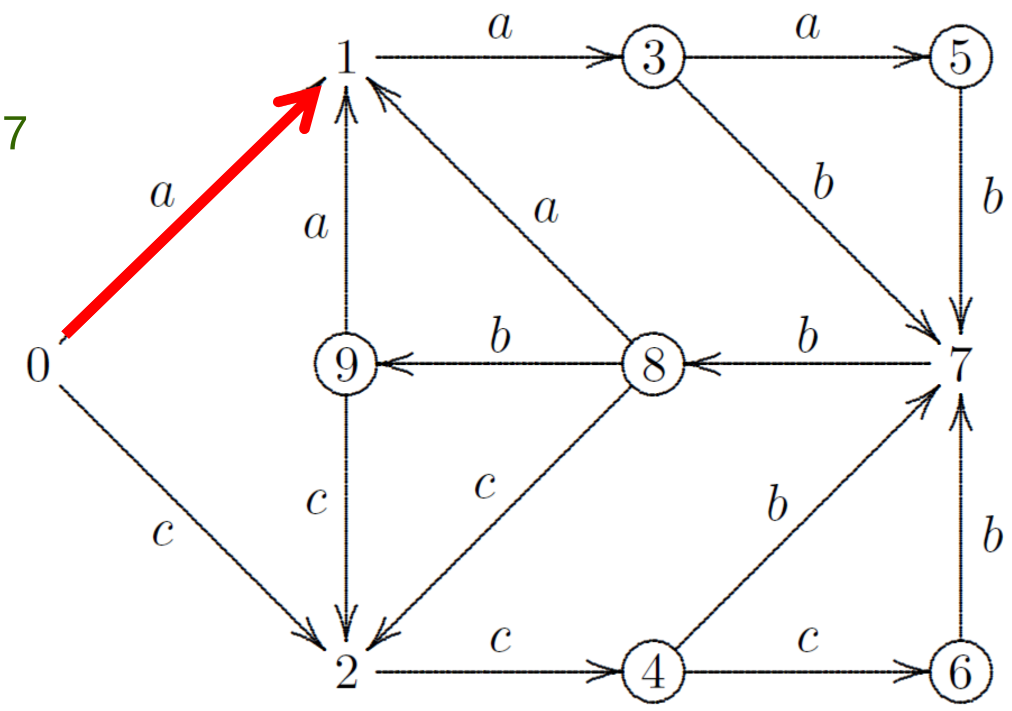
x_1	x_2	x_3	x_4	x_5	x_6	x_7
a	a	a		a	a	a
	b	b	b		b	b
c	c		c	c		

$$\text{regular}\left((x_1,\ldots,x_7),A\right)$$

Deterministic finite automaton

2 solutions of length 7

Solution 1: *a*



Absorbing states in circles.
State labels are arbitrary.

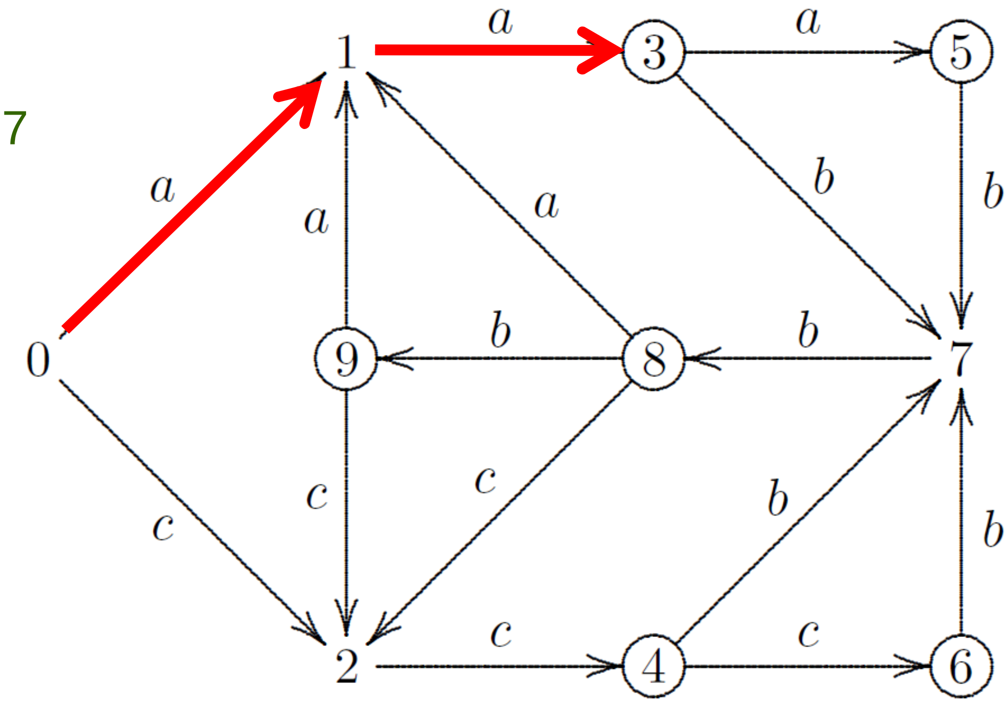
Regular constraint

- Use same stretch example

$$\text{regular}\left((x_1,\ldots,x_7),A\right)$$

x_1	x_2	x_3	x_4	x_5	x_6	x_7
a	a	a		a	a	a
	b	b	b		b	b
c	c		c	c		

Deterministic finite automaton



2 solutions of length 7

Solution 1: *aa*

Absorbing states
in circles.
State labels are
arbitrary.

Regular constraint

- Use same stretch example

$$\text{regular}\left((x_1,\ldots,x_7),A\right)$$

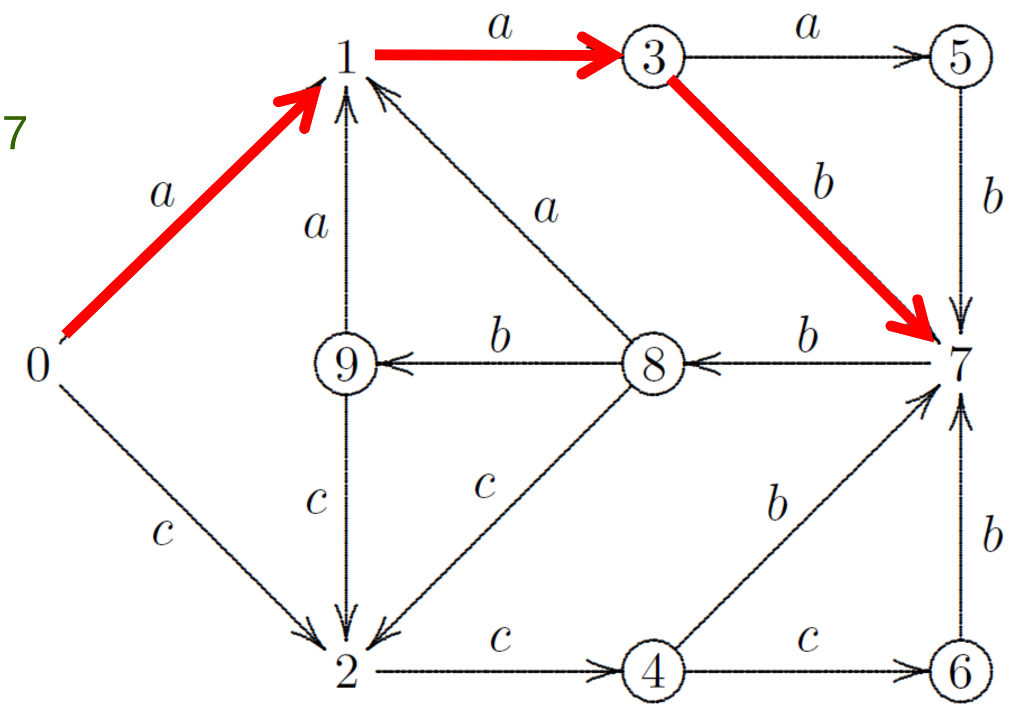
x_1	x_2	x_3	x_4	x_5	x_6	x_7
a	a	a		a	a	a
	b	b	b		b	b
c	c		c	c		

Deterministic finite automaton

2 solutions of length 7

Solution 1: *aab*

Absorbing states in circles.
State labels are arbitrary.



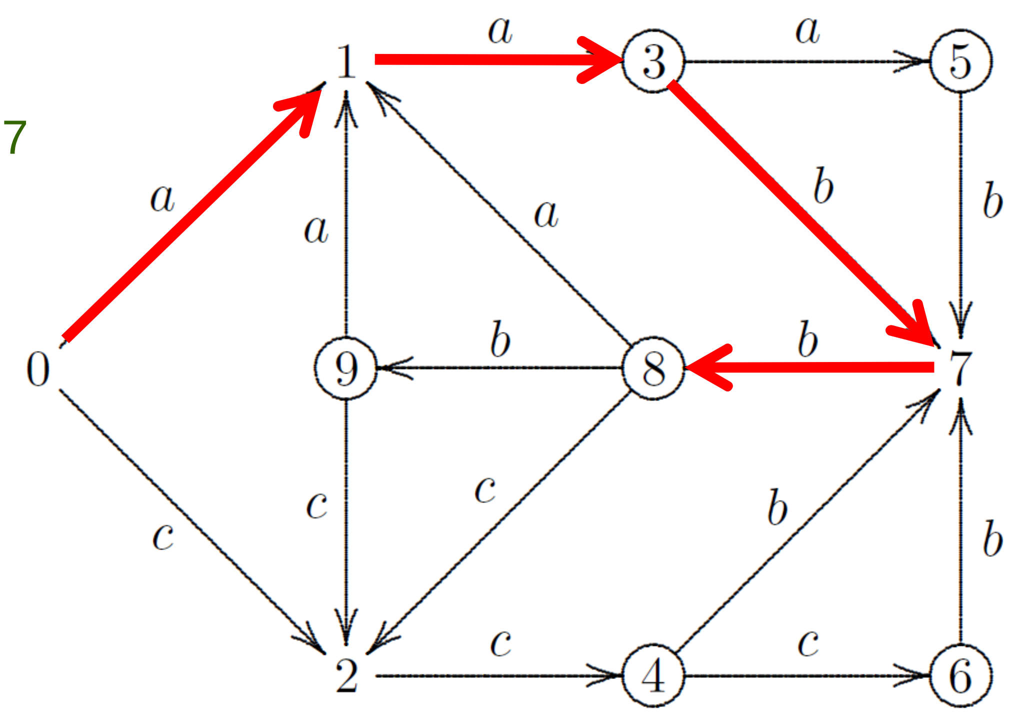
Regular constraint

- Use same stretch example

x_1	x_2	x_3	x_4	x_5	x_6	x_7
a	a	a		a	a	a
	b	b	b		b	b
c	c		c	c		

$$\text{regular}((x_1,\ldots,x_7),A)$$

Deterministic finite automaton



2 solutions of length 7

Solution 1: *aabb*

Absorbing states in circles. State labels are arbitrary.

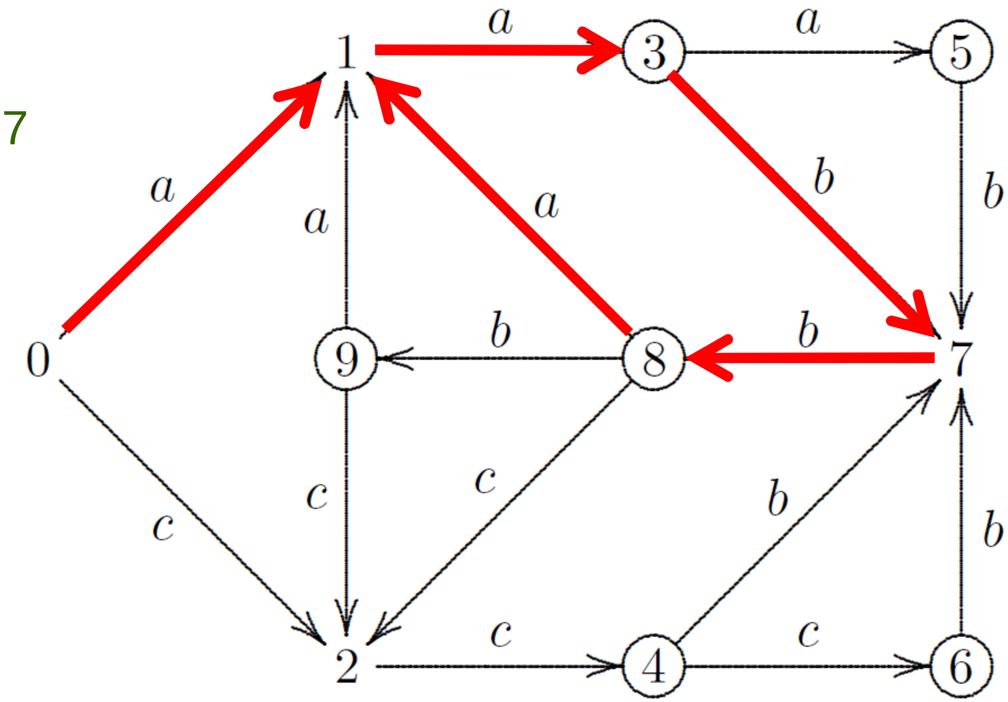
Regular constraint

- Use same stretch example

$$\text{regular}\left((x_1,\ldots,x_7),A\right)$$

x_1	x_2	x_3	x_4	x_5	x_6	x_7
a	a	a		a	a	a
	b	b	b		b	b
c	c		c	c		

Deterministic finite automaton



2 solutions of length 7

Solution 1: *aabba*

Absorbing states in circles.
State labels are arbitrary.

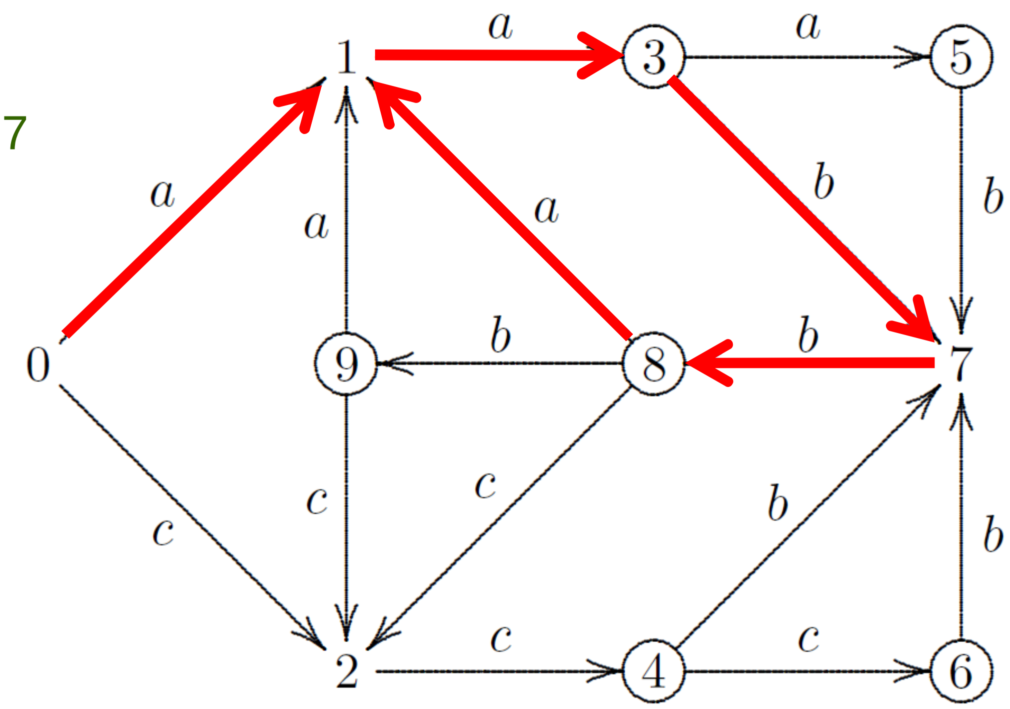
Regular constraint

- Use same stretch example

$$\text{regular}\left((x_1,\ldots,x_7),A\right)$$

x_1	x_2	x_3	x_4	x_5	x_6	x_7
a	a	a		a	a	a
	b	b	b		b	b
c	c		c	c		

Deterministic finite automaton



2 solutions of length 7

Solution 1: *aabbaa*

Absorbing states in circles.
State labels are arbitrary.

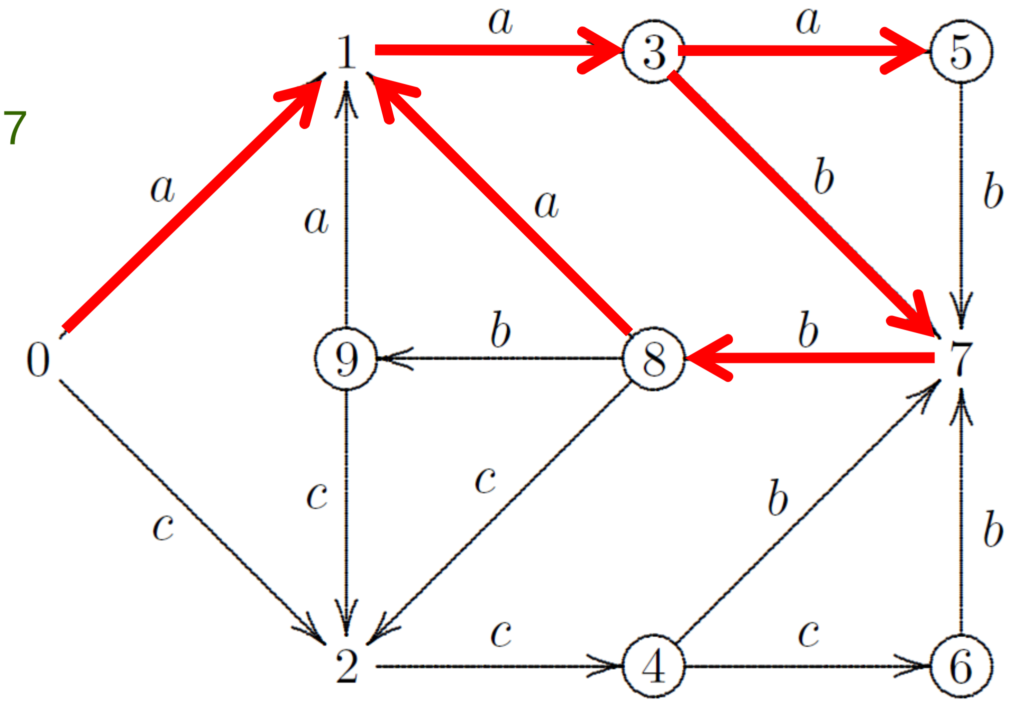
Regular constraint

- Use same stretch example

$$\text{regular}((x_1,\ldots,x_7),A)$$

x_1	x_2	x_3	x_4	x_5	x_6	x_7
a	a	a		a	a	a
	b	b	b		b	b
c	c		c	c		

Deterministic finite automaton



2 solutions of length 7

Solution 1: *aabbaaa*

Absorbing states in circles. State labels are arbitrary.

Regular constraint

- Use same stretch example

$$\text{regular}((x_1,\ldots,x_7),A)$$

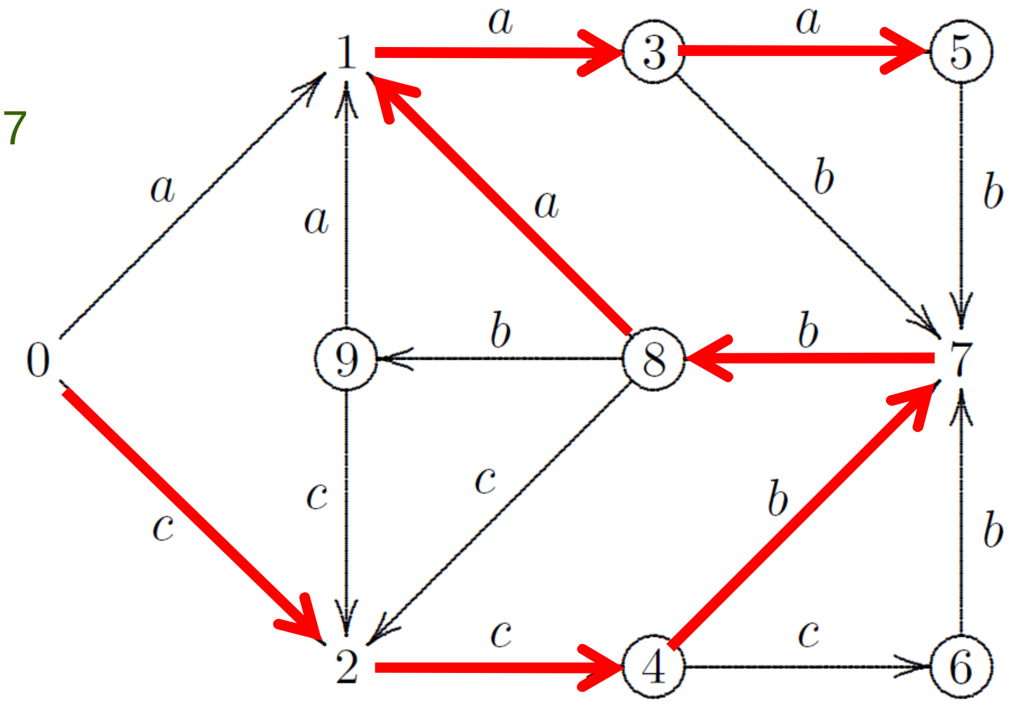
x_1	x_2	x_3	x_4	x_5	x_6	x_7
a	a	a		a	a	a
	b	b	b		b	b
c	c		c	c		

Deterministic finite automaton

2 solutions of length 7

Solution 1: *aabbaaa*
Solution 2: *ccbbaaa*

Absorbing states in circles.
State labels are arbitrary.



Regular constraint

- Use same stretch example

$$\text{regular}((x_1,\ldots,x_7),A)$$

x_1	x_2	x_3	x_4	x_5	x_6	x_7
a	a	a		a	a	a
	b	b	b		b	b
c	c		c	c		

Regular expression:

$$((aaa*bbb*)*\,|\,(ccc*bbb*)*)*(\varepsilon\,|\,aaa*\,|\,ccc*)$$

Kleene star
(repeat 0 or more times)

Empty string

Regular constraint

- Use same stretch example

$$\text{regular}((x_1,\ldots,x_7),A)$$

x_1	x_2	x_3	x_4	x_5	x_6	x_7
a	a	a		a	a	a
	b	b	b		b	b
c	c		c	c		

Regular expression:

$$((aaa*bbb*)*|(ccc*bbb*)*)*(\varepsilon|aaa*|ccc*)$$

Kleene star
(repeat 0 or more times)

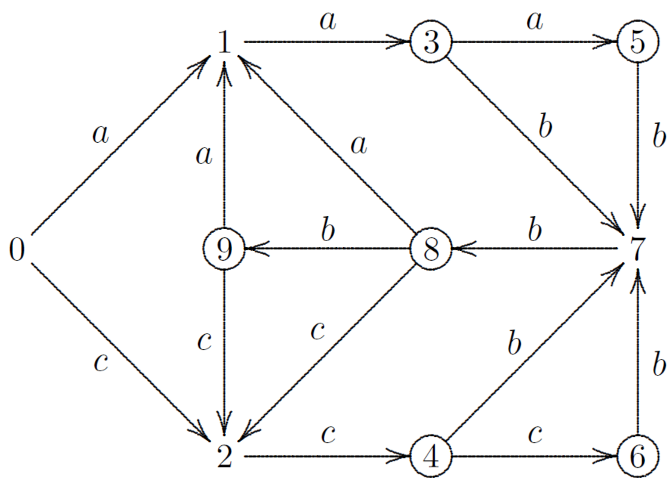
Empty string

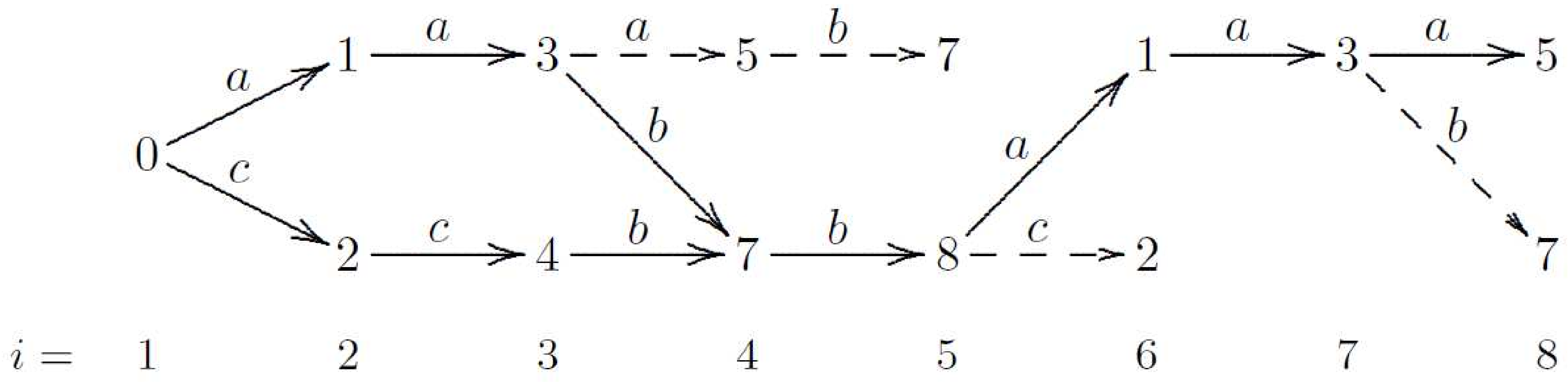
Solutions: *aabbaaa*, *ccbbaaa*

Filtering by dynamic programming

- Use same stretch example

$$\text{regular}((x_1,\ldots,x_7),A)$$





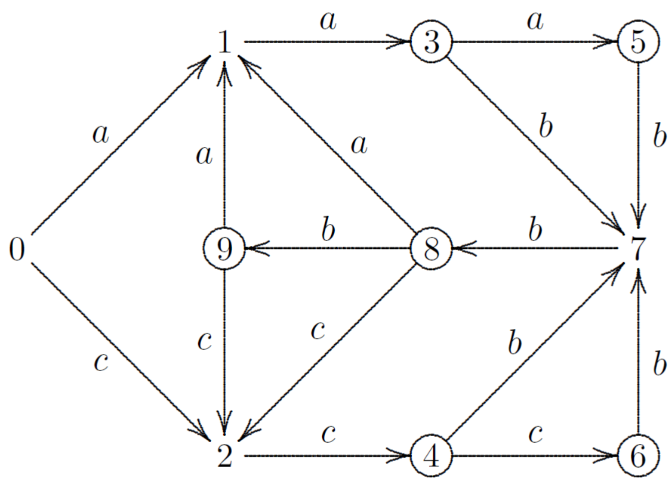
Stage (day)

x_1	x_2	x_3	x_4	x_5	x_6	x_7
a	a	a		a	a	a
	b	b	b		b	b
c	c		c	c		

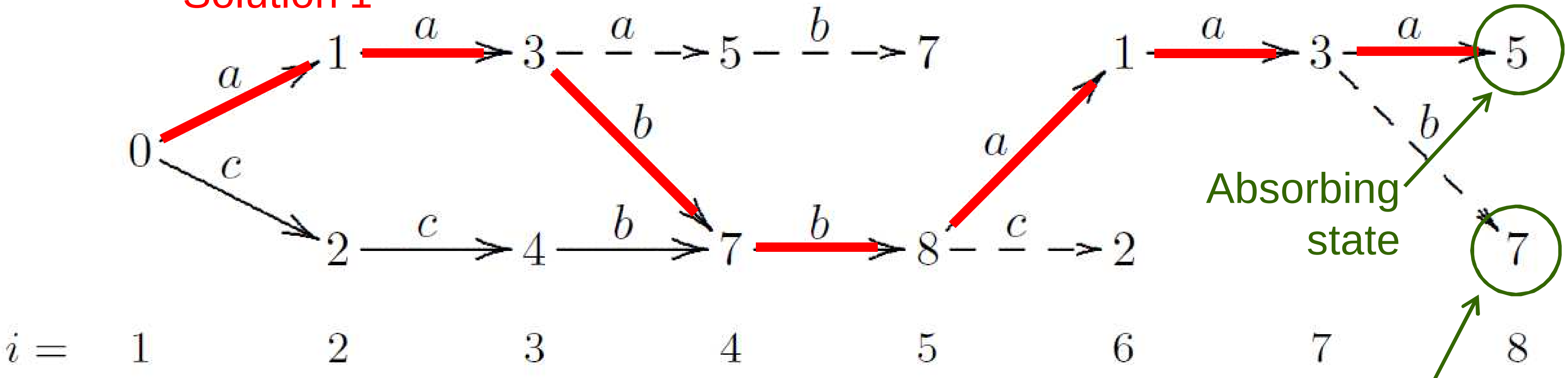
Filtering by dynamic programming

- Use same stretch example

$$\text{regular}((x_1,\ldots,x_7),A)$$



Solution 1



Absorbing state

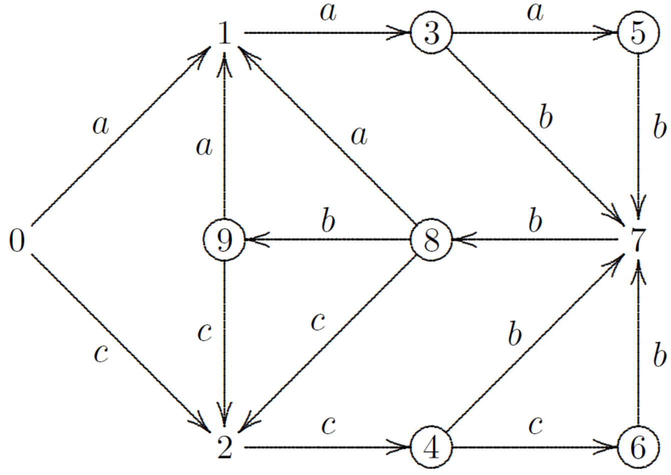
Not an absorbing state

x_1	x_2	x_3	x_4	x_5	x_6	x_7
a	a	a		a	a	a
	b	b	b		b	b
c	c		c	c		

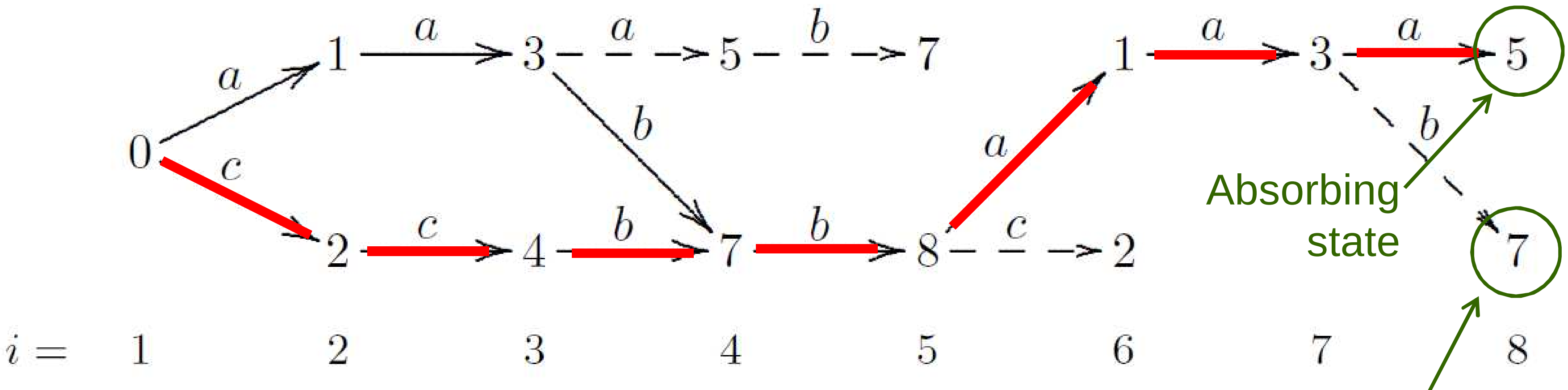
Filtering by dynamic programming

- Use same stretch example

$$\text{regular}((x_1,\ldots,x_7),A)$$



Solution 2



Absorbing state

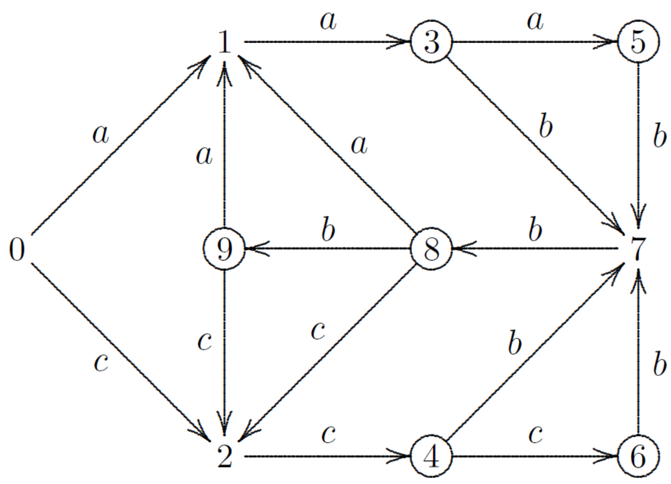
Not an absorbing state

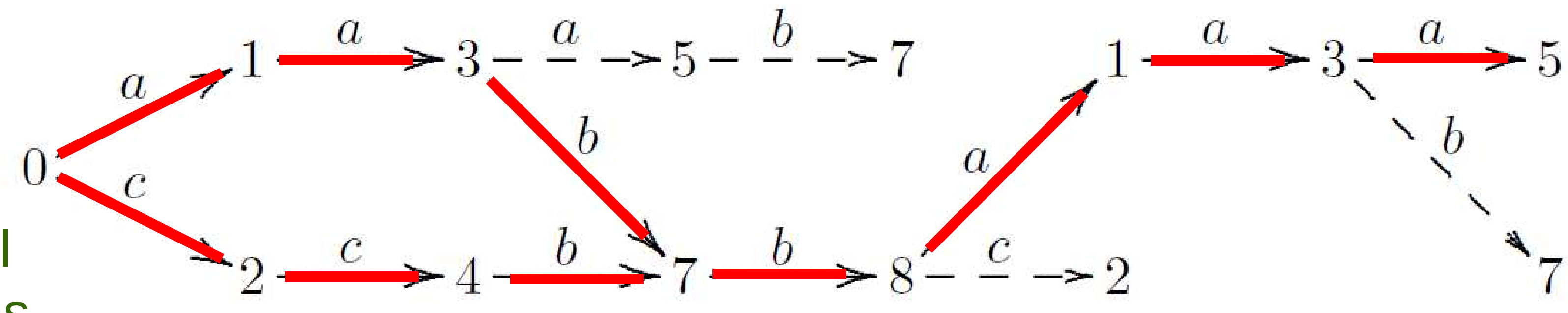
x_1	x_2	x_3	x_4	x_5	x_6	x_7
a	a	a		a	a	a
	b	b	b		b	b
c	c		c	c		

Filtering by dynamic programming

- Use same stretch example

$$\text{regular}((x_1,\ldots,x_7),A)$$





Original domains

$i =$	1	2	3	4	5	6	7	8
$D_{x_i} =$	$\{a,c\}$	$\{a,b,c\}$	$\{a,b\}$	$\{b,c\}$	$\{a,c\}$	$\{a,b\}$	$\{a,b\}$	
$D'_{x_i} =$	$\{a,c\}$	$\{a,c\}$	$\{b\}$	$\{b\}$	$\{a\}$	$\{a\}$	$\{a\}$	

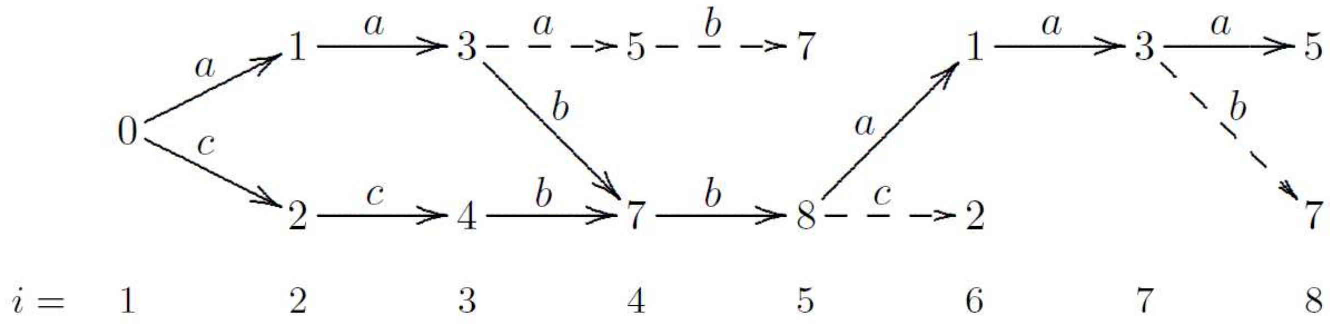
Filtered domains (projections onto each variable)

Filtering by dynamic programming

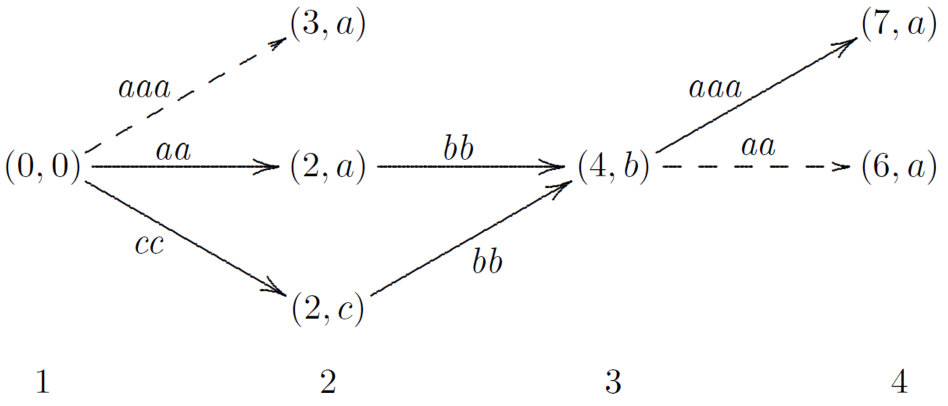
- Use same stretch example

$\text{regular}((x_1,\ldots,x_7),A)$

x_1	x_2	x_3	x_4	x_5	x_6	x_7
a	a	a		a	a	a
	b	b	b		b	b
c	c		c	c		



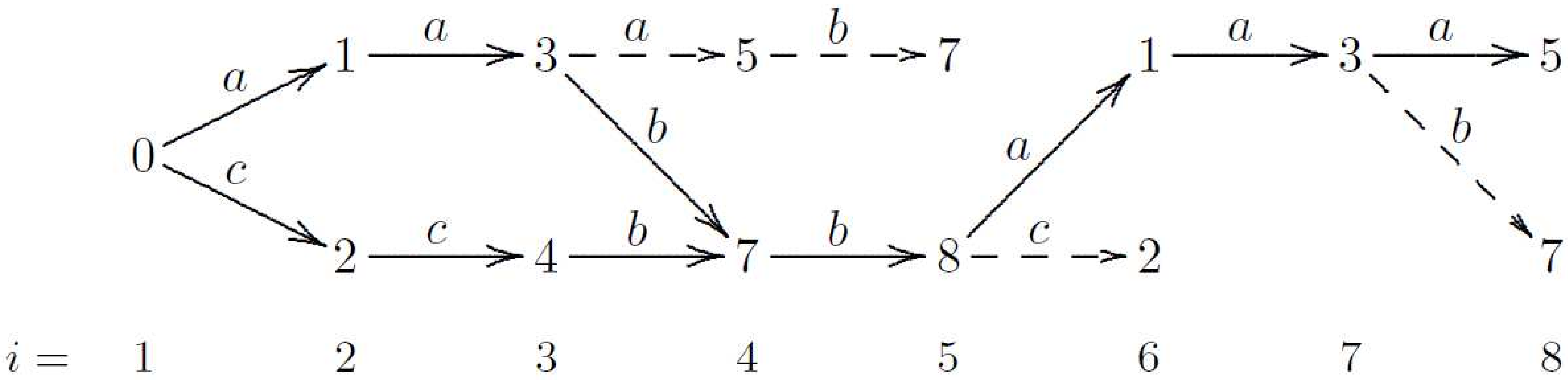
Compare with DP model for **stretch**



Dynamic programming model

- Alternative: Formulate the problem as dynamic programming from the start.

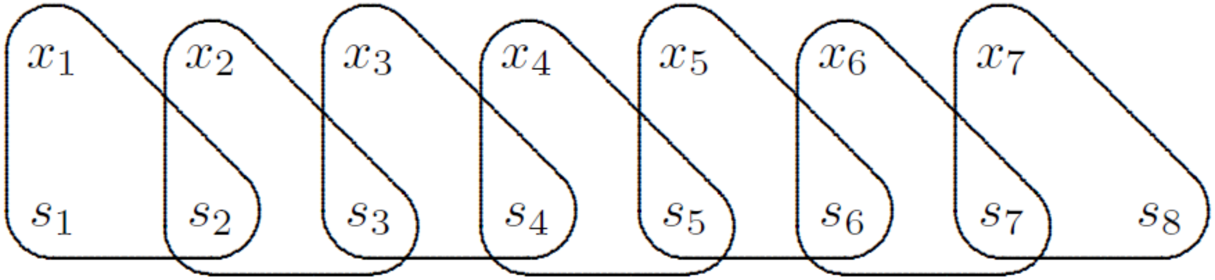
x_1	x_2	x_3	x_4	x_5	x_6	x_7
a	a	a		a	a	a
	b	b	b		b	b
c	c		c	c		



Filtering by decomposition

- Recursive equations: $s_{i+1} = t_{i+1}(s_i, x_i), \ i = 1, \ldots, 7$
    - where $t_{i+1}()$ are transition functions, s_i is state variable.
    - Propagate these equations in 2 passes (forward and backward).
    - This achieves domain consistency because constraint hypergraph is Berge acyclic.
    - Based on a result from database theory.

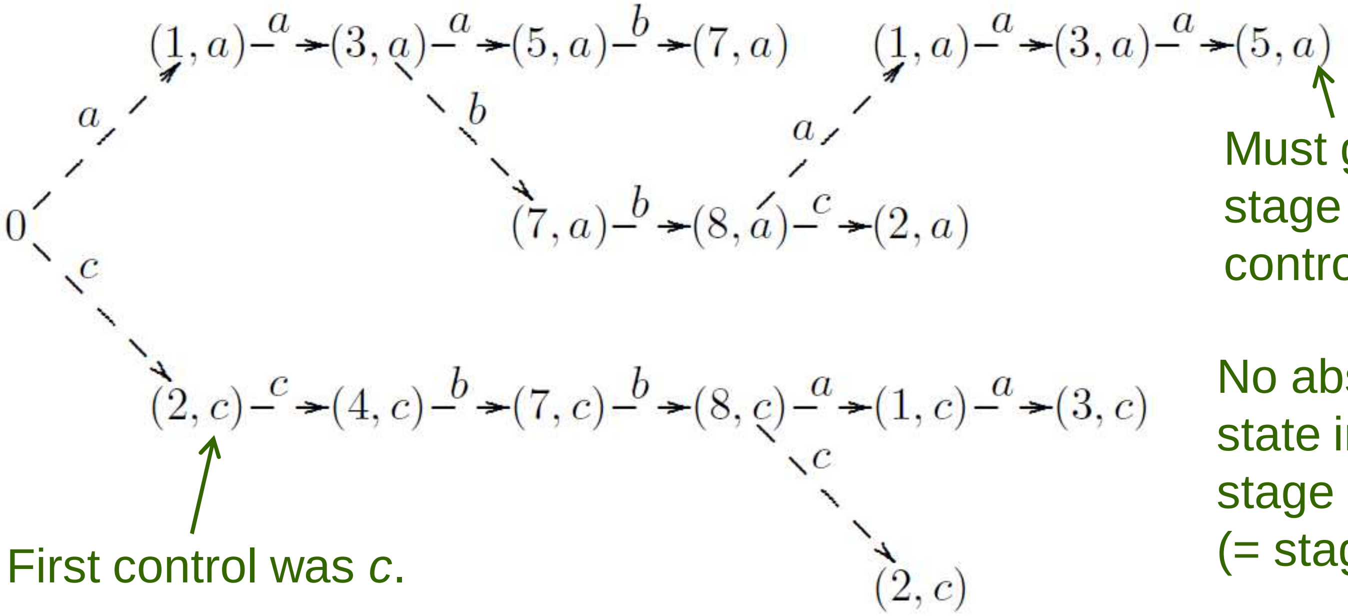
Constraint hypergraph



- Filtering by decomposition is an active research area iln CP.

Cyclic regular constraint

- The regular-cycle constraint is filtered by using an additional state variable to indicate the first control.



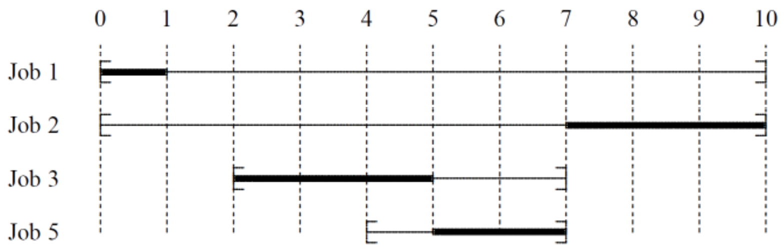
Must go to stage 9 with control *a*.

No absorbing state in last stage (= stage 1)

First control was *c*.

$i =$	1	2	3	4	5	6	7	8	9
$D_{x_i} =$	$\{a,c\}$	$\{a,b,c\}$	$\{a,b\}$	$\{b,c\}$	$\{a,c\}$	$\{a,b\}$	$\{a,b\}$	$\{a,c\}$	
$D'_{x_i} =$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	$\emptyset$	

Example problem is infeasible.



Disjunctive Scheduling

Edge Finding

Not-first/Not-last Rules

Disjunctive scheduling

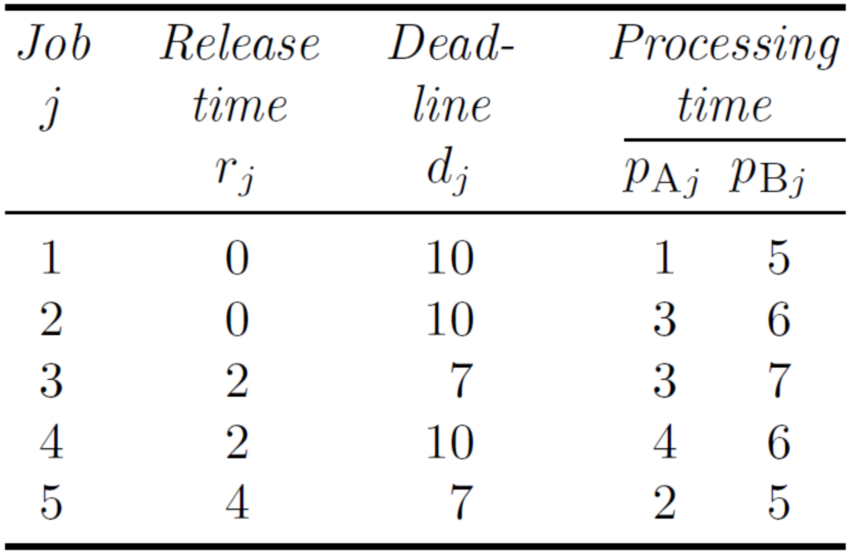
- **Disjunctive scheduling** assigns start times to jobs so that they do not overlap.
    - Also known as **single machine scheduling** problem
    - Jobs have release times and deadlines
    - There may be precedence constraints
    - Various objective functions
        - Makespan, number of late jobs, total tardiness, etc.
- Filtering is well developed.
    - Edge finding (old OR technique by Carlier and Pinson)
    - Not-first/not-last rules

Disjunctive scheduling

Consider a disjunctive scheduling constraint:

$$\text{noOverlap}\left(\left(\boxed{s_1, s_2, s_3, s_5}\right), (p_1, p_2, p_3, p_5)\right)$$

Start time variables



Job j	*Release time* r_j	*Dead-line* d_j	*Processing time*	
			p_{Aj}	p_{Bj}
1	0	10	1	5
2	0	10	3	6
3	2	7	3	7
4	2	10	4	6
5	4	7	2	5

Edge finding for disjunctive scheduling

Consider a disjunctive scheduling constraint:

$$\text{noOverlap}\big((s_1,s_2,s_3,s_5),\boxed{(p_1,p_2,p_3,p_5}\,)\big)$$

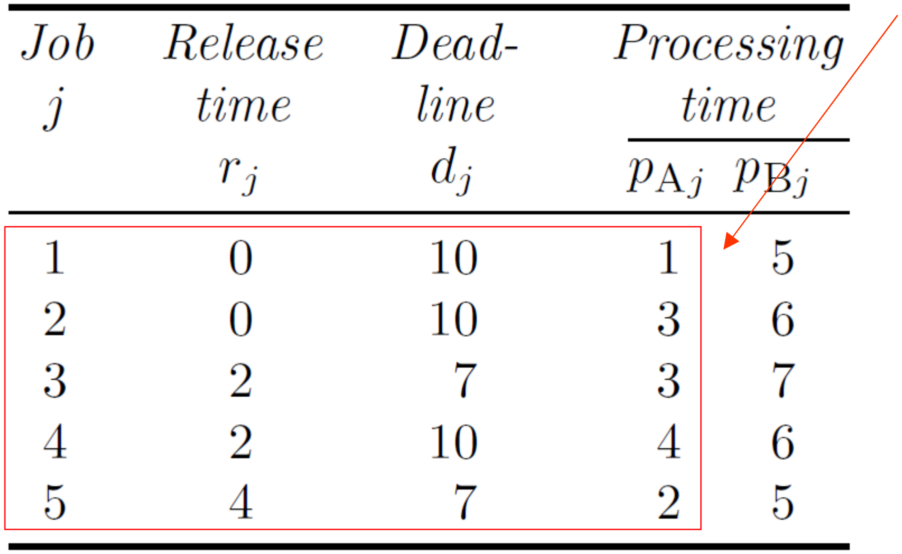
Job j	*Release time* r_j	*Dead-line* d_j	*Processing time*	
			$p_{\mathrm{A}j}$	$p_{\mathrm{B}j}$
1	0	10	1	5
2	0	10	3	6
3	2	7	3	7
4	2	10	4	6
5	4	7	2	5

Processing times

Edge finding for disjunctive scheduling

Consider a disjunctive scheduling constraint:

$$\text{noOverlap}\big((s_1,s_2,s_3,s_5),(p_1,p_2,p_3,p_5)\big)$$



Job	*Release*	*Dead-*	*Processing*	
j	*time*	*line*	*time*	
	r_j	d_j	$p_{\mathrm{A}j}$	$p_{\mathrm{B}j}$
1	0	10	1	5
2	0	10	3	6
3	2	7	3	7
4	2	10	4	6
5	4	7	2	5

Variable domains defined by time windows and processing times

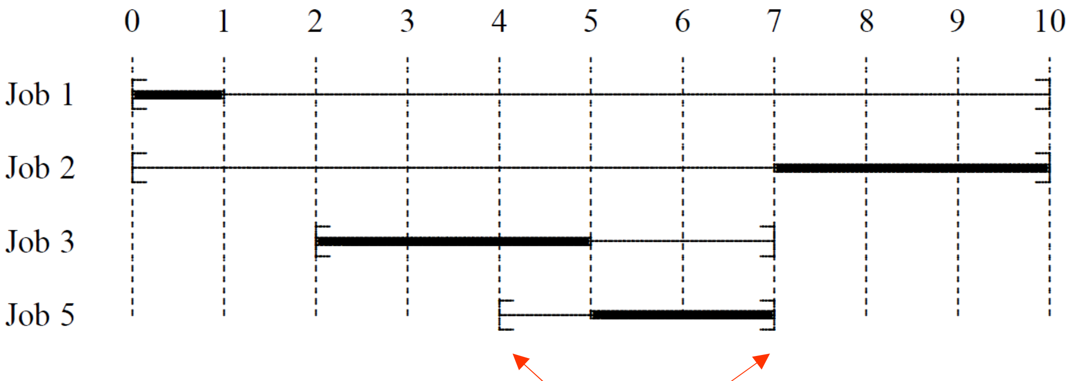
$$s_1 \in [0,10-1]$$
$$s_2 \in [0,10-3]$$
$$s_3 \in [2,7-3]$$
$$s_5 \in [4,7-2]$$

Edge finding for disjunctive scheduling

Consider a disjunctive scheduling constraint:

$$\text{noOverlap}\big((s_1,s_2,s_3,s_5),(p_1,p_2,p_3,p_5)\big)$$

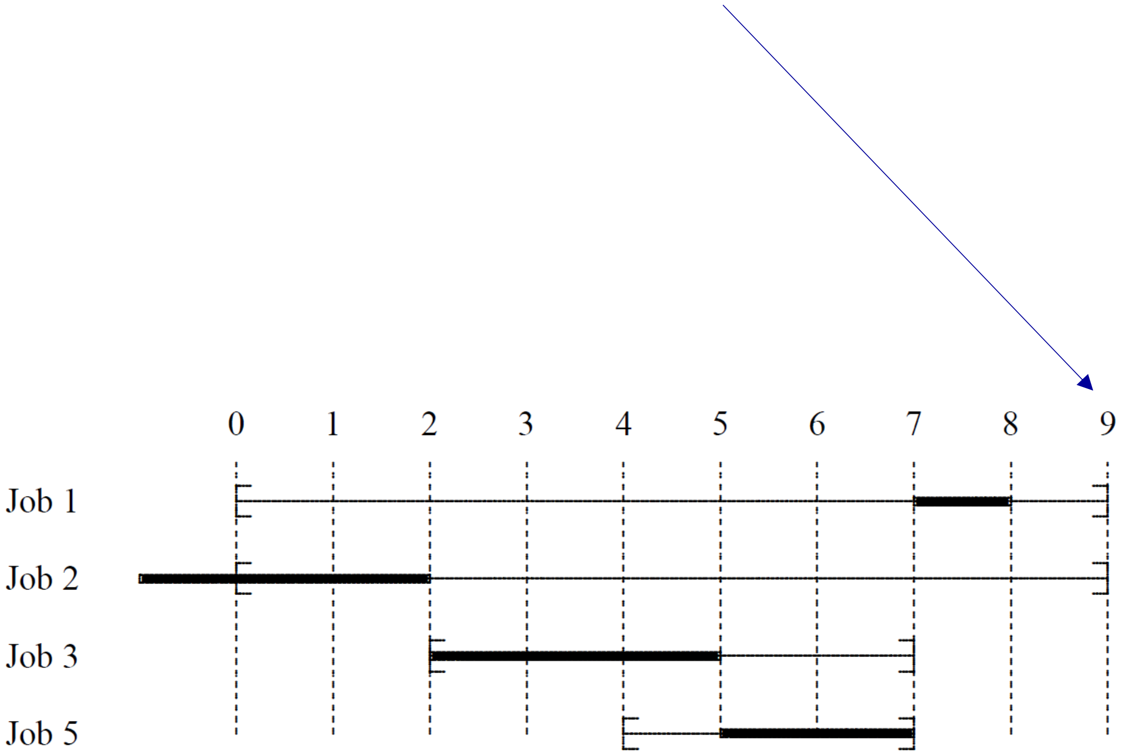
A feasible (min makespan) solution:



Time window

Edge finding for disjunctive scheduling

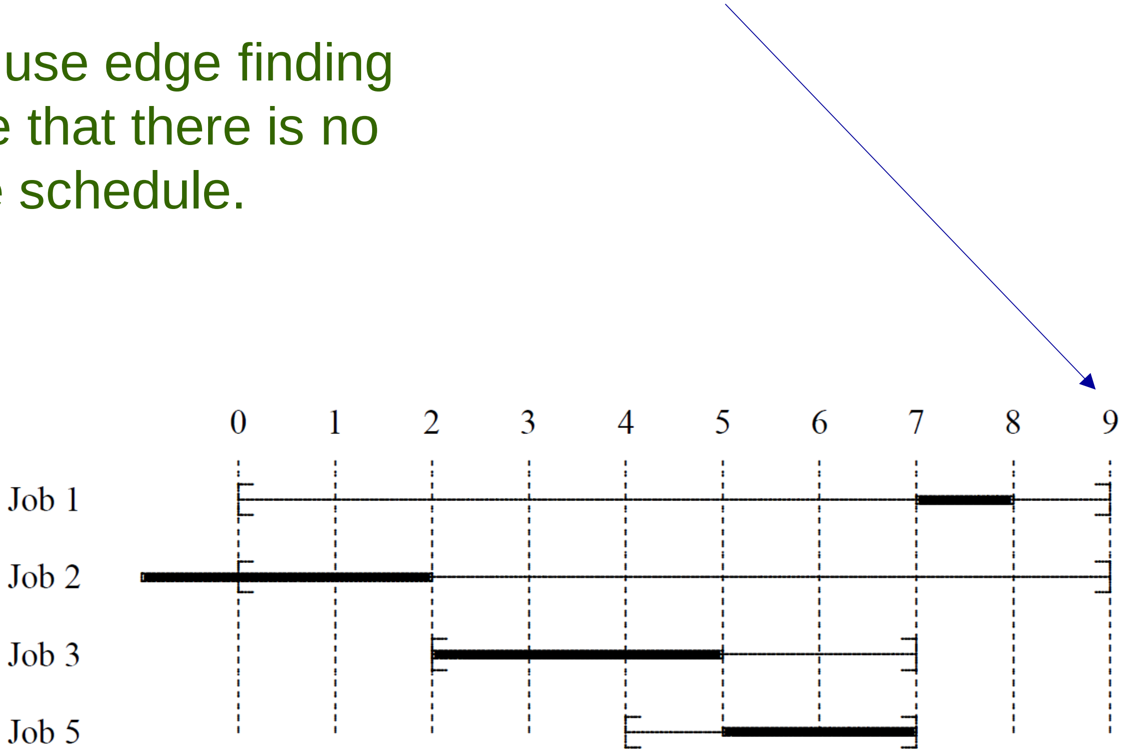
But let’s reduce 2 of the deadlines to 9:



Edge finding for disjunctive scheduling

But let’s reduce 2 of the deadlines to 9:

We will use edge finding to prove that there is no feasible schedule.

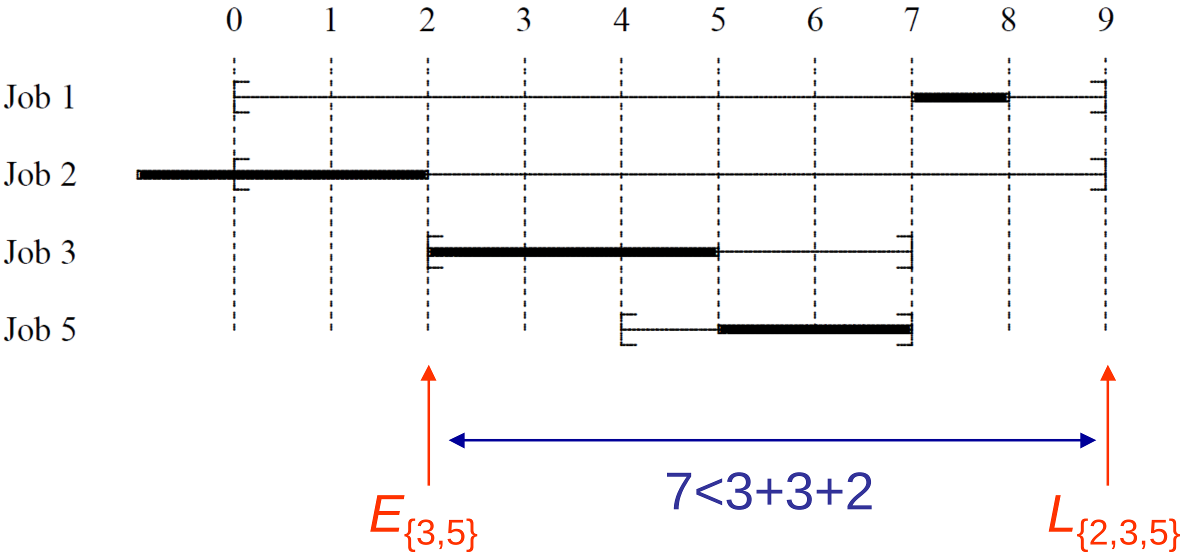


Edge finding for disjunctive scheduling

We can deduce that job 2 must precede jobs 3 and 5: $2 \ll \{3,5\}$

Because if job 2 is not first, there is not enough time for all 3 jobs within the time windows:

$$L_{\{2,3,5\}} - E_{\{3,5\}} < p_{\{2,3,5\}}$$



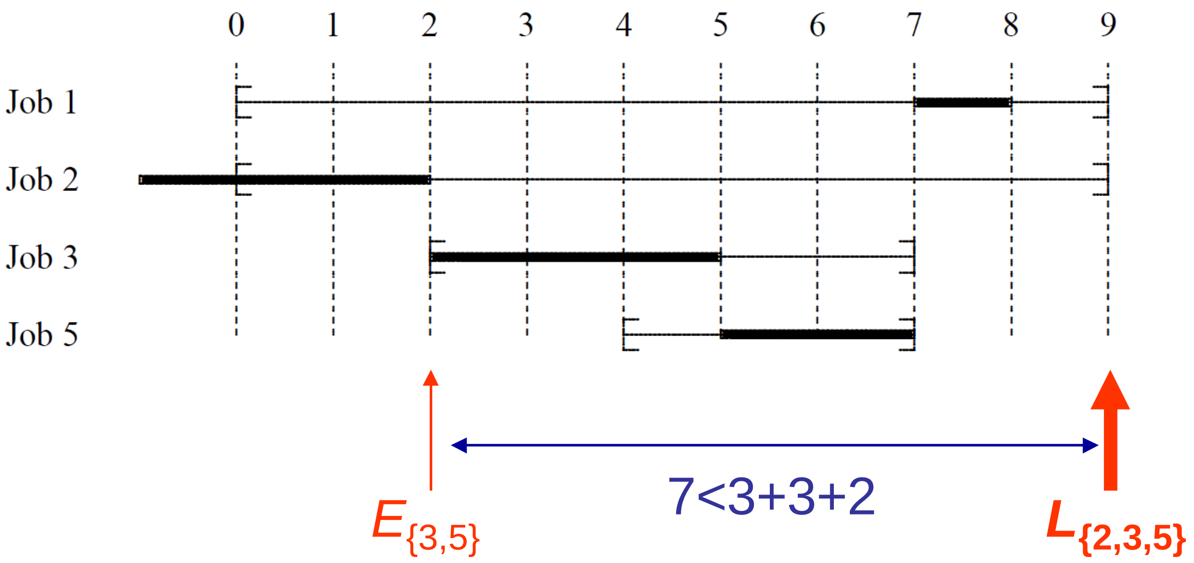
Edge finding for disjunctive scheduling

We can deduce that job 2 must precede jobs 3 and 5: $2 \ll \{3,5\}$

Because if job 2 is not first, there is not enough time for all 3 jobs within the time windows:

$$\boxed{L_{\{2,3,5\}}} - E_{\{3,5\}} < p_{\{2,3,5\}}$$

Latest deadline



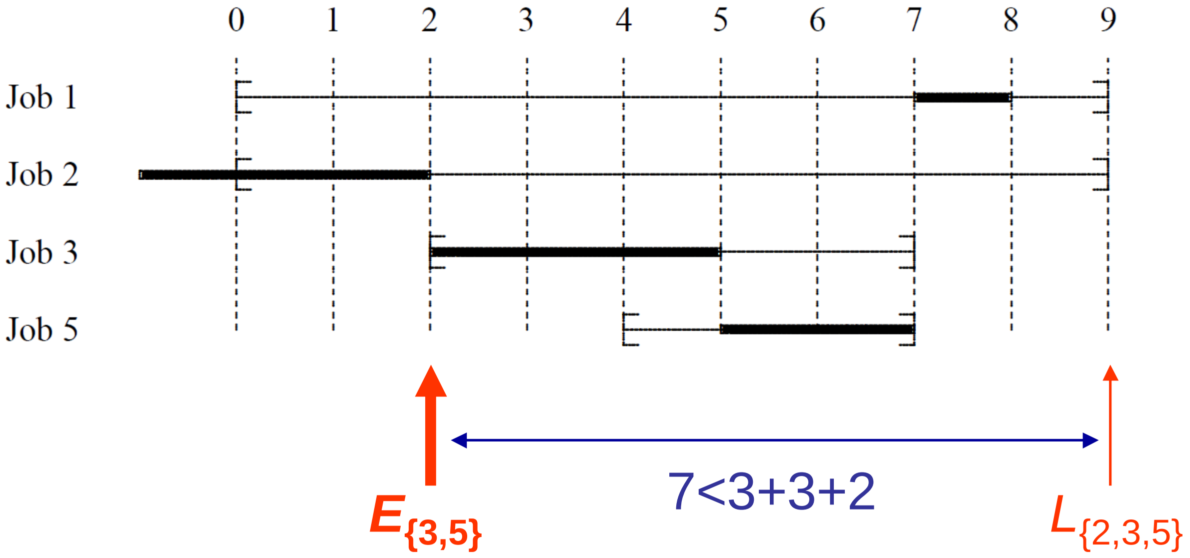
Edge finding for disjunctive scheduling

We can deduce that job 2 must precede jobs 3 and 5: $2 \ll \{3,5\}$

Because if job 2 is not first, there is not enough time for all 3 jobs within the time windows:

$$L_{\{2,3,5\}} - \boxed{E_{\{3,5\}}} < p_{\{2,3,5\}}$$

Earliest release time



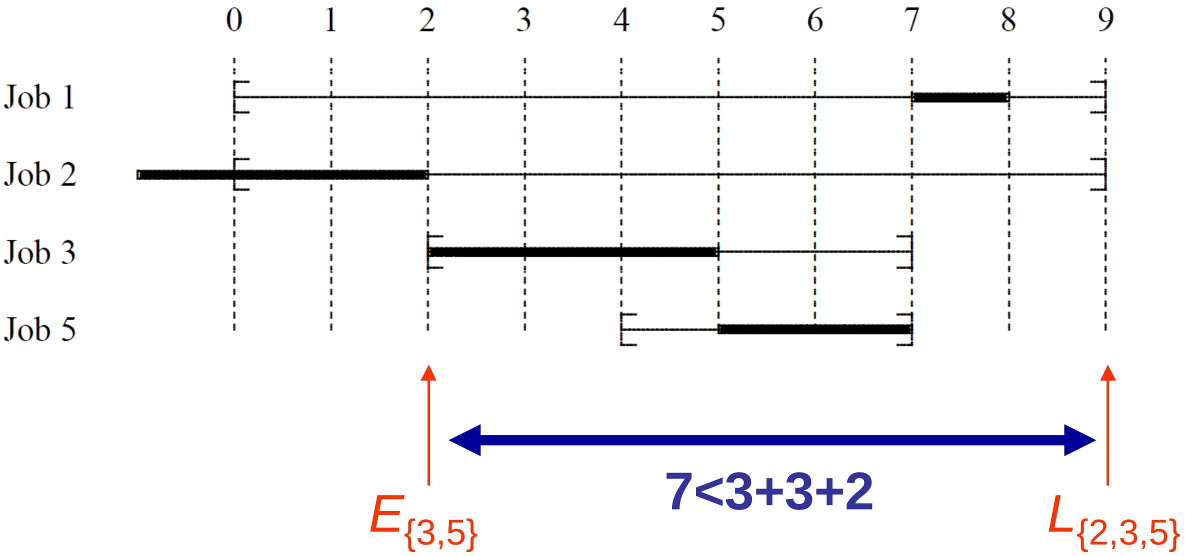
Edge finding for disjunctive scheduling

We can deduce that job 2 must precede jobs 3 and 5: $2 \ll \{3,5\}$

Because if job 2 is not first, there is not enough time for all 3 jobs within the time windows:

$$L_{\{2,3,5\}} - E_{\{3,5\}} < \boxed{p_{\{2,3,5\}}}$$

Total processing time



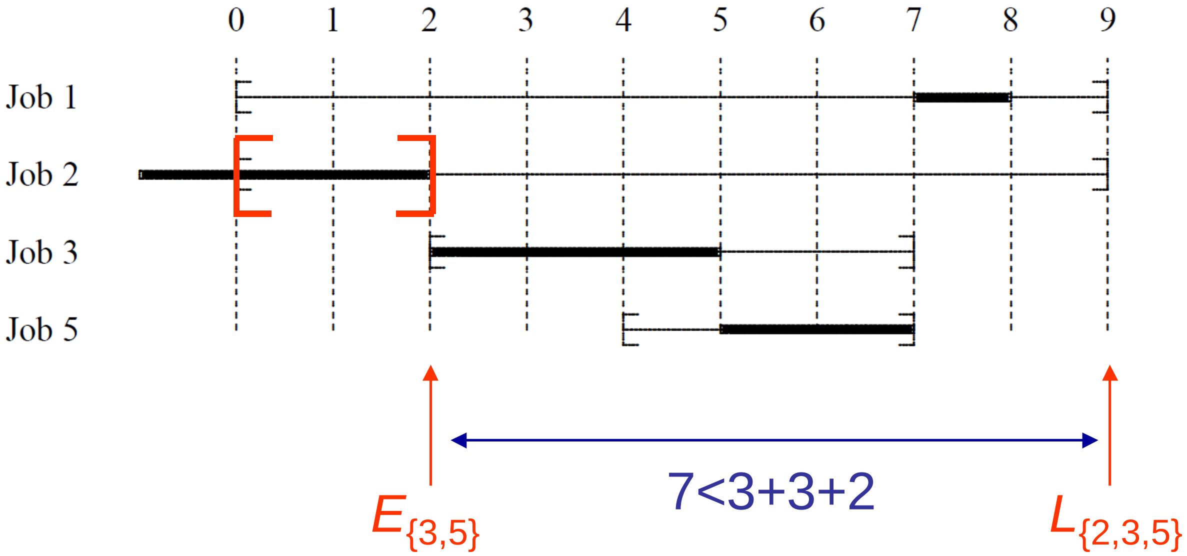
Edge finding for disjunctive scheduling

We can deduce that job 2 must precede jobs 3 and 5: $2 \ll \{3,5\}$

So we can tighten deadline of job 2 to minimum of

$$L_{\{3\}} - p_{\{3\}} = 4 \qquad L_{\{5\}} - p_{\{5\}} = 5 \qquad L_{\{3,5\}} - p_{\{3,5\}} = 2$$

Since time window of job 2 is now too narrow, there is no feasible schedule.



Edge finding for disjunctive scheduling

In general, we can deduce that job *k* must precede all the jobs in set *J*: $k \ll J$

If there is not enough time for all the jobs after the earliest release time of the jobs in *J*

$$L_{J\cup\{k\}} - E_J < p_{J\cup\{k\}} \qquad L_{\{2,3,5\}} - E_{\{3,5\}} < p_{\{2,3,5\}}$$

Edge finding for disjunctive scheduling

In general, we can deduce that job *k* must precede all the jobs in set *J*: $k \ll J$

If there is not enough time for all the jobs after the earliest release time of the jobs in *J*

$$L_{J\cup\{k\}} - E_J < p_{J\cup\{k\}} \qquad L_{\{2,3,5\}} - E_{\{3,5\}} < p_{\{2,3,5\}}$$

Now we can tighten the deadline for job *k* to:

$$\min_{J'\subset J}\left\{L_{J'} - p_{J'}\right\} \qquad L_{\{3,5\}} - p_{\{3,5\}} = 2$$

Edge finding for disjunctive scheduling

There is a symmetric rule: $k \gg J$

If there is not enough time for all the jobs before the latest deadline of the jobs in *J:*

$$L_J - E_{J \cup \{k\}} < p_{J \cup \{k\}}$$

Now we can tighten the release date for job *k* to*:*

$$\max_{J' \subset J} \left\{ E_{J'} + p_{J'} \right\}$$

Edge finding for disjunctive scheduling

Problem: how can we avoid enumerating all subsets *J* of jobs to find edges?

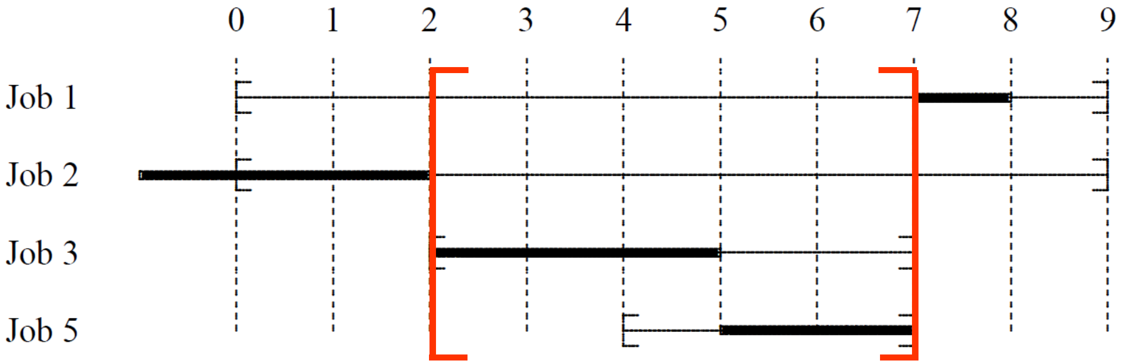
$$L_{J\cup\{k\}} - E_J < p_{J\cup\{k\}}$$

…and all subsets *J´* of *J* to tighten the bounds?

$$\min_{J'\subset J}\left\{L_{J'} - p_{J'}\right\}$$

Edge finding for disjunctive scheduling

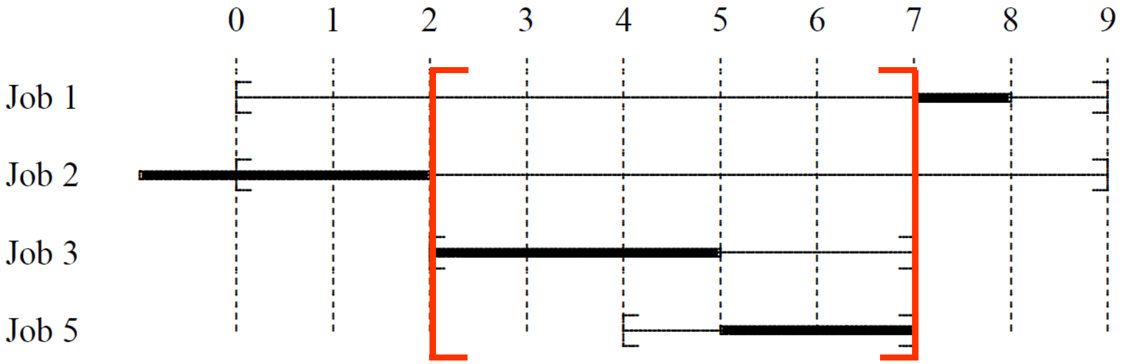
Key result: We only have to consider sets *J* whose time windows lie within some interval between release times/deadlines



e.g., *J* = {3,5}

Edge finding for disjunctive scheduling

Key result: We only have to consider sets *J* whose time windows lie within some interval between release times/deadlines.



e.g., *J* = {3,5}

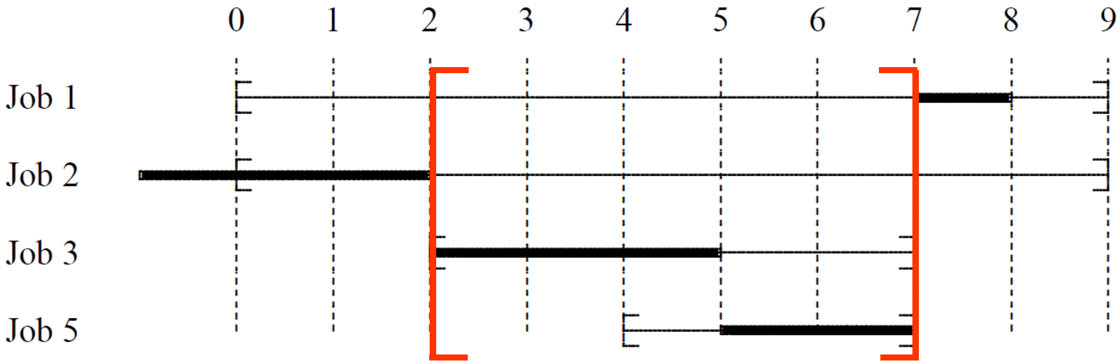
Removing a job from those within an interval only weakens the test

$$L_{J\cup\{k\}} - E_J < p_{J\cup\{k\}}$$

There are a polynomial number of intervals defined by release times and deadlines.

Edge finding for disjunctive scheduling

Key result: We only have to consider sets *J* whose time windows lie within some interval between release times/deadlines.

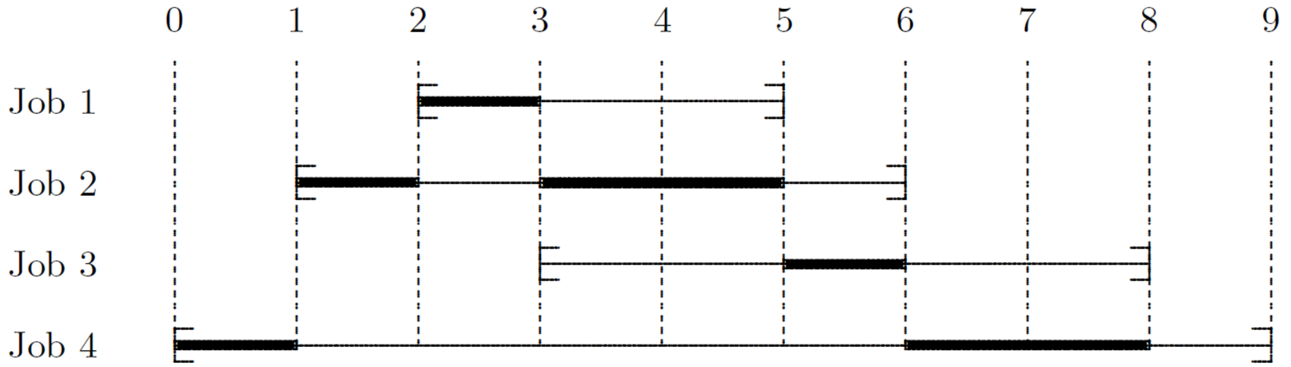


e.g., *J* = {3,5}

Note: Edge finding does not achieve bounds consistency, which is an NP-hard problem.

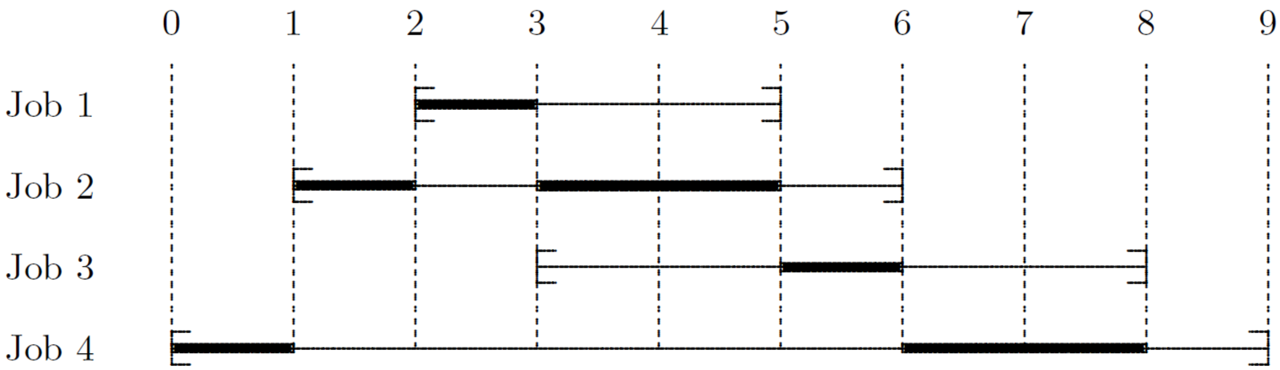
Edge finding for disjunctive scheduling

One $O(n^2)$ algorithm is based on the Jackson pre-emptive schedule (JPS). Using a different example, the JPS is:



Edge finding for disjunctive scheduling

One $O(n^2)$ algorithm is based on the Jackson pre-emptive schedule (JPS). Using a different example, the JPS is:



Jobs unfinished at time E_i in JPS

For each job i

Scan jobs $k \in \boxed{J_i}$ in decreasing order of L_k

Select first k for which $L_k - E_i < p_i + \boxed{\bar{p}_{J_{ik}}}$

Total remaining processing time in JPS of jobs in J_{ik}

Conclude that $i \gg \boxed{J_{ik}}$

Jobs $j \neq i$ in J_i with $L_j \leq L_k$

Update E_i to $\boxed{\text{JPS}(i,k)}$

Latest completion time in JPS of jobs in J_{ik}

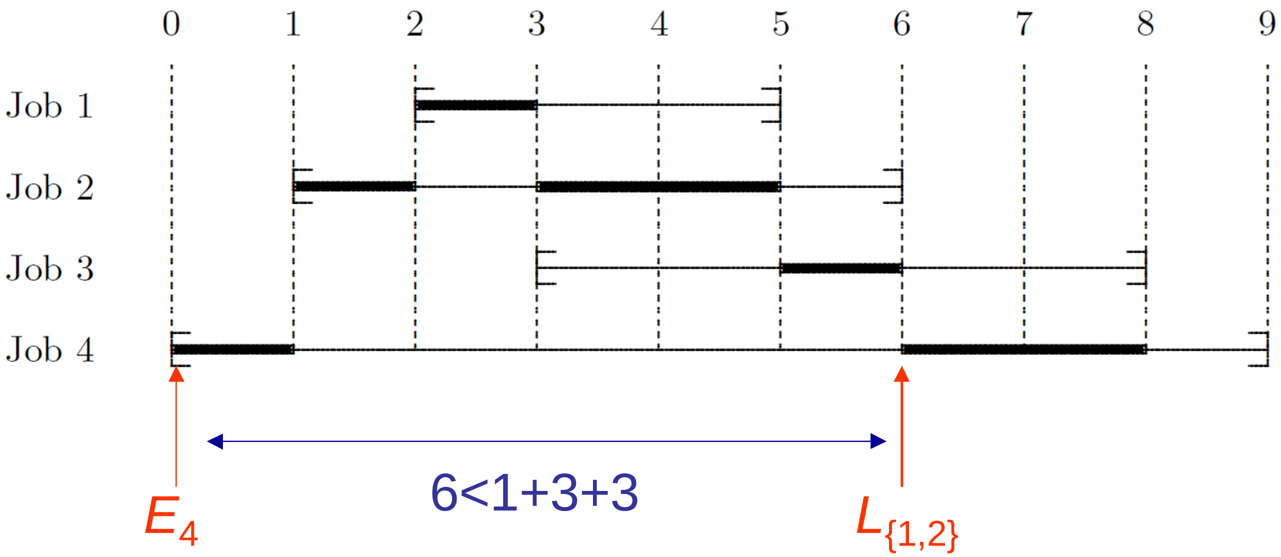
Not-first/not-last rules

We can deduce that job 4 cannot precede jobs 1 and 2:

$$\neg\left(4 \ll \{1,2\}\right)$$

Because if job 4 is first, there is too little time to complete the jobs before the later deadline of jobs 1 and 2:

$$L_{\{1,2\}} - E_4 < p_1 + p_2 + p_4$$



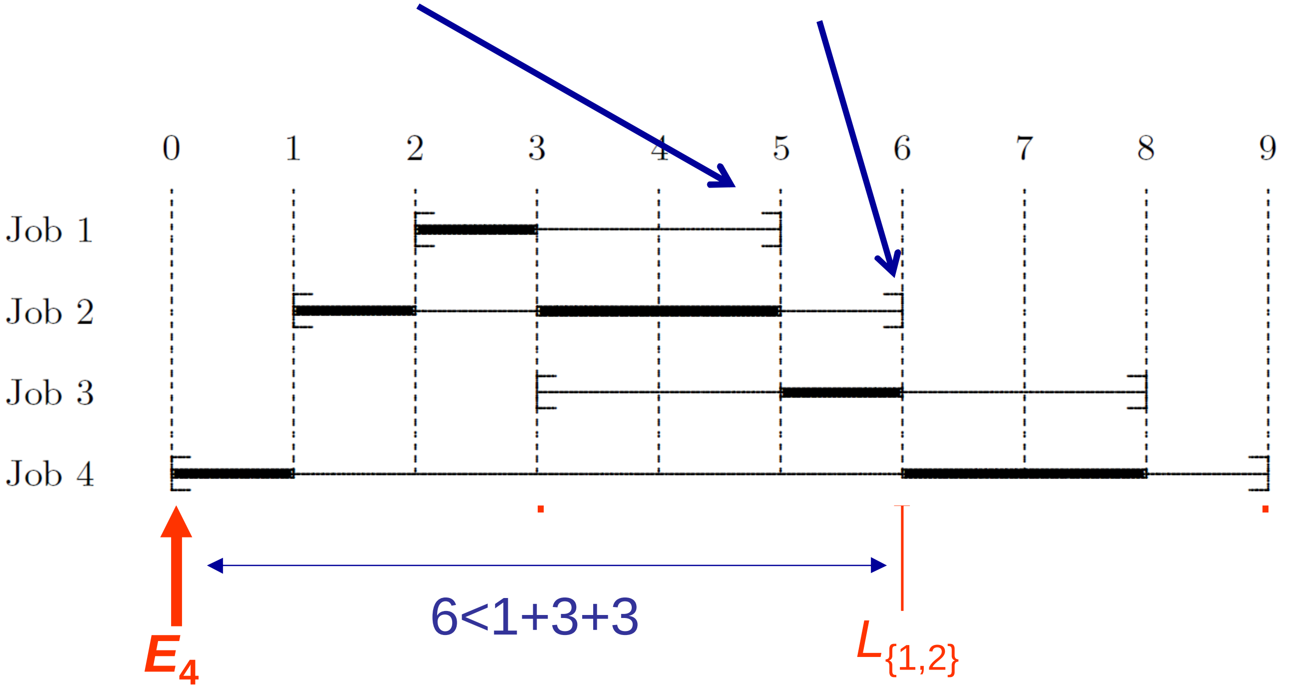
Not-first/not-last rules

We can deduce that job 4 cannot precede jobs 1 and 2:

$$\neg(4 \ll \{1,2\})$$

Now we can tighten the release time of job 4 to minimum of:

$$E_1 + p_1 = 3 \qquad E_2 + p_2 = 4$$



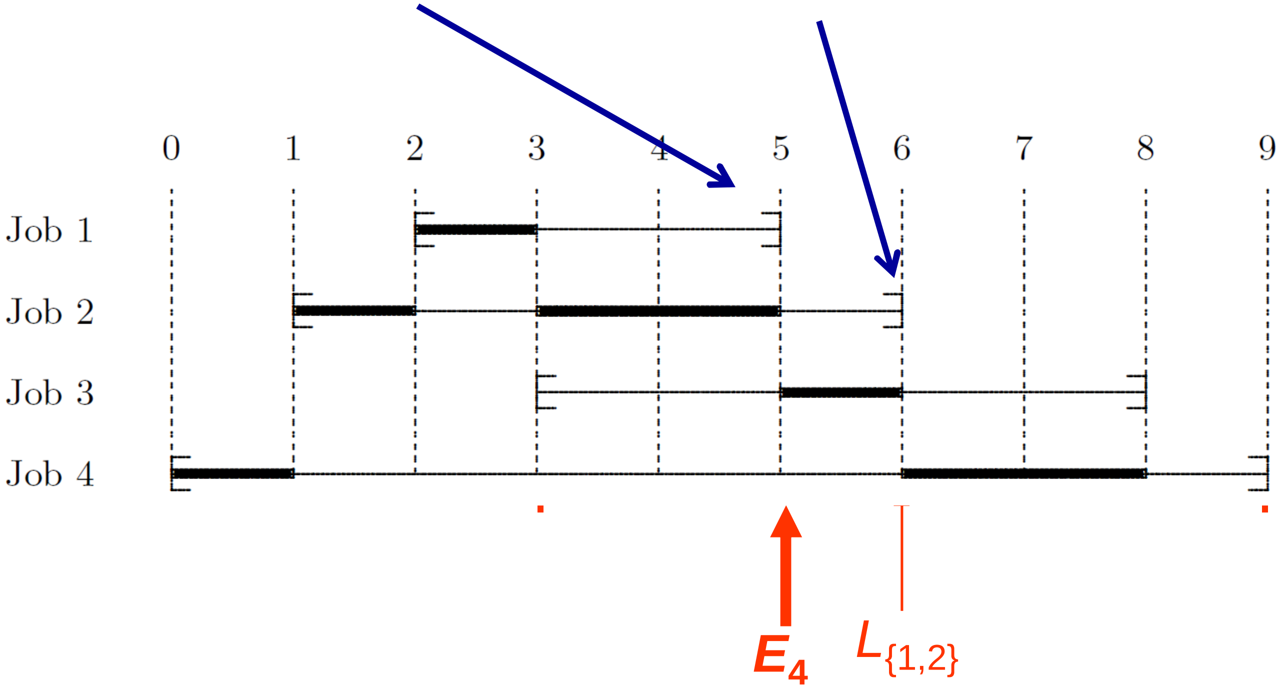
Not-first/not-last rules

We can deduce that job 4 cannot precede jobs 1 and 2:

$$\neg\left(4 \ll \{1,2\}\right)$$

Now we can tighten the release time of job 4 to minimum of:

$E_1 + p_1 = 3$ $\qquad$ $E_2 + p_2 = 4$



Not-first/not-last rules

In general, we can deduce that job *k* cannot precede all the jobs in *J*:

$$\neg\left(k \ll J\right)$$

if there is too little time after release time of job *k* to complete all jobs before the latest deadline in *J*:

$$L_J - E_k < p_J$$

Now we can update E_i to

$$\min_{j \in J}\left\{E_j + p_j\right\}$$

Not-first/not-last rules

In general, we can deduce that job *k* cannot precede all the jobs in *J*:

$$\neg(k \ll J)$$

if there is too little time after release time of job *k* to complete all jobs before the latest deadline in *J*:

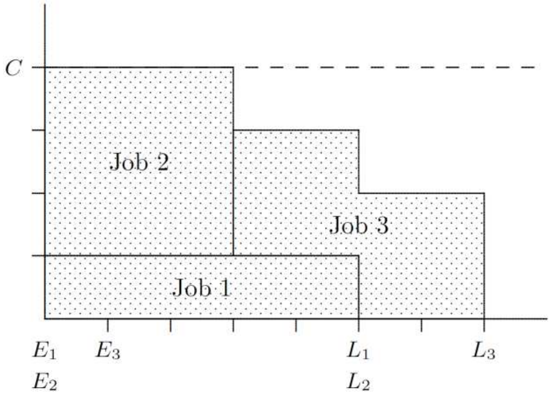
$$L_J - E_k < p_J$$

Now we can update E_i to

$$\min_{j \in J} \left\{ E_j + p_j \right\}$$

There is a symmetric not-last rule.

The rules can be applied in polynomial time, although an efficient algorithm is quite complicated.



Cumulative Scheduling

Edge Finding

Extended Edge Finding

Not-first/Not-last Rules

Energetic Reasoning

Cumulative scheduling

- **Cumulative scheduling** assigns start times to jobs so that total rate of resource consumption is within a limit.
    - A form of **resource-constrained scheduling**
    - Several jobs can run simultaneously
    - **Multiple-machine scheduling** problem is special case
        - Resource consumption rate is 1 for each job, resource limit is number of machines
- Filtering is well developed.
    - Edge finding
    - Extended edge finding
    - Not-first/not-last rules
    - Energetic reasoning

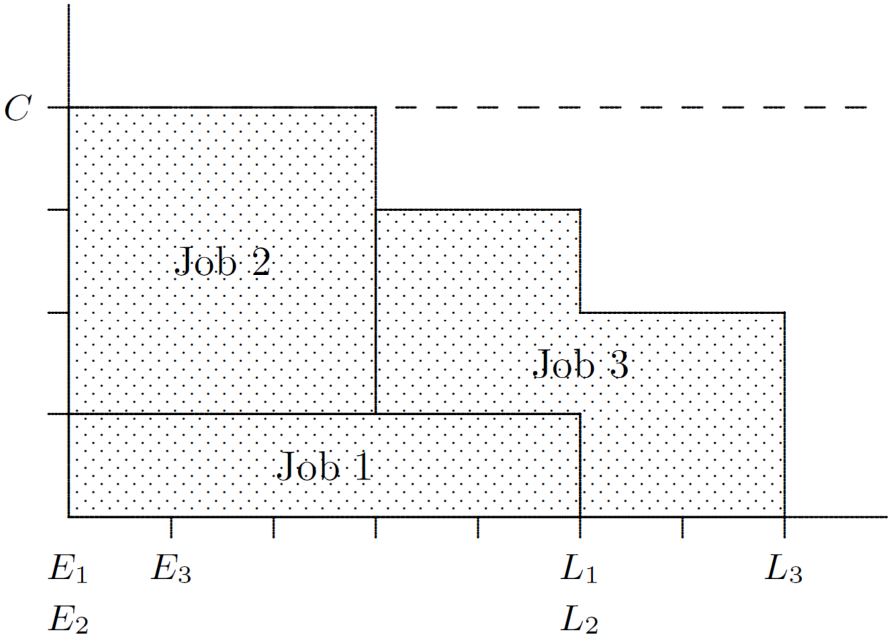
Cumulative scheduling

Consider a cumulative scheduling constraint:

$$\text{cumulative}\left((s_1,s_2,s_3),(p_1,p_2,p_3),(c_1,c_2,c_3),C\right)$$

j	p_j	c_j	E_j	L_j
1	5	1	0	5
2	3	3	0	5
3	4	2	1	7

A feasible solution:

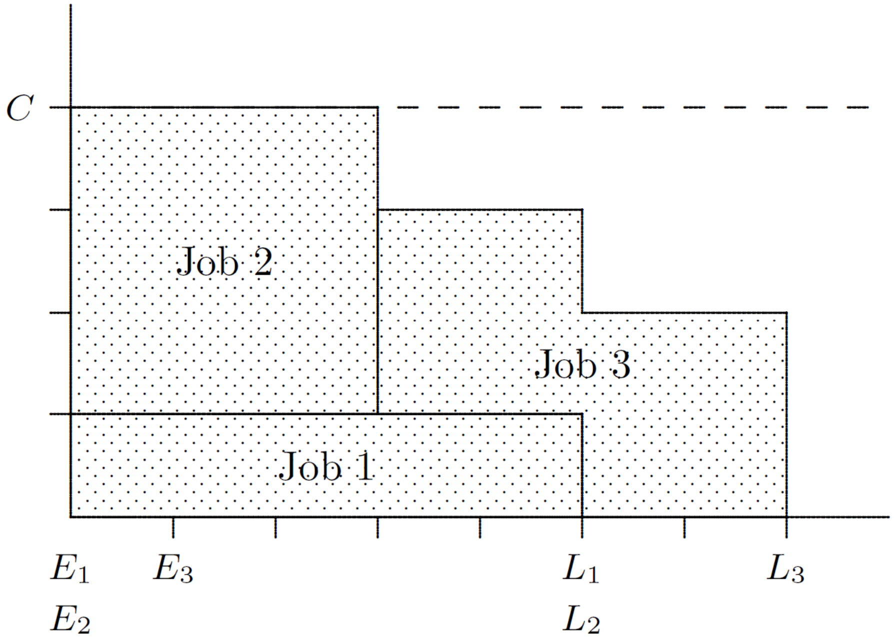


Edge finding for cumulative scheduling

We can deduce that job 3 must finish after the others finish: $3 > \{1,2\}$

Suppose that job 3 is **not** the last to finish.

$$e_3 + e_{\{1,2\}} > C \cdot \left(L_{\{1,2\}} - E_{\{1,2,3\}} \right)$$



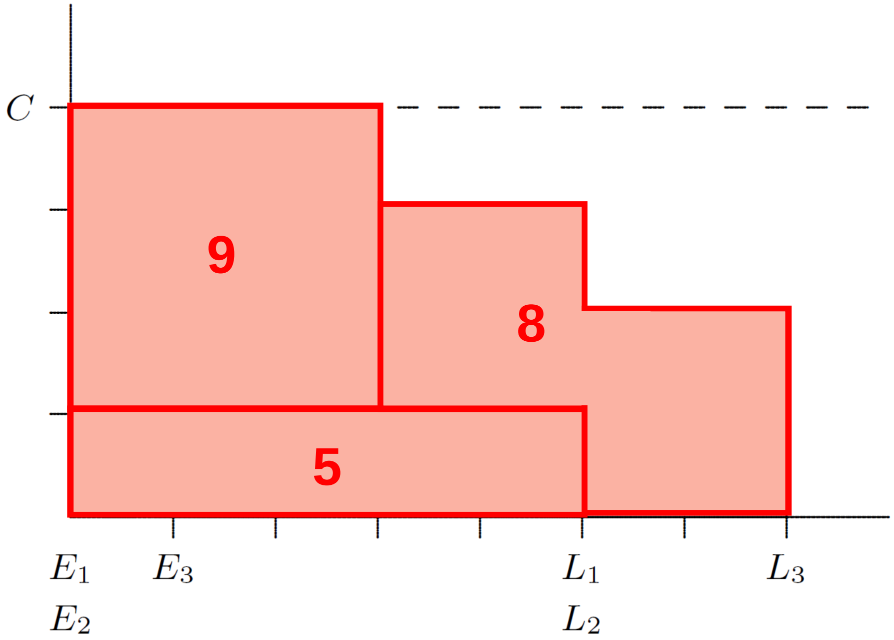
Edge finding for cumulative scheduling

We can deduce that job 3 must finish after the others finish: $3 > \{1,2\}$

Suppose that job 3 is **not** the last to finish.

$$\boxed{e_3 + e_{\{1,2\}}} > C \cdot \left(L_{\{1,2\}} - E_{\{1,2,3\}} \right)$$

Total energy required = 22



Edge finding for cumulative scheduling

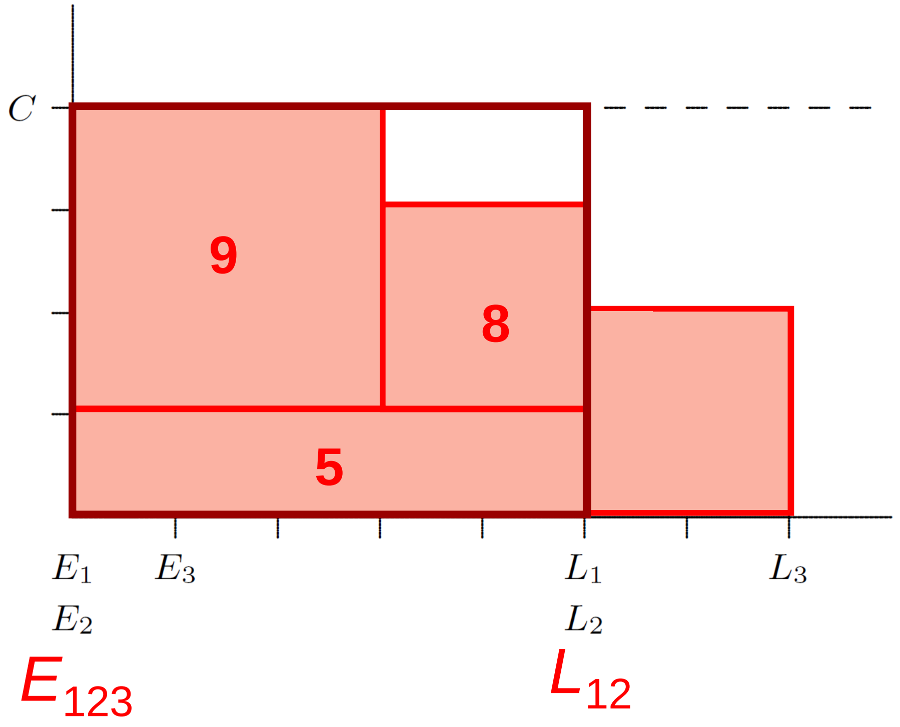
We can deduce that job 3 must finish after the others finish: $3 > \{1,2\}$

Because the total **energy** required exceeds the area between the earliest release time and the later deadline of jobs 1,2:

$$\boxed{e_3 + e_{\{1,2\}}} > \boxed{C \cdot \left(L_{\{1,2\}} - E_{\{1,2,3\}}\right)}$$

Total energy required = 22

Area available = 20



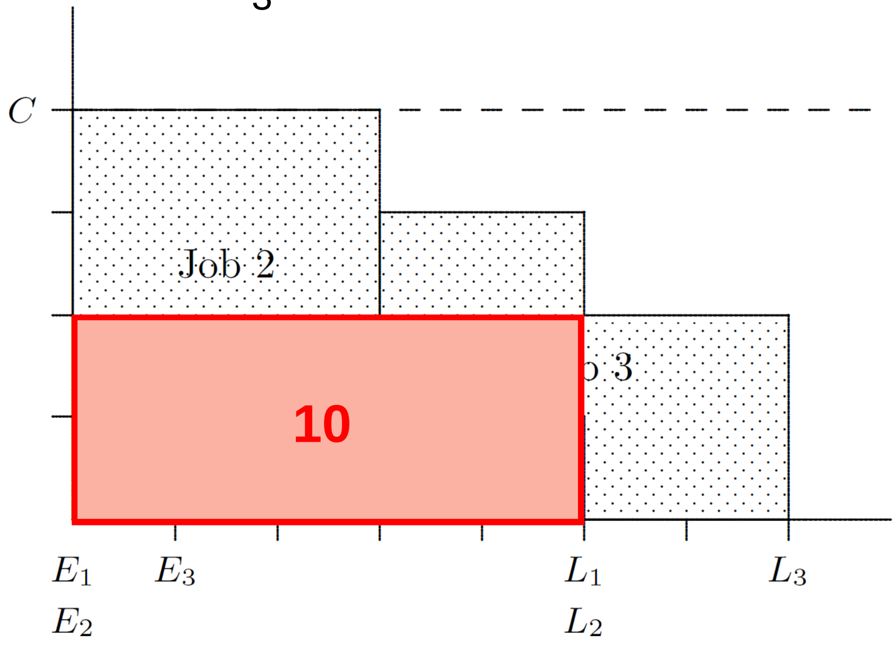
Edge finding for cumulative scheduling

We can deduce that job 3 must finish after the others finish: $3 > \{1,2\}$

We can update the release time of job 3 to

$$E_{\{1,2\}} + \frac{e_J - \boxed{(C - c_3)(L_{\{1,2\}} - E_{\{1,2\}})}}{c_3}$$

Energy available for jobs 1,2 if space is left for job 3 to start anytime = 10



Edge finding for cumulative scheduling

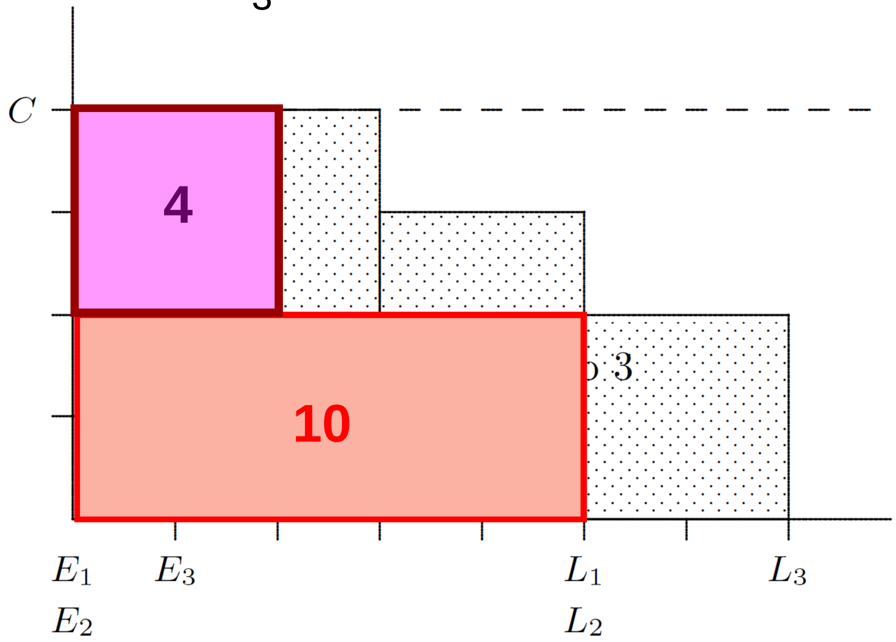
We can deduce that job 3 must finish after the others finish: $3 > \{1,2\}$

We can update the release time of job 3 to

$$E_{\{1,2\}} + \frac{\boxed{e_J - (C - c_3)(L_{\{1,2\}} - E_{\{1,2\}})}}{c_3}$$

Energy available for jobs 1,2 if space is left for job 3 to start anytime = 10

Excess energy required by jobs 1,2 = 4



Edge finding for cumulative scheduling

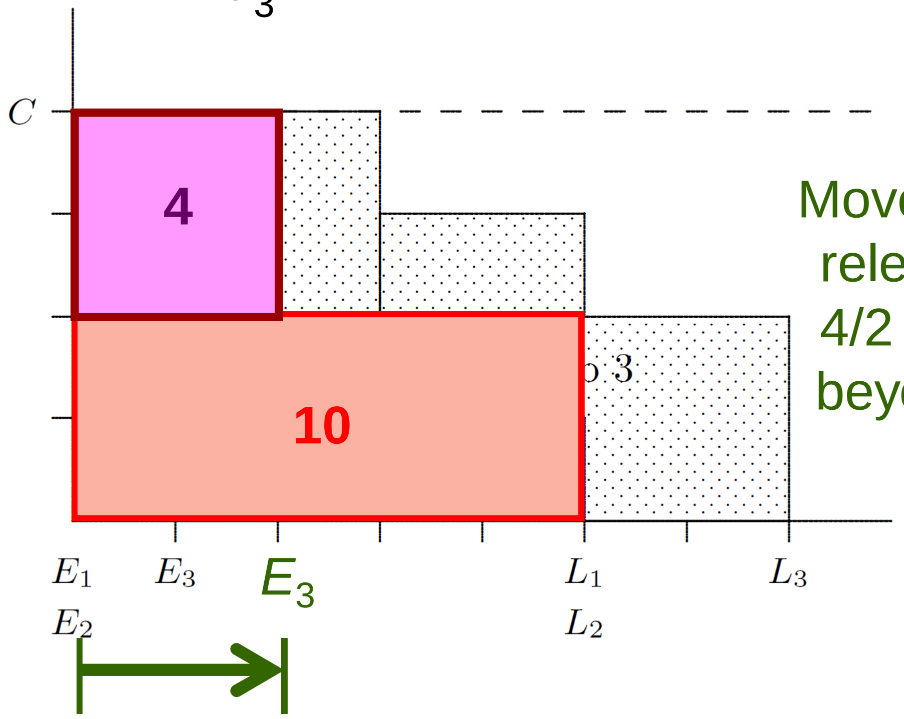
We can deduce that job 3 must finish after the others finish: $3 > \{1,2\}$

We can update the release time of job 3 to

$$E_{\{1,2\}} + \frac{e_{\{12\}} - (C - c_3)(L_{\{1,2\}} - E_{\{1,2\}})}{c_3}$$

Energy available for jobs 1,2 if space is left for job 3 to start anytime = 10

Excess energy required by jobs 1,2 = 4



Move up job 3 release time 4/2 = 2 units beyond $E_{\{1,2\}}$

Edge finding for cumulative scheduling

In general, if $e_{J\cup\{k\}} > C\cdot\left(L_J - E_{J\cup\{k\}}\right)$

then $k > J$, and update E_k to

$$\max_{\substack{J'\subset J \\ e_{J'}-(C-c_k)(L_{J'}-E_{J'})>0}} \left\{ E_{J'} + \frac{e_{J'}-(C-c_k)(L_{J'}-E_{J'})}{c_k} \right\}$$

In general, if $e_{J\cup\{k\}} > C\cdot\left(L_{J\cup\{k\}} - E_J\right)$

then $k < J$, and update L_k to

$$\min_{\substack{J'\subset J \\ e_{J'}-(C-c_k)(L_{J'}-E_{J'})>0}} \left\{ L_{J'} - \frac{e_{J'}-(C-c_k)(L_{J'}-E_{J'})}{c_k} \right\}$$

Edge finding for cumulative scheduling

There is an $O(n^2)$ algorithm that finds all applications of the edge finding rules.

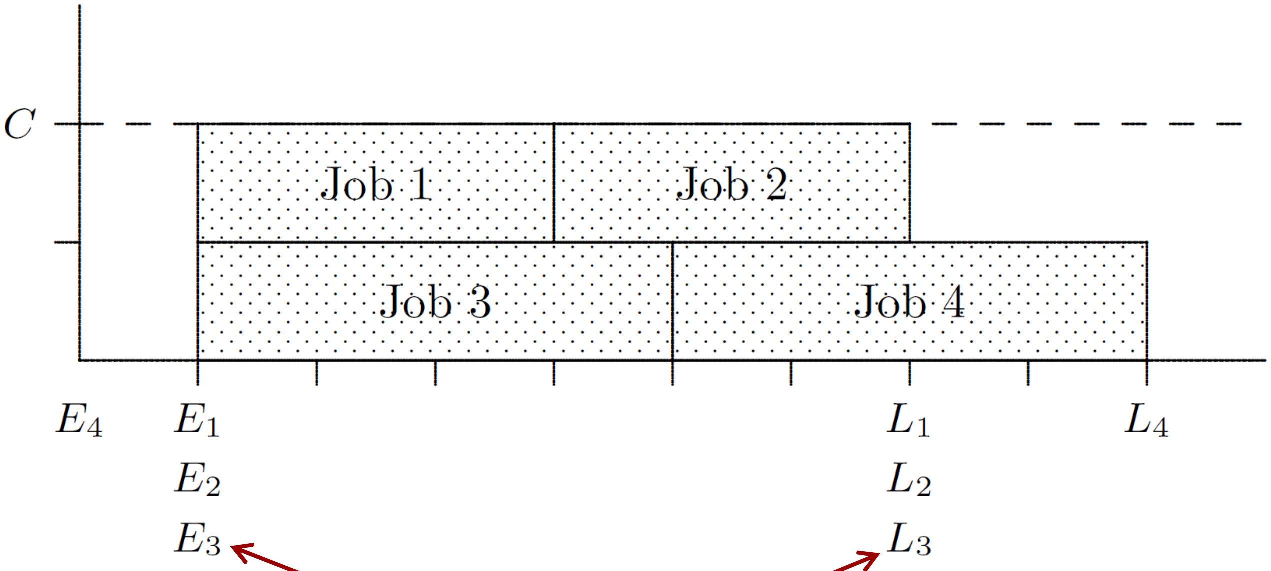
Extended edge finding

Useful when a job with an early release time must finish after other jobs.

Ordinary edge finding may not detect this situation.

Extended edge finding

Consider the problem:

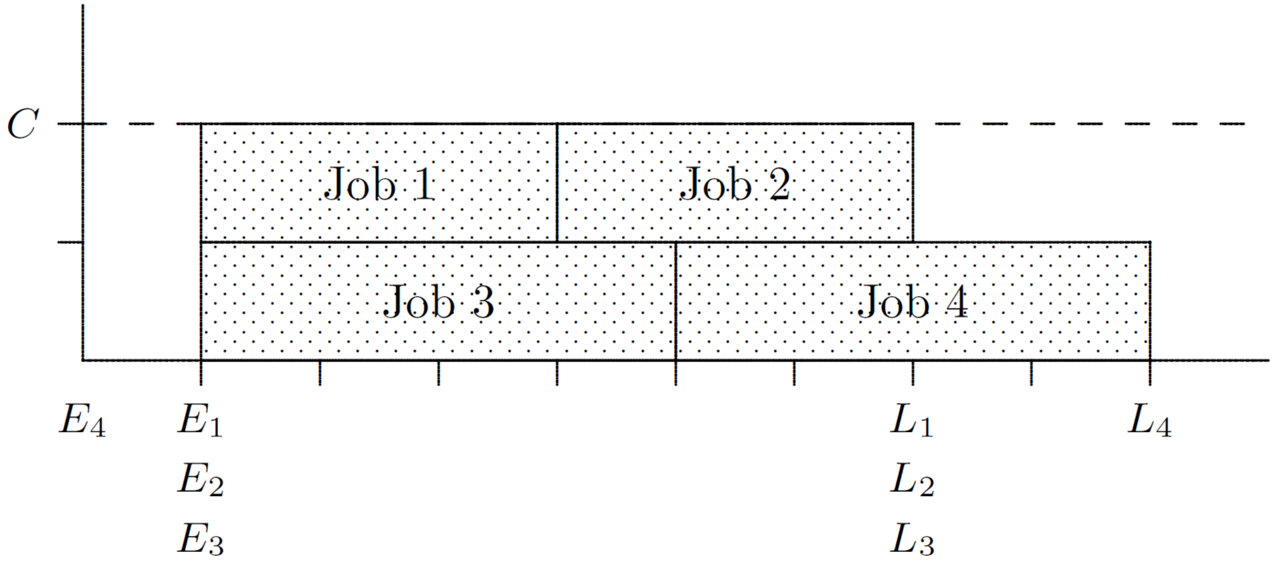


Time window

A feasible solution is shown.

Extended edge finding

Consider the problem:

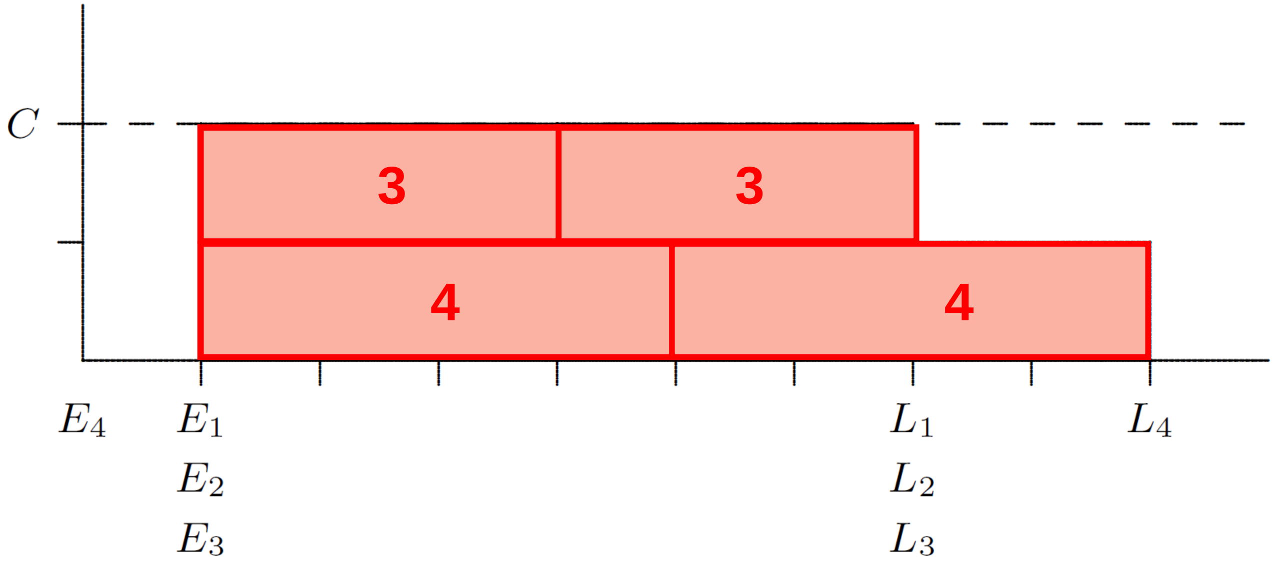


Job 4 must finish after the others: 4 > {1,2,3}.

Extended edge finding

Consider the problem:

Total energy required = 14



Job 4 must finish after the others: 4 > {1,2,3}.

Edge finding does not deduce this:

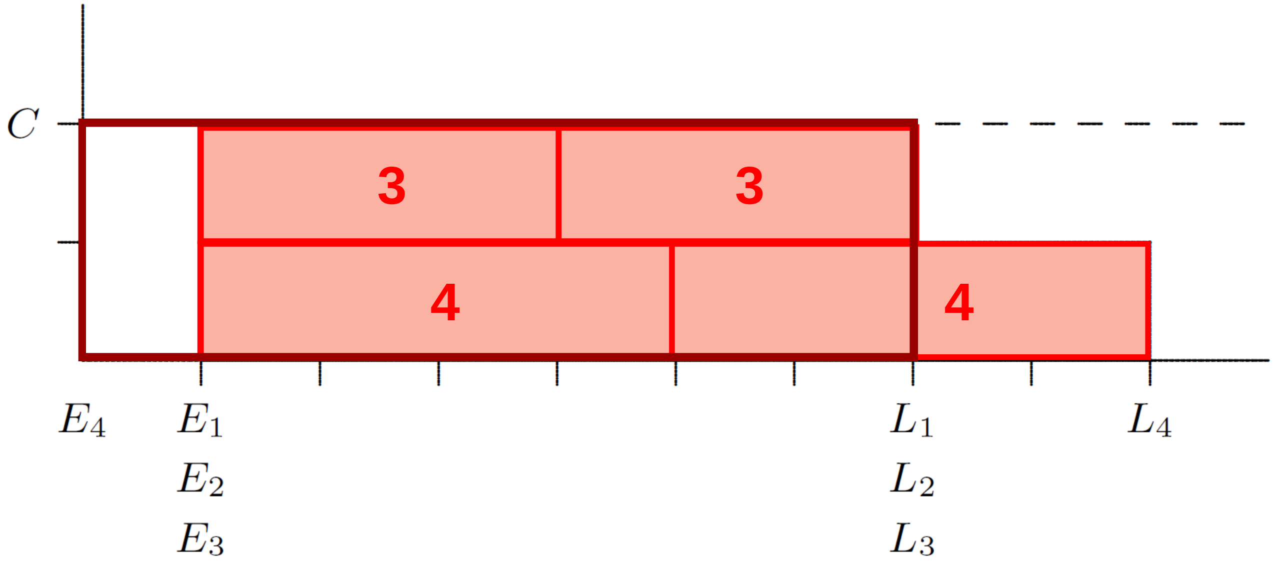
$$e_4 + e_{\{123\}} \leq C \cdot \left(L_{\{123\}} - E_{\{1234\}} \right)$$

Extended edge finding

Consider the problem:

Total energy required = 14

Area available = 14



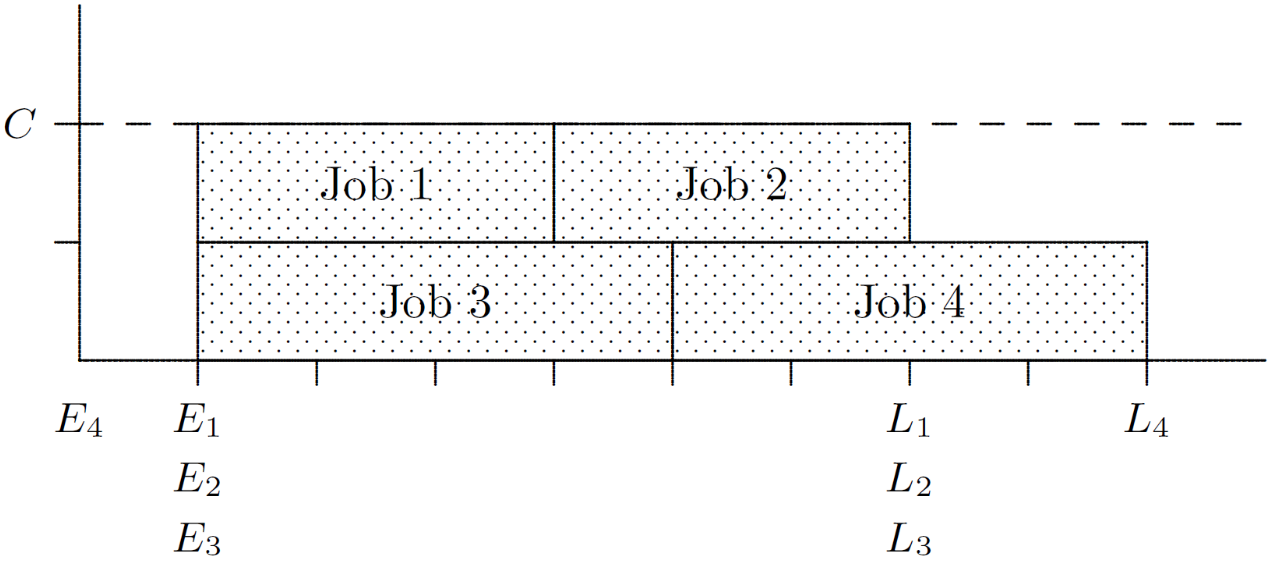
Job 4 must finish after the others: 4 > {1,2,3}.

Edge finding does not deduce this:

$$e_4 + e_{\{123\}} \leq C \cdot \left(L_{\{123\}} - E_{\{1234\}} \right)$$

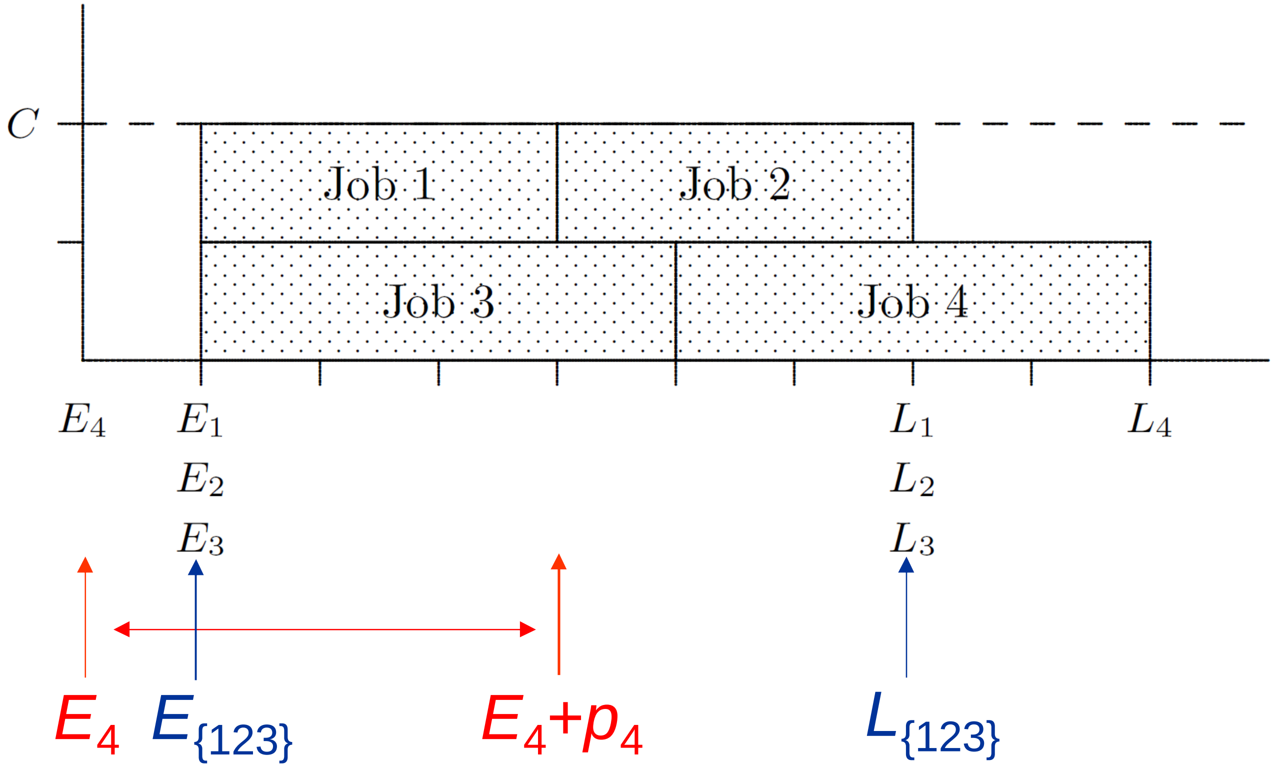
Extended edge finding

Suppose that job 4 does **not** finish last. We will prove a contradiction.



Extended edge finding

Note that job 4 has an earlier release time than the other jobs but can’t finish before the earliest release time of the other jobs:

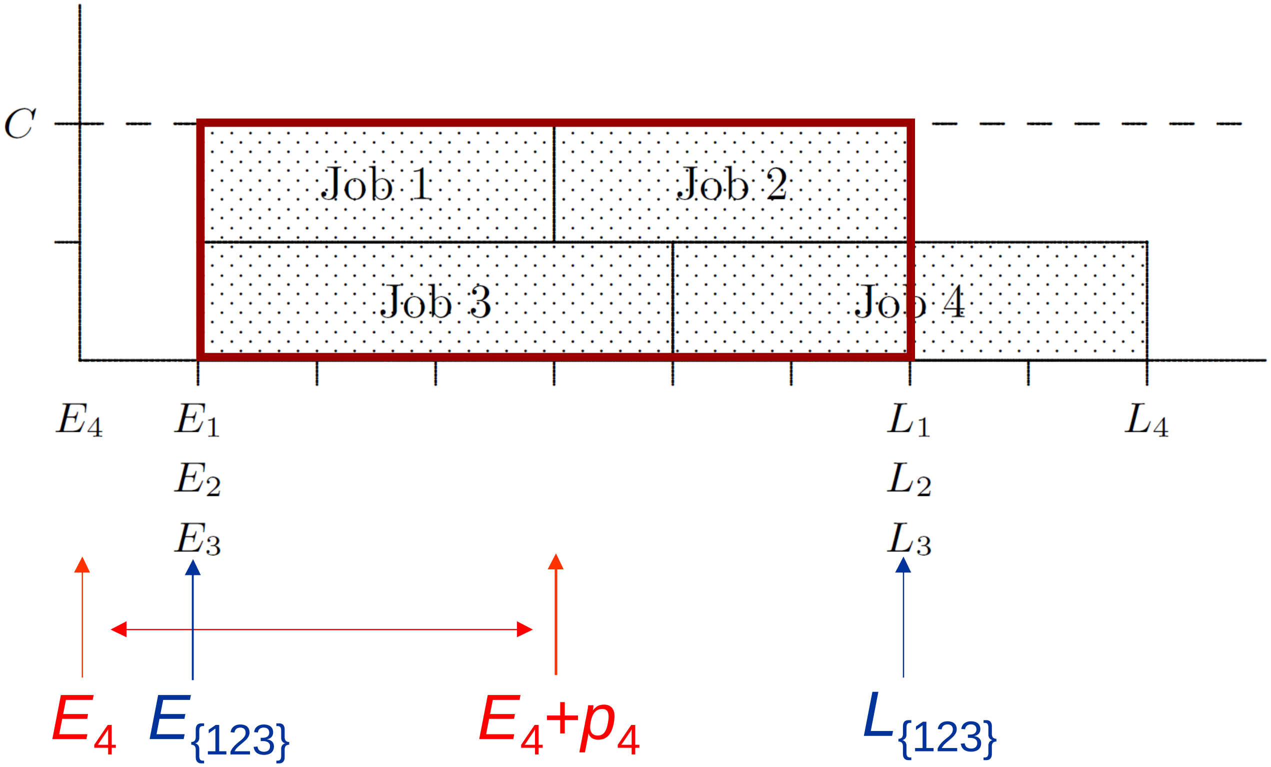


$$E_4 \le E_{\{123\}} < E_4 + p_4$$

Extended edge finding

This area…

Area available = 12

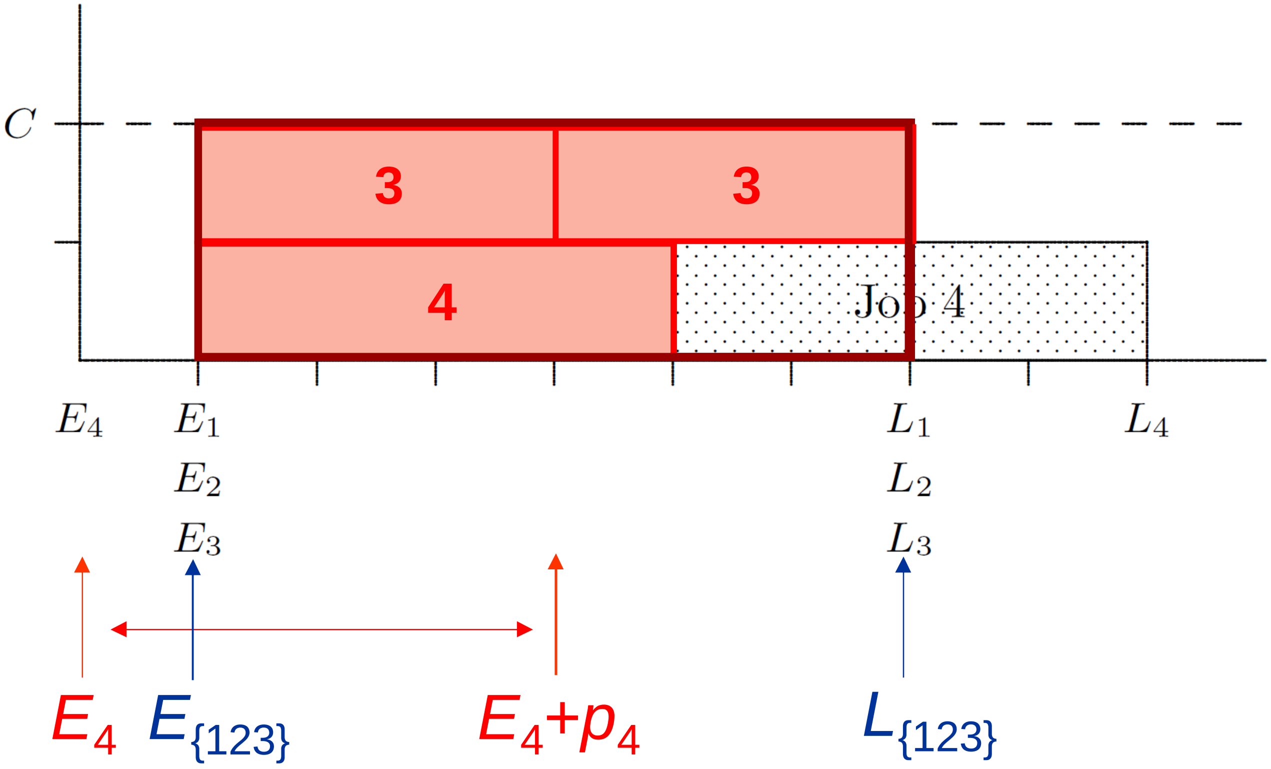


Extended edge finding

This area must contain jobs 1,2,3…

Total energy required = 10 +

Area available = 12

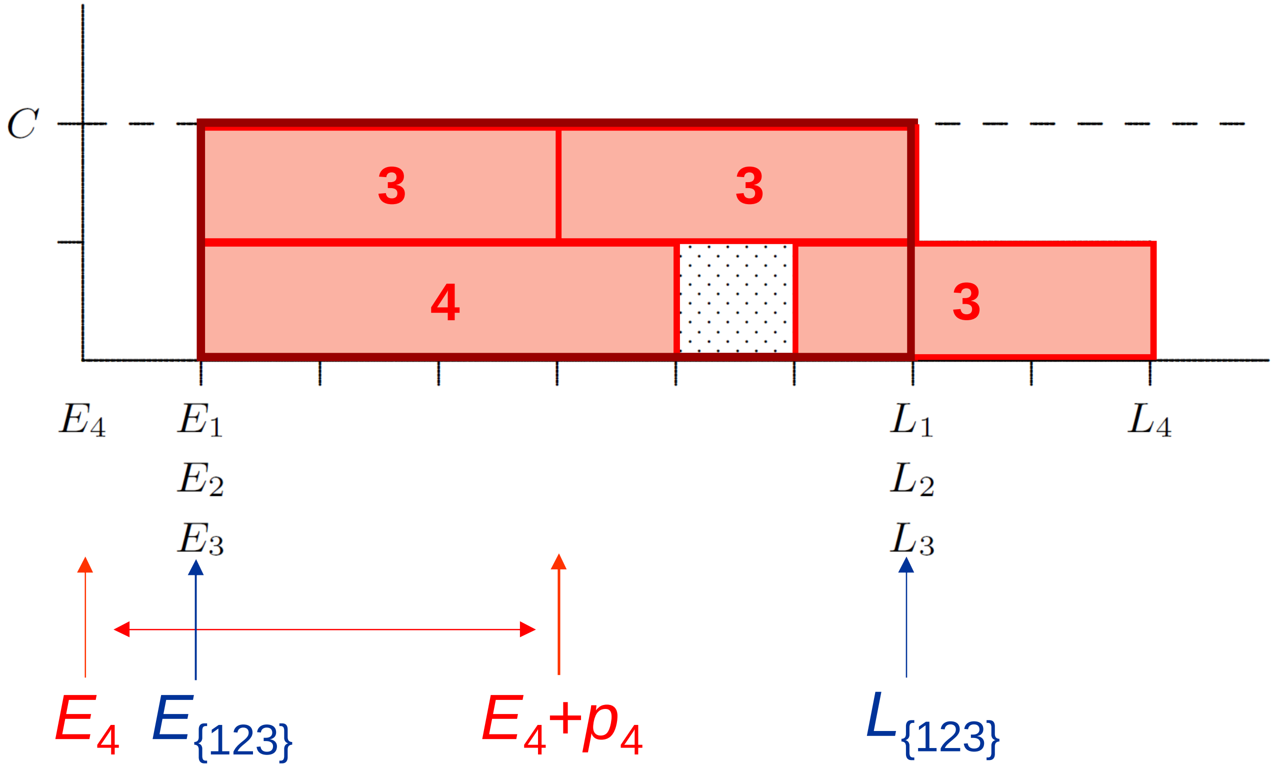


Extended edge finding

This area must contain jobs 1,2,3 plus portion of job 4 that must run after $E_{\{123\}}$:

Total energy required = 10 + 3

Area available = 12

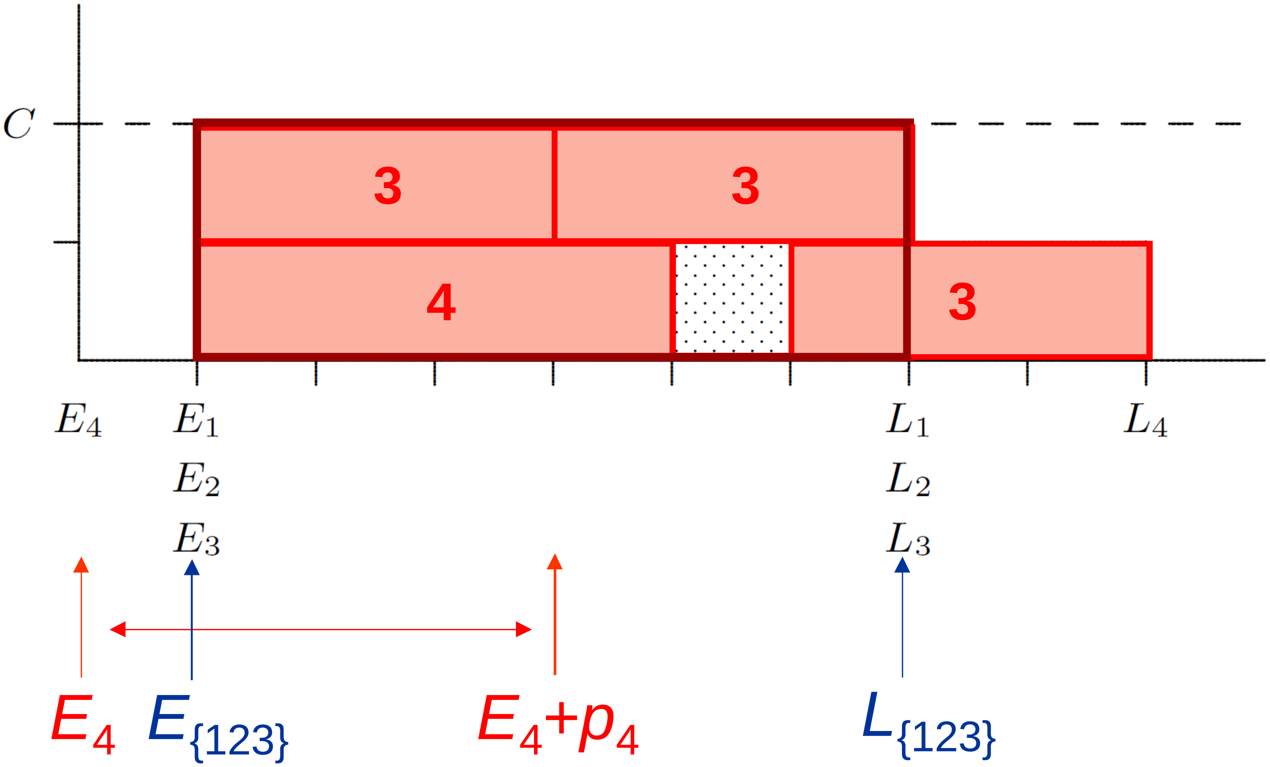


Extended edge finding

This area must contain jobs 1,2,3 plus portion of job 4 that must run after $E_{\{123\}}$:

Total energy required = 10 + 3

Area available = 12



E_4 $E_{\{123\}}$ E_4+p_4 $L_{\{123\}}$

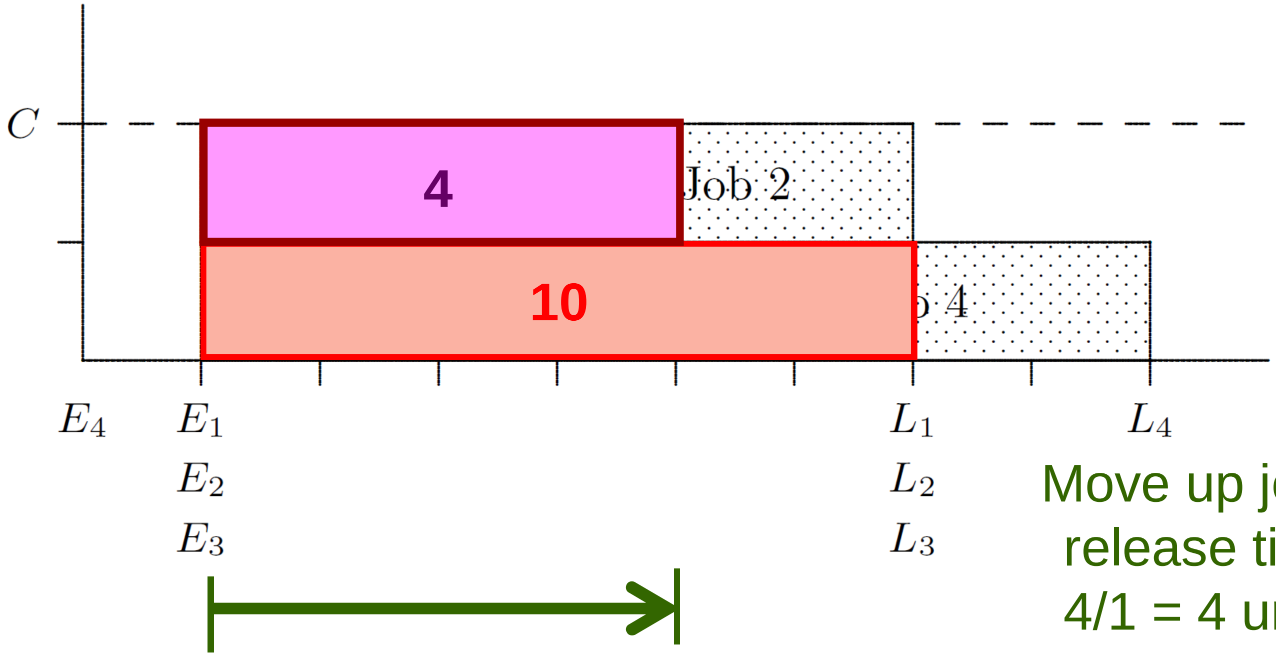
$$e_{\{123\}} + c_4\left(E_4 + p_4 - E_{\{123\}}\right) > 2\cdot\left(L_{\{123\}} - E_{\{123\}}\right)$$

Extended edge finding

We conclude that job 4 finishes after 1,2,3 finish: 4 > {123}.
Update bound E_4 as before.

Energy for jobs 1,2,3 if space is left for job 4 = 10

Excess energy required by jobs 1,2,3 = 4



Move up job 4 release time 4/1 = 4 units beyond $E_{\{123\}}$

$$E_{\{123\}} + \frac{e_{\{123\}} - (C - c_4)(L_{\{123\}} - E_{\{123\}})}{c_4}$$

Edge finding for cumulative scheduling

In general, if $E_k \le E_J < E_k + p_k$

and $e_J + c_k\left(E_k + p_k - E_J\right) > C \cdot \left(L_J - E_J\right)$,

then $i > J$, and update E_k to

$$\max_{\substack{J' \subset J \\ e_{J'} - (C - c_k)(L_{J'} - E_{J'}) > 0}} \left\{ E_{J'} + \frac{e_{J'} - (C - c_k)(L_{J'} - E_{J'})}{c_k} \right\}$$

Similarly for proving $k < J$.

Not-first/not-last rules

These rules deduce

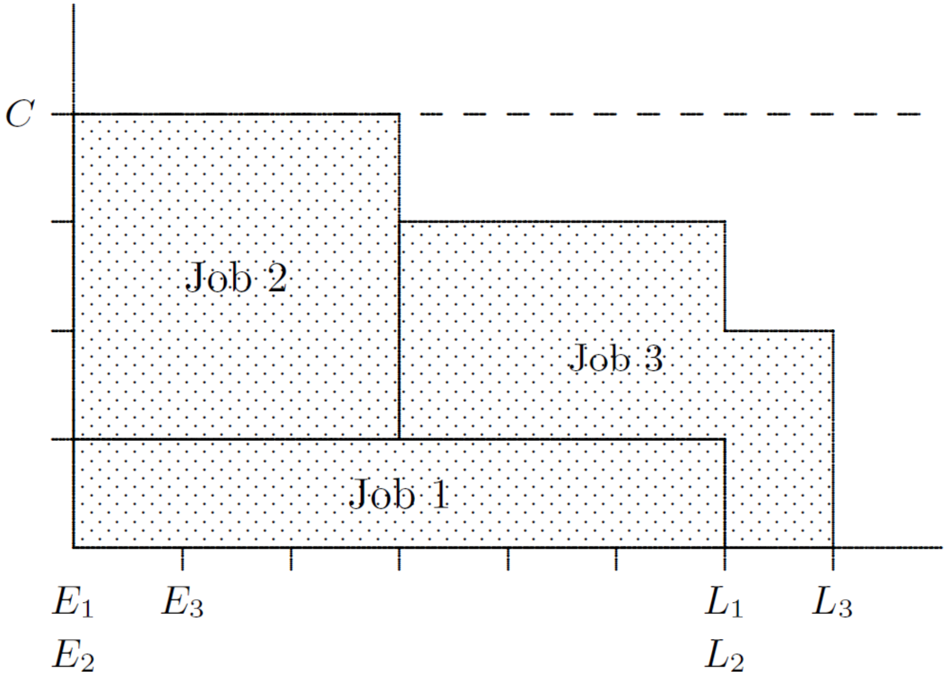
$$\neg\left(k \ll J\right)$$

as in disjunctive scheduling. That is, job *k* starts after some job in *J* finishes.

A feasible solution is shown.

Not-first/not-last rules

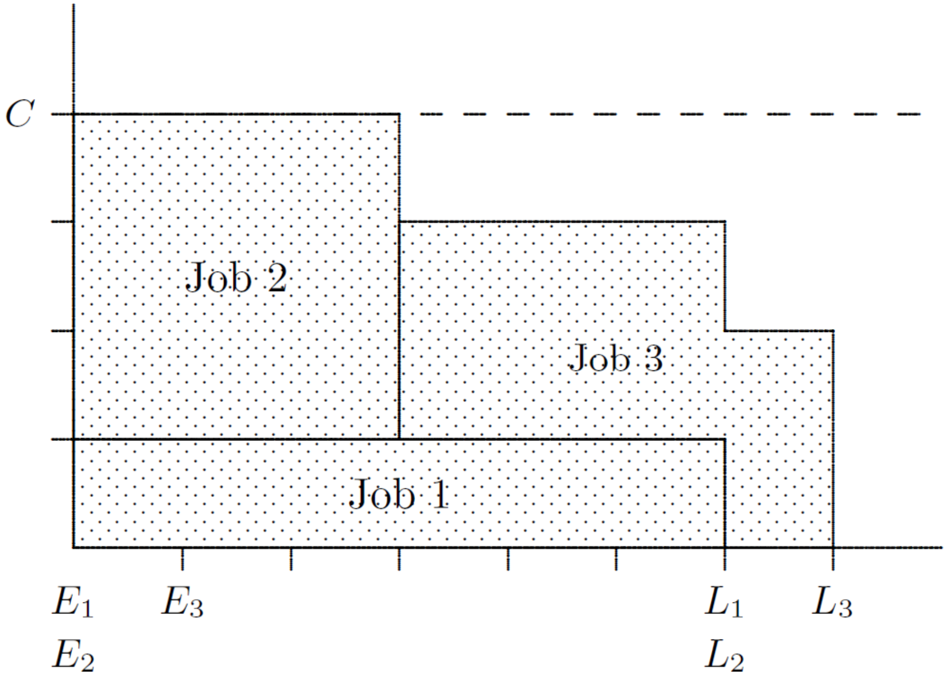
Consider the problem:



A feasible solution is shown.

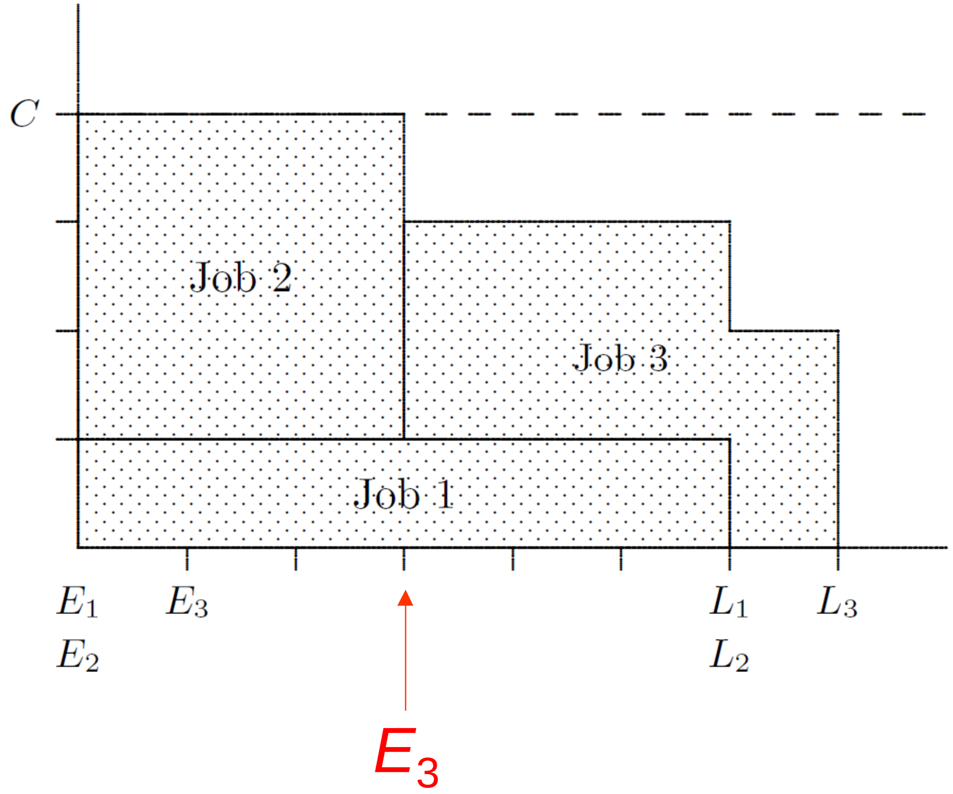
Not-first/not-last rules

Job 3 must start after some job in {1,2} finishes (namely, job 2).



Not-first/not-last rules

Job 3 must start after some job in {1,2} finishes (namely, job 2).

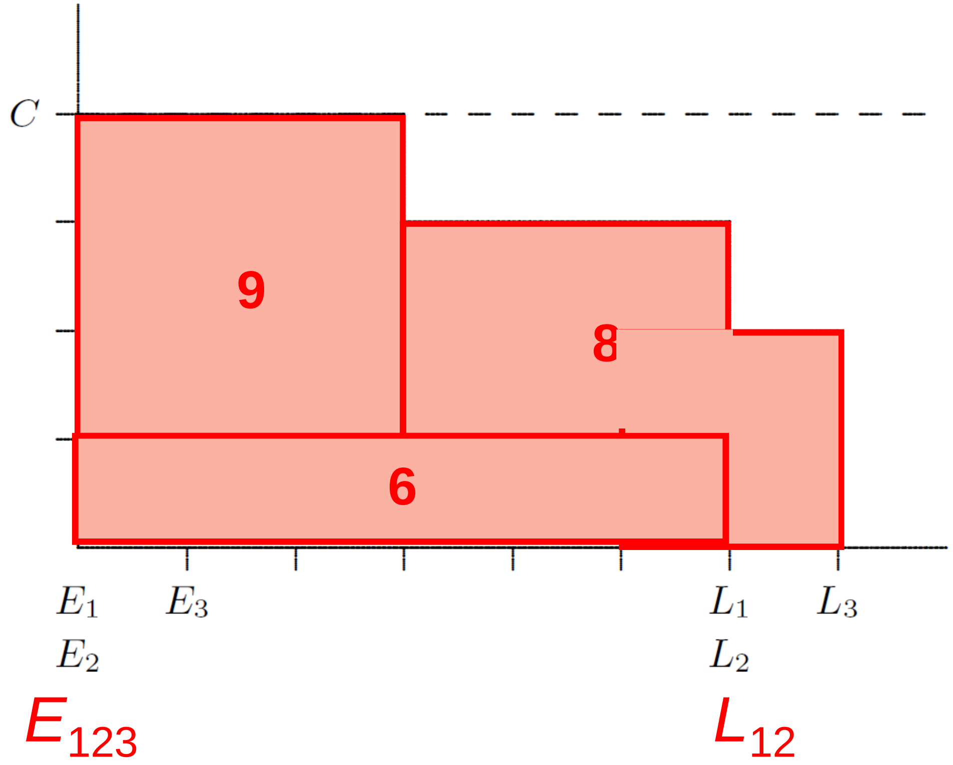


So E_3 can be updated to 3.

Not-first/not-last rules

Let's first try to update E_3 using edge finding.

Total energy required = 23

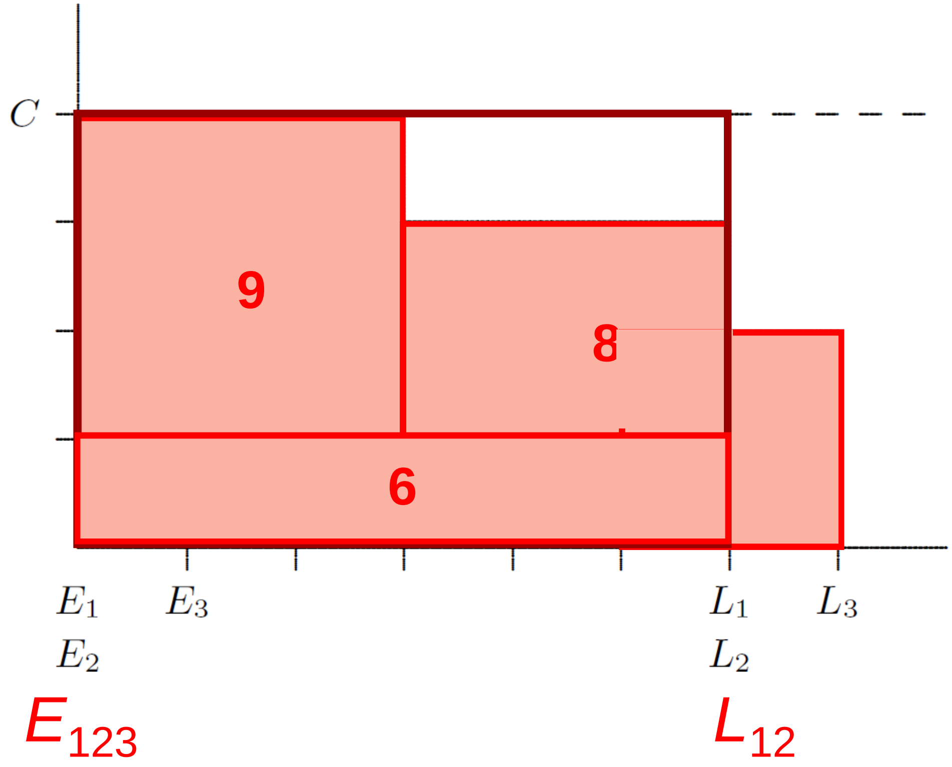


Not-first/not-last rules

Let’s first try to update E_3 using edge finding.

Total energy required = 23

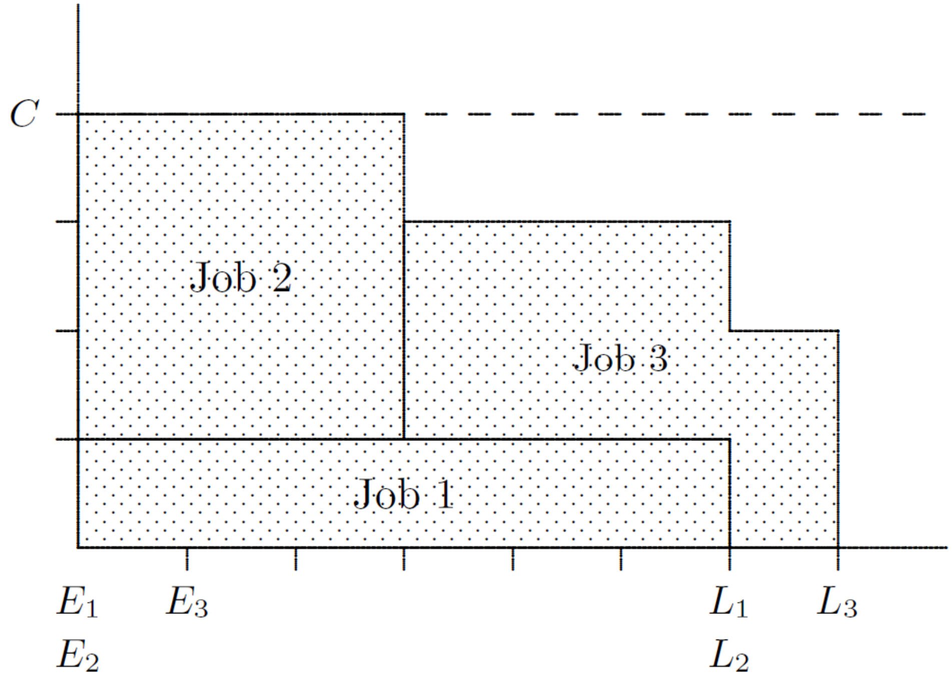
Area available = 24



Cannot prove 3 > {1,2}.

Not-first/not-last rules

Cannot apply extended edge finding to show 3 > {1,2}

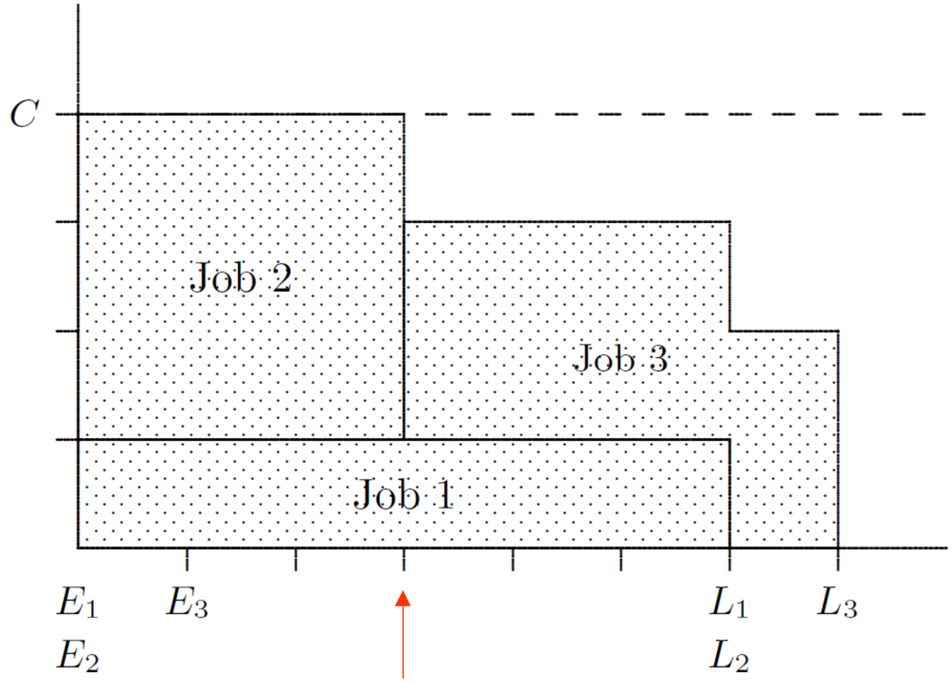


E_{12} E_3

We don’t have $E_3 \leq E_{\{12\}}$

Not-first/not-last rules

So we use not-first/not-last rule.



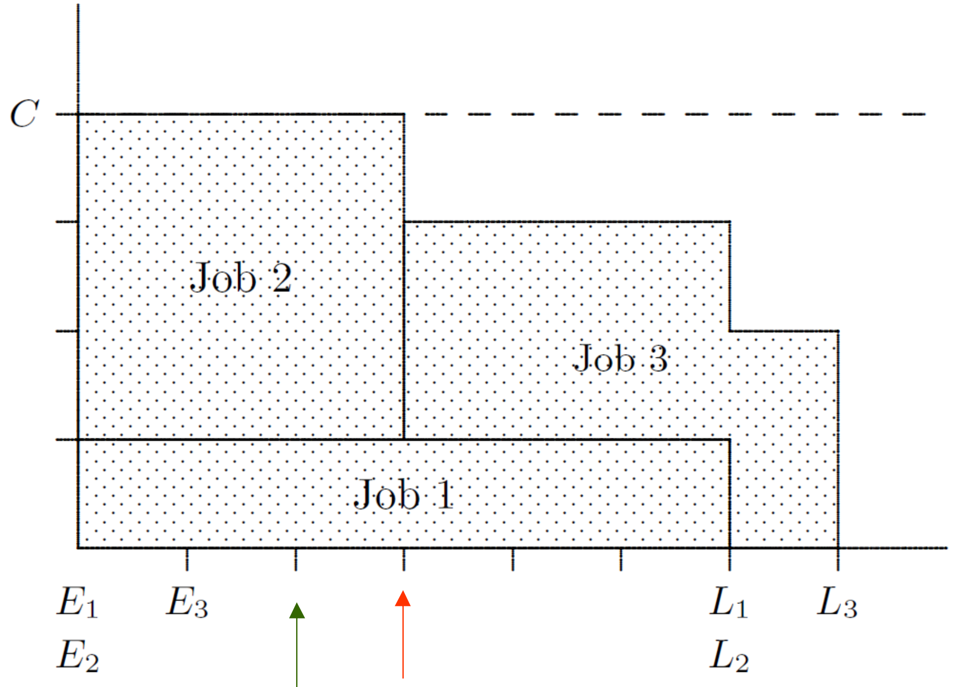
E_{12} E_3 F_{12}

Note that $E_{\{12\}} \leq E_3 < F_{\{12\}}$

Minimum earliest finish time
$= \min \{E_1 + p_1, E_2 + p_2\}$

Not-first/not-last rules

Now suppose that job 3 starts at some time *t* before F_{12}. We will derive a contradiction.



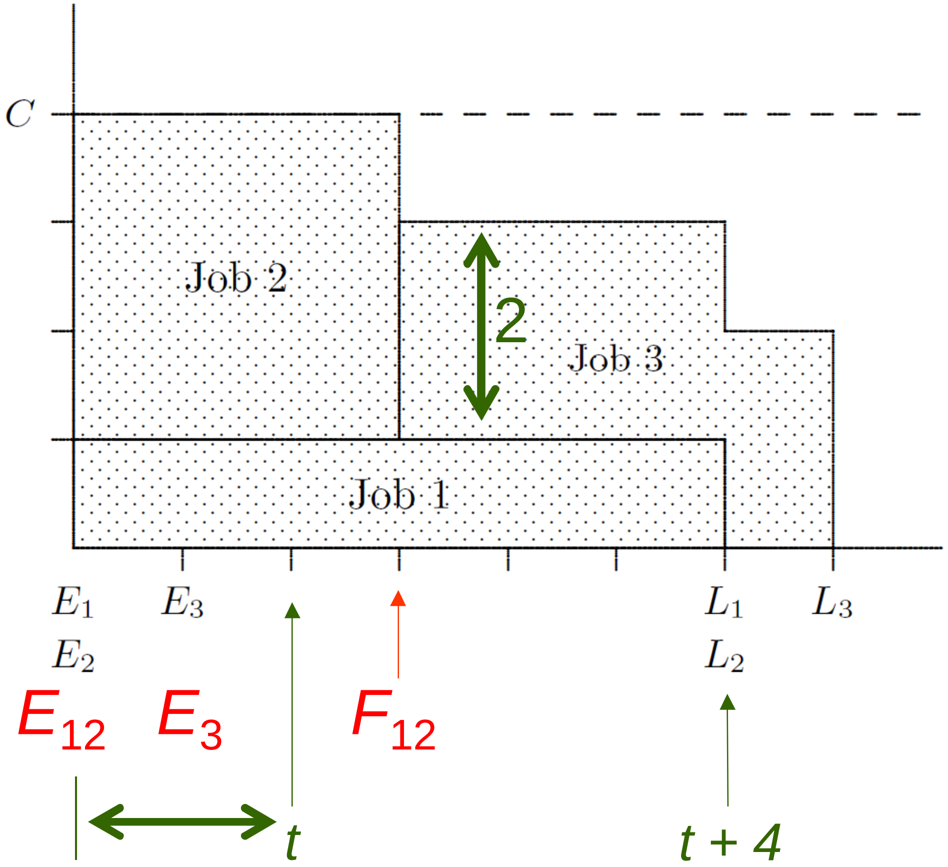
E_{12} E_3 F_{12}

t *t* + *4*

Start and end of job 3

Not-first/not-last rules

Now suppose that job 3 starts at some time *t* before F_{12}. We will derive a contradiction.



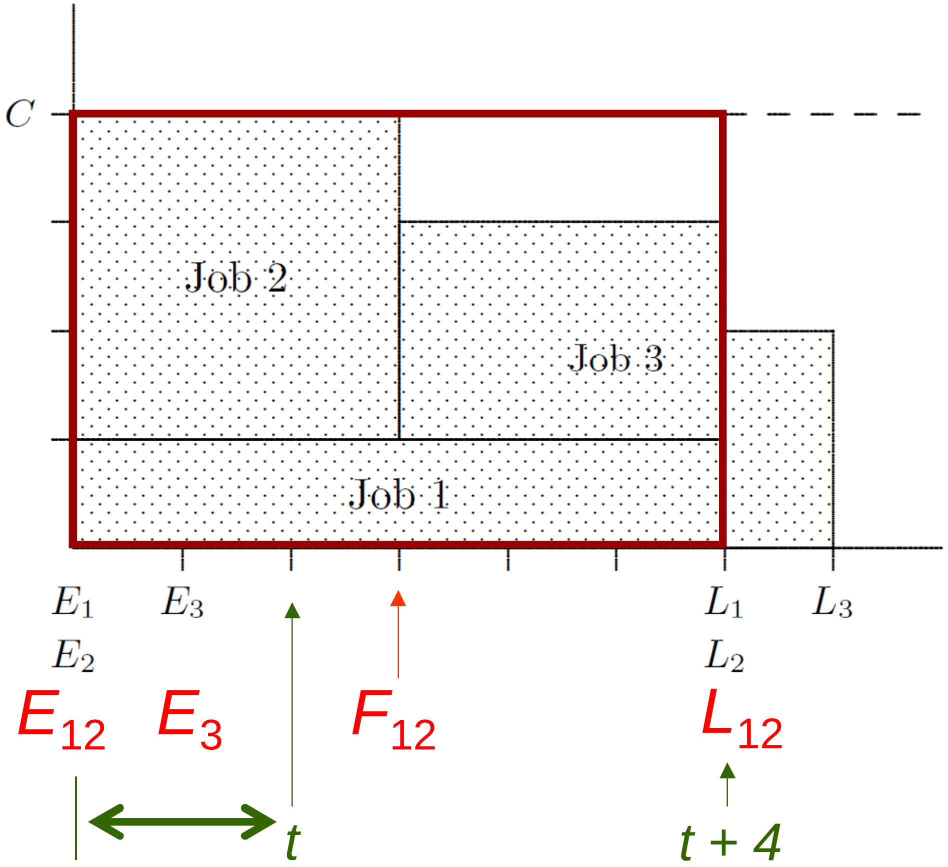
Resource consumption 2 of job 3 cannot be used during this period

Not-first/not-last rules

Now suppose that job 3 starts at some time *t* before F_{12}. We will derive a contradiction.

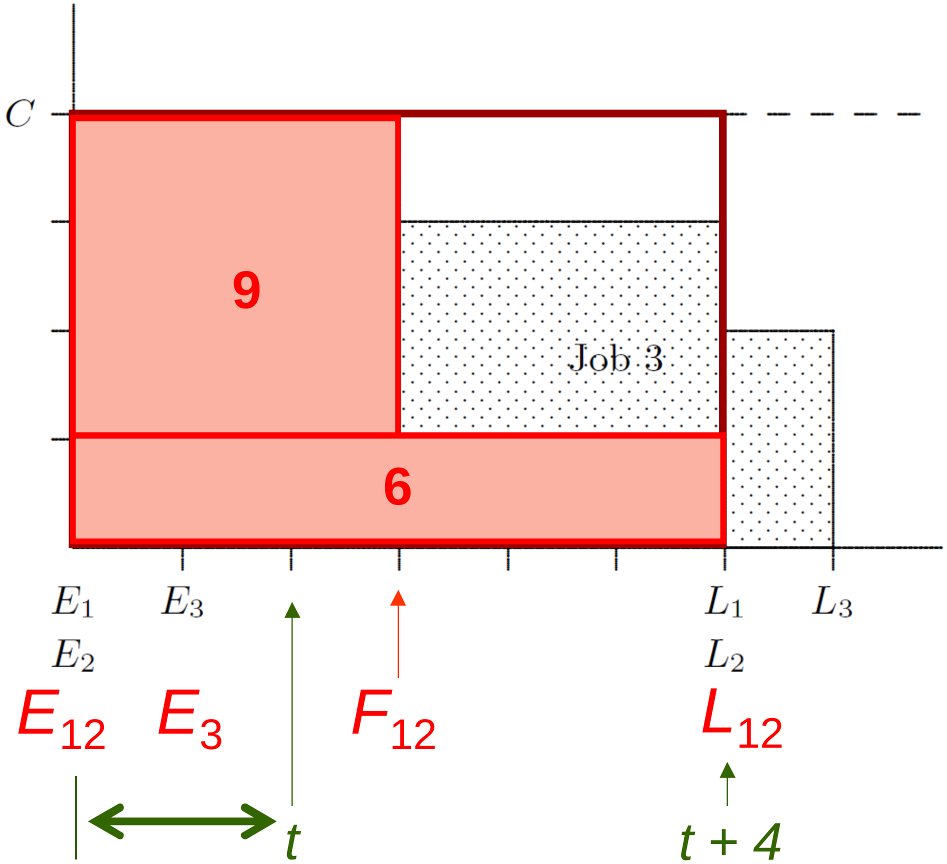
Total energy required between E_{12} and L_{12} is…

Resource consumption 2 of job 3 cannot be used during this period



Not-first/not-last rules

Now suppose that job 3 starts at some time *t* before F_{12}. We will derive a contradiction.



Total energy required between E_{12} and L_{12} is 6 + 9 + …

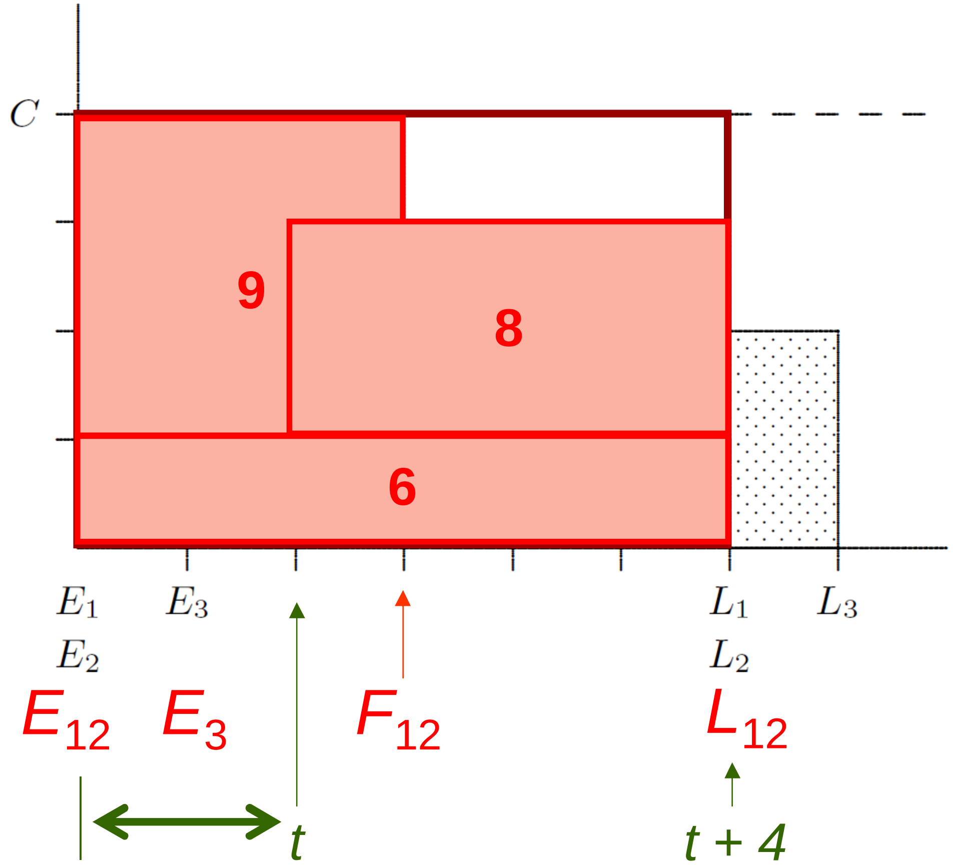
Resource consumption 2 of job 3 cannot be used during this period

$e_{\{12\}} +$

Not-first/not-last rules

Now suppose that job 3 starts at some time *t* before F_{12}. We will derive a contradiction.

Total energy
required between
E_{12} and L_{12} is
6 + 9 + 2·(min{t + 4,6} – *t*)
+ …



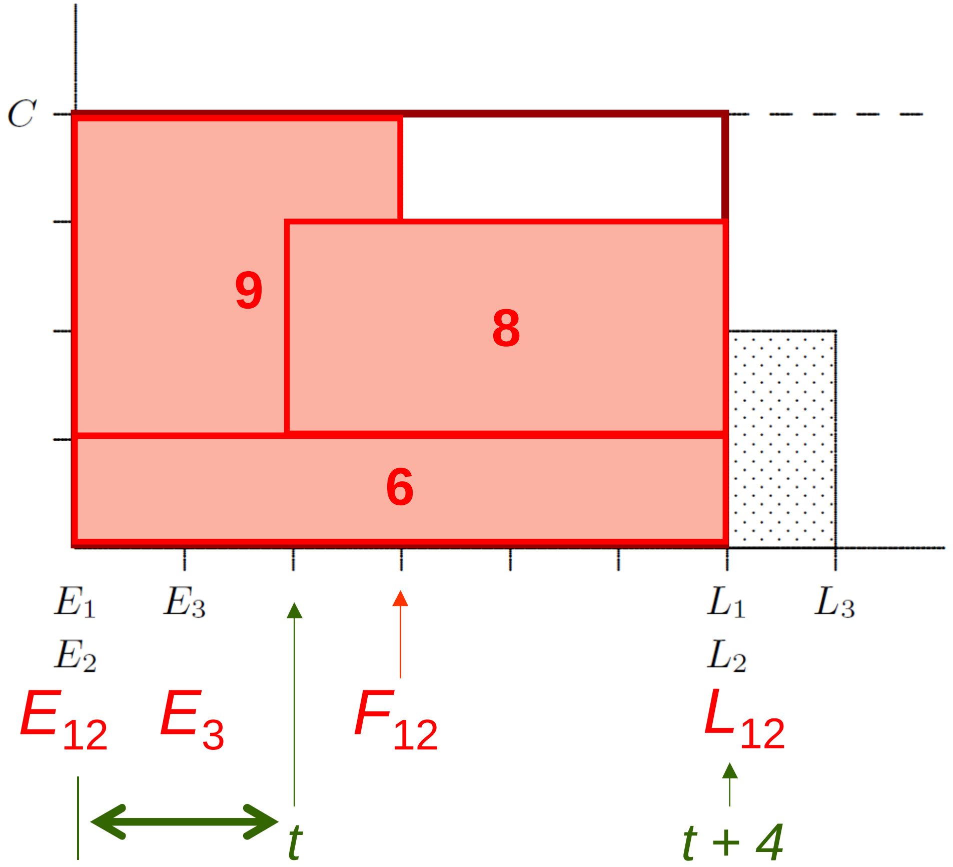
Resource
consumption 2 of
job 3 cannot be
used during this
period

$$e_{\{12\}} + c_3\left(\min\left\{t + p_3, L_{\{12\}}\right\} - t\right)$$

Not-first/not-last rules

Now suppose that job 3 starts at some time *t* before F_{12}. We will derive a contradiction.

Total energy
required between
E_{12} and L_{12} is
6 + 9 + 2·(min{t + 4,6} – *t*)
+ 2·(t – 0)



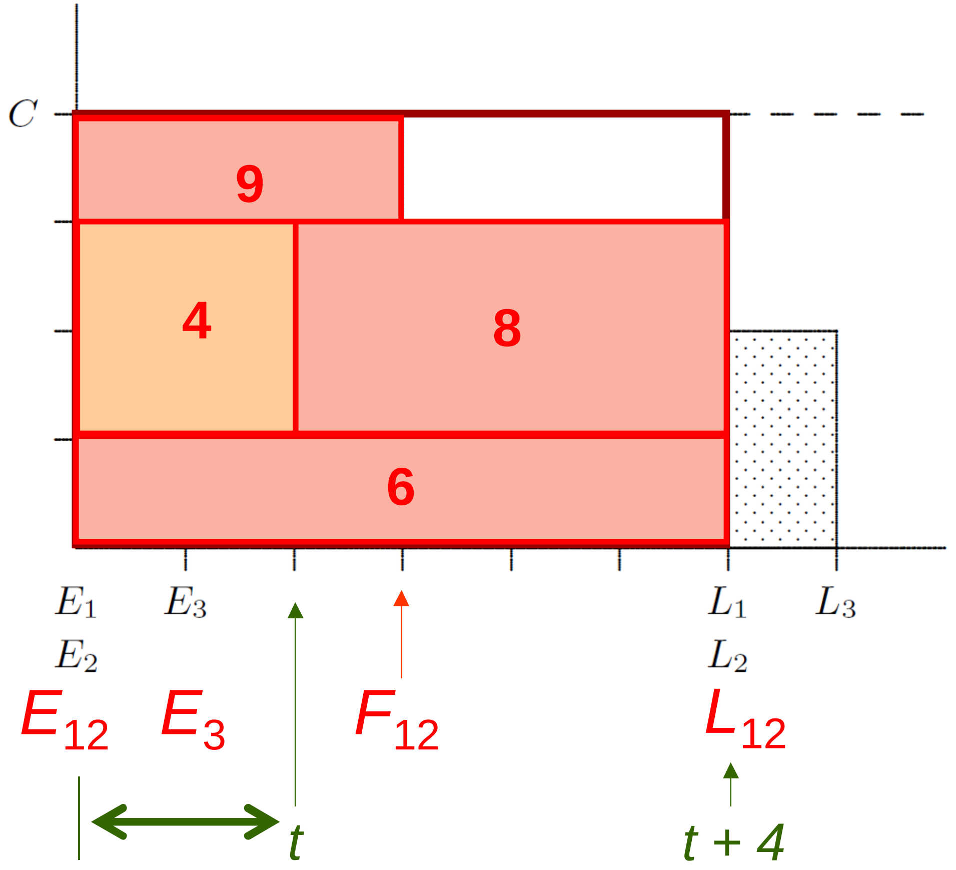
Resource
consumption 2 of
job 3 cannot be
used during this
period

$$e_{\{12\}} + c_3\left(\min\left\{t + p_3, L_{\{12\}}\right\} - t\right) + c_3\left(t - E_{\{12\}}\right)$$

Not-first/not-last rules

Now suppose that job 3 starts at some time *t* before F_{12}. We will derive a contradiction.

Total energy required between E_{12} and L_{12} is
6 + 9 + 2·(min{t + 4,6} – *t*)
+ 2·(t – 0)



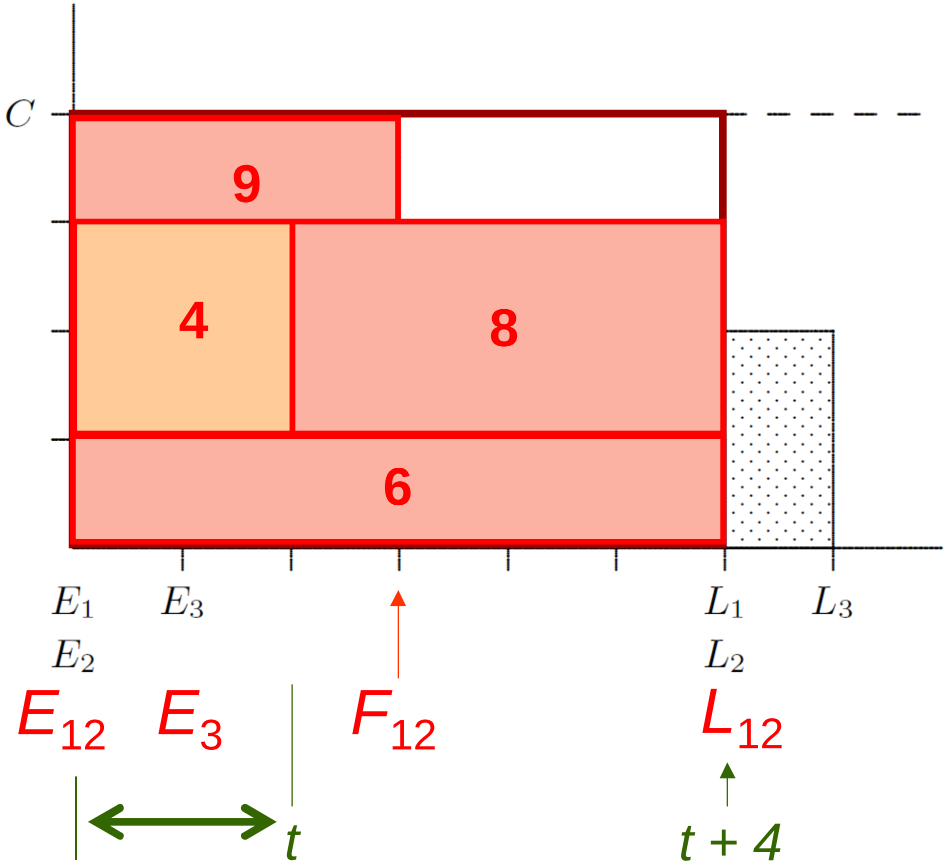
This expression simplifies…

$$e_{\{12\}} + c_3\left(\min\left\{t + p_3, L_{\{12\}}\right\} - t\right) + c_3\left(t - E_{\{12\}}\right)$$

Not-first/not-last rules

Now suppose that job 3 starts at some time *t* before F_{12}. We will derive a contradiction.

Total energy required between E_{12} and L_{12} is 6 + 9 + 2·(min{*t* + 4,6} – 0



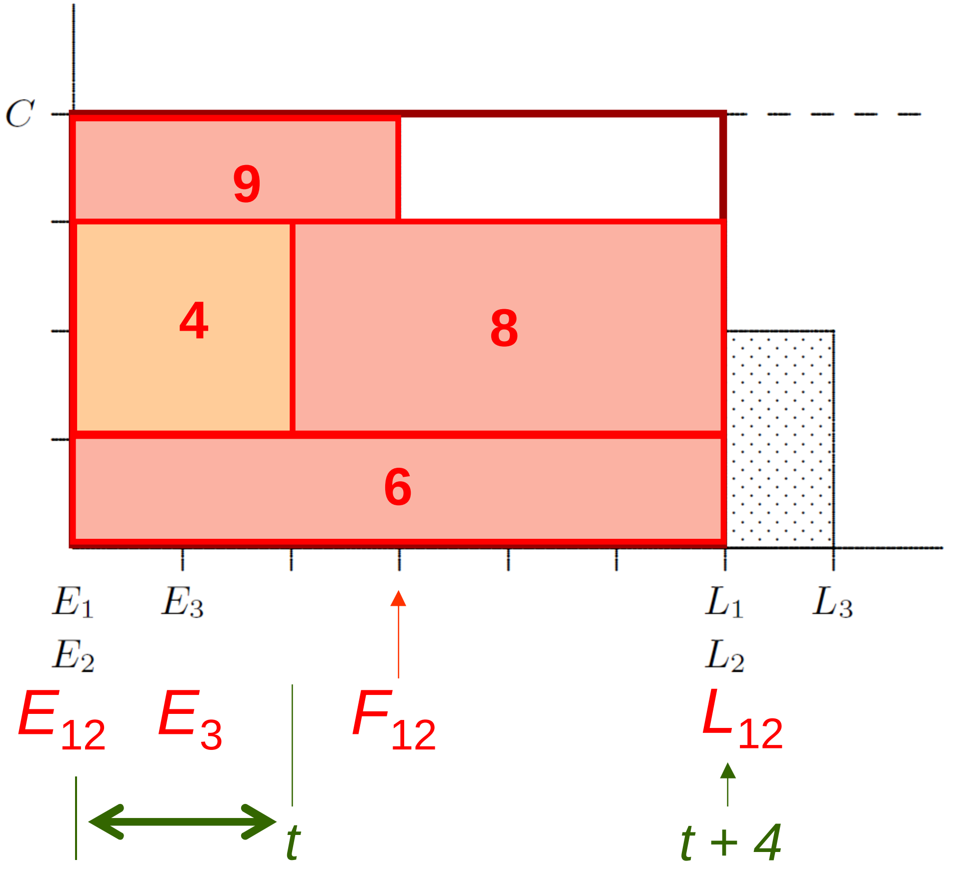
This expression simplifies…

$$e_{\{12\}} + c_3\left(\min\left\{t + p_3, L_{\{12\}}\right\} - E_{\{12\}}\right)$$

Not-first/not-last rules

Now suppose that job 3 starts at some time *t* before F_{12}. We will derive a contradiction.

Total energy required between E_{12} and L_{12} is $6 + 9 + 2{\cdot}(\min\{t + 4{,}6\} – 0$



Because $t \geq E_3$, we have…

$$e_{\{12\}} + c_3\left(\min\left\{t + p_3, L_{\{12\}}\right\} - E_{\{12\}}\right)$$

Not-first/not-last rules

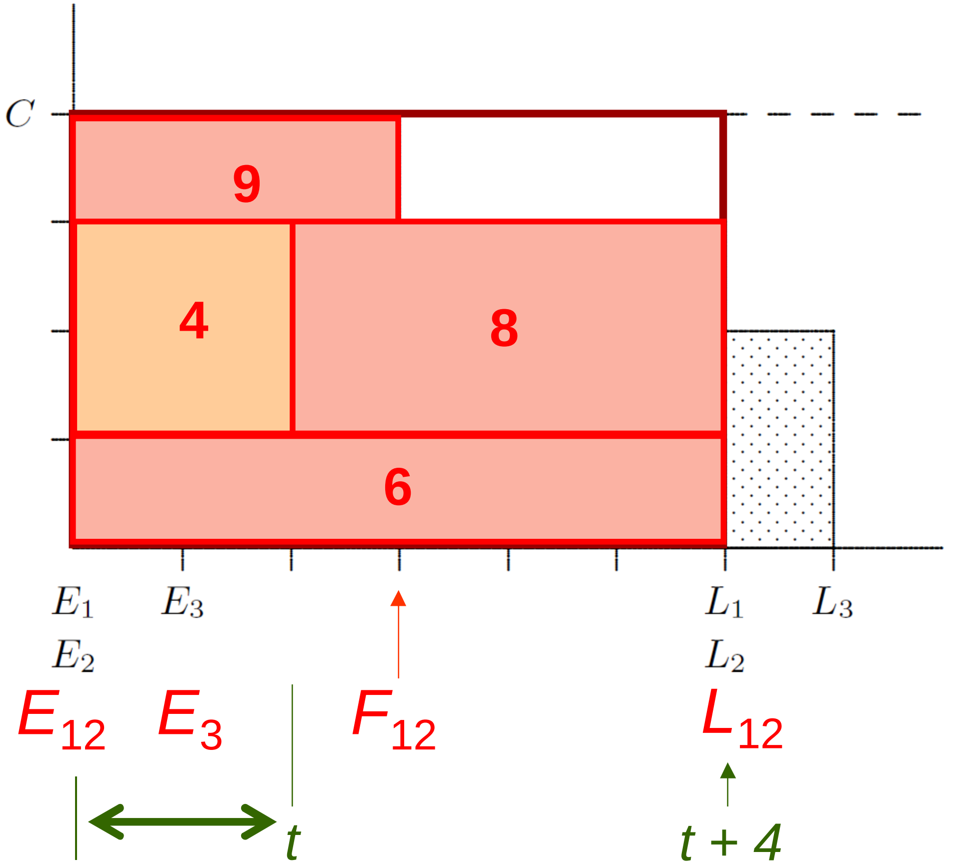
Now suppose that job 3 starts at some time *t* before F_{12}. We will derive a contradiction.

Total energy required between E_{12} and L_{12} is

$6 + 9 + 2 \cdot (\min\{t + 4, 6\} – 0)$

$\geq 6 + 9 + 2 \cdot (\min\{0 + 4, 6\} – 0$

Because $t \geq E_3$, we have…



$$\geq e_{\{12\}} + c_3\left(\min\left\{E_3 + p_3, L_{\{12\}}\right\} - E_{\{12\}}\right)$$

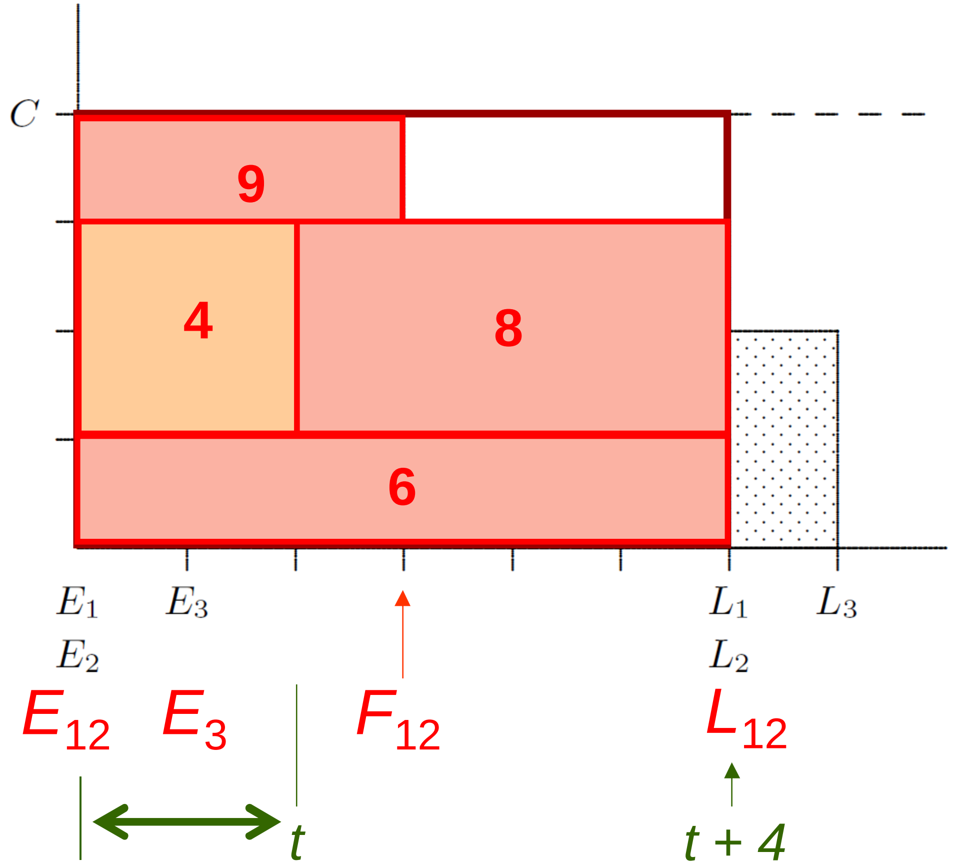
Not-first/not-last rules

Now suppose that job 3 starts at some time *t* before F_{12}. We will derive a contradiction.

Total energy required between E_{12} and L_{12} is $6 + 9 + 2\cdot(\min\{t + 4,6\} – 0)$

$\geq 6 + 9 + 2\cdot(\min\{1 + 4,6\} – 0$
$= 25$

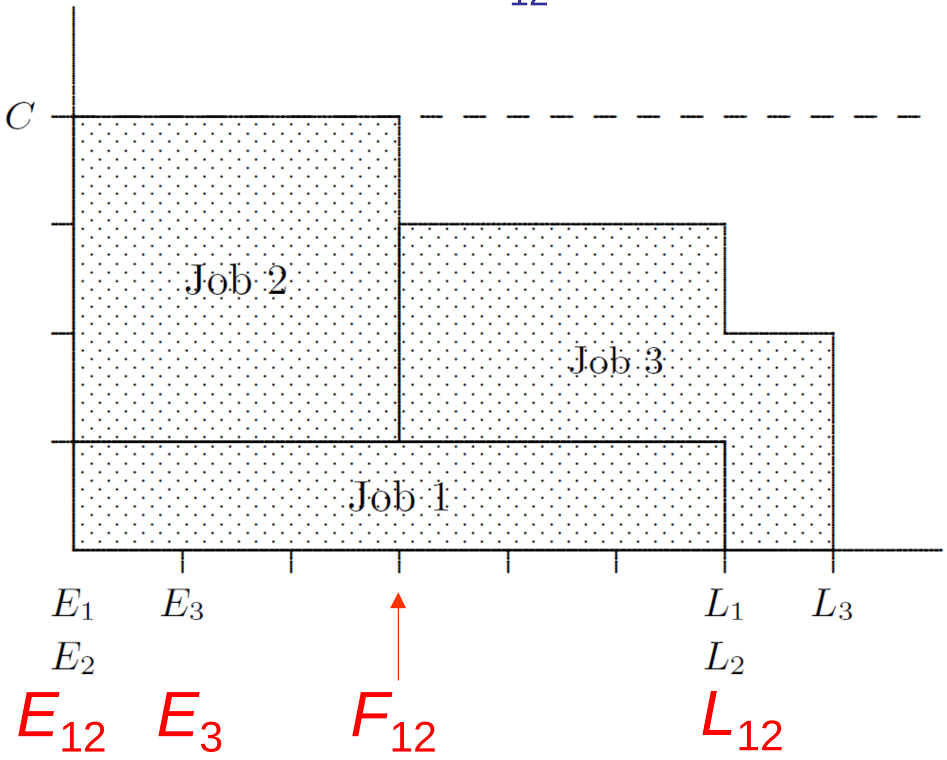
Available energy is $4\cdot 6 = 24$



$$\geq e_{\{12\}} + c_3\left(\min\left\{E_3 + p_3, L_{\{12\}}\right\} - E_{\{12\}}\right) > C\cdot\left(L_{\{12\}} - E_{\{12\}}\right)$$

Not-first/not-last rules

We conclude that job 3 cannot start before F_{12}.

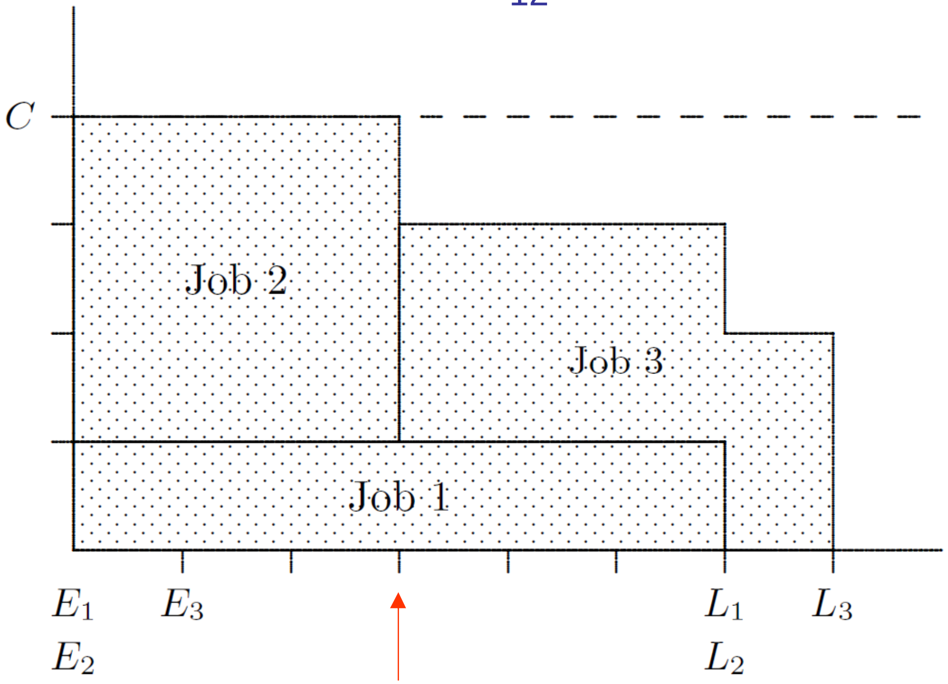


$$\geq e_{\{12\}} + c_3\left(\min\left\{E_3 + p_3, L_{\{12\}}\right\} - E_{\{12\}}\right) > C \cdot \left(L_{\{12\}} - E_{\{12\}}\right)$$

Not-first/not-last rules

We conclude that job 3 cannot start before F_{12}.

Update E_3 to F_{12} = 3



E_{12} F_{12} L_{12}

$\boldsymbol{E_3}$

$$\geq e_{\{12\}} + c_3 \left(\min\left\{E_3 + p_3, L_{\{12\}}\right\} - E_{\{12\}}\right) > C \cdot \left(L_{\{12\}} - E_{\{12\}}\right)$$

Not-first/not-last rules

In general,

If $E_J \le E_k < F_J$

and $e_J + c_k\left(\min\{E_k + p_k, L_J\} - E_J\right) > C \cdot (L_J - E_J)$

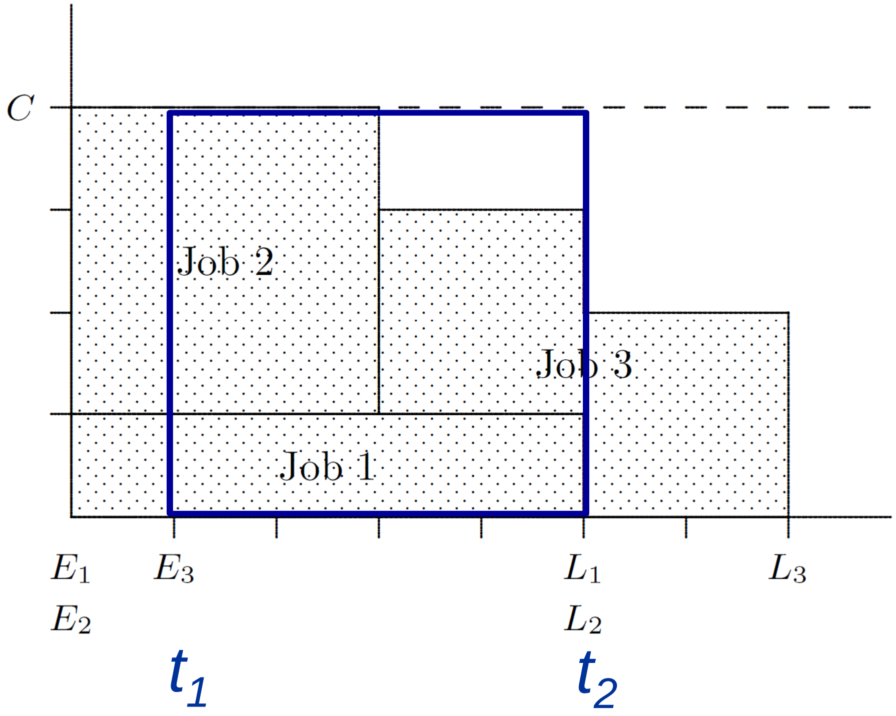
then $\neg(k \ll J)$

and we update E_k to F_J.

$$\ge e_{\{12\}} + c_3\left(\min\left\{E_3 + p_3, L_{\{12\}}\right\} - E_{\{12\}}\right) > C \cdot \left(L_{\{12\}} - E_{\{12\}}\right)$$

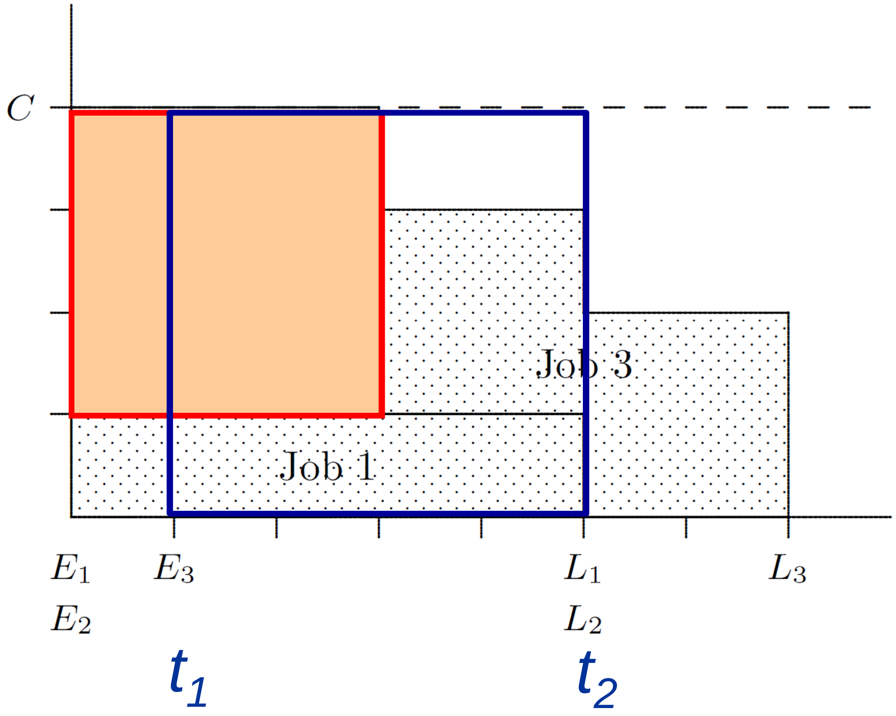
Energetic reasoning

Choose an interval [t_1,t_2]



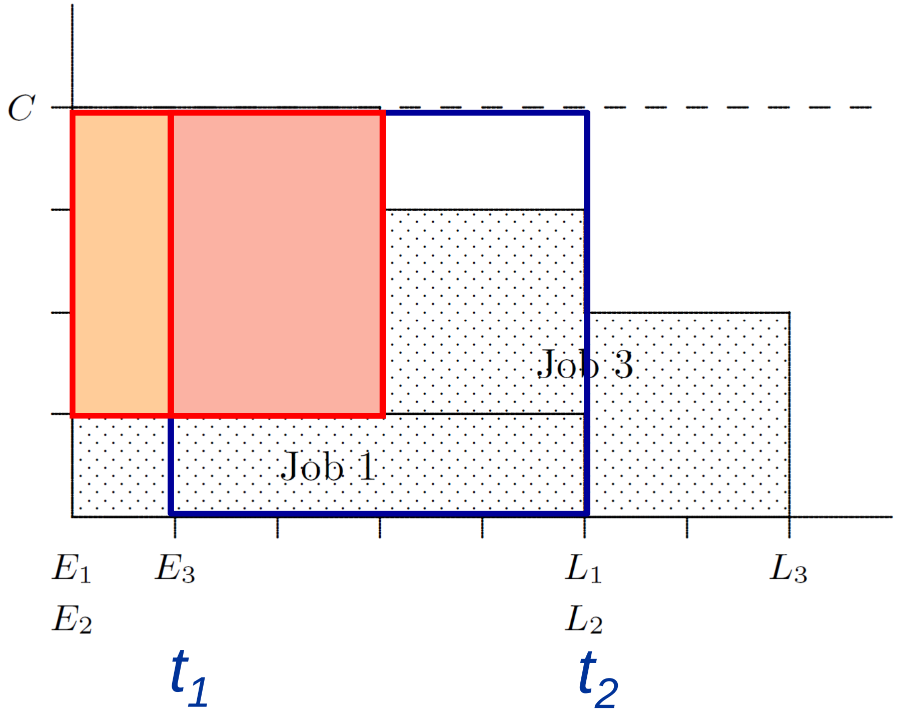
Energetic reasoning

Left shift job 2 (move it as far left as possible).



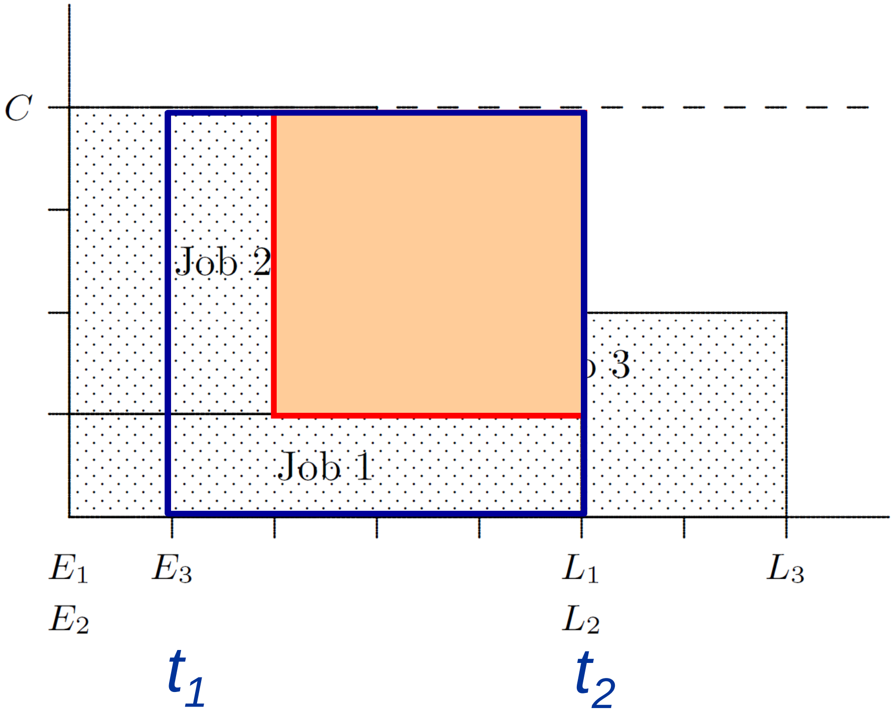
Energetic reasoning

Overlap area is 6.



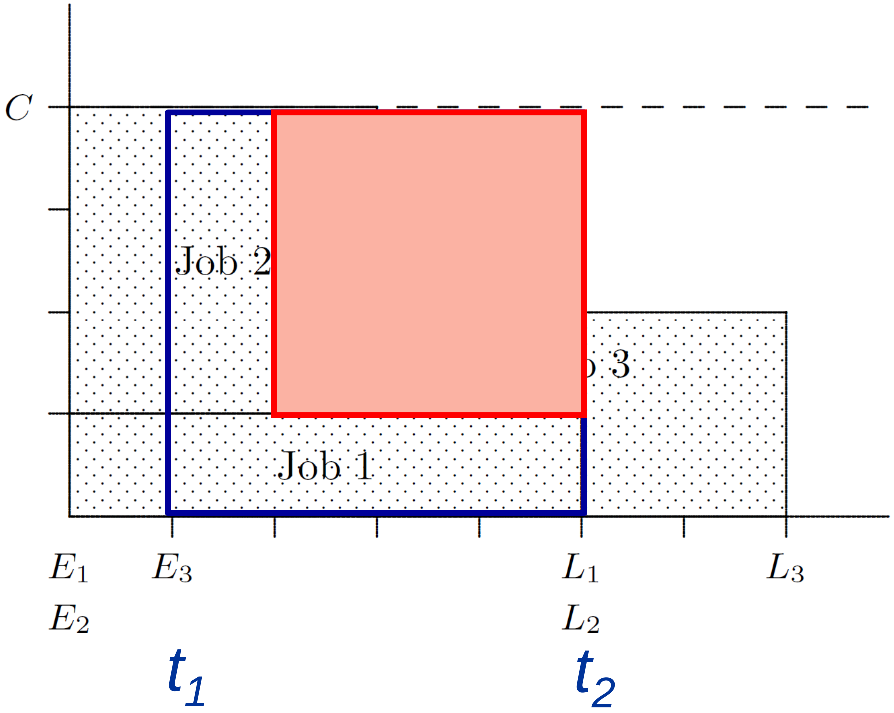
Energetic reasoning

Right shift job 2 (move it as far right as possible).



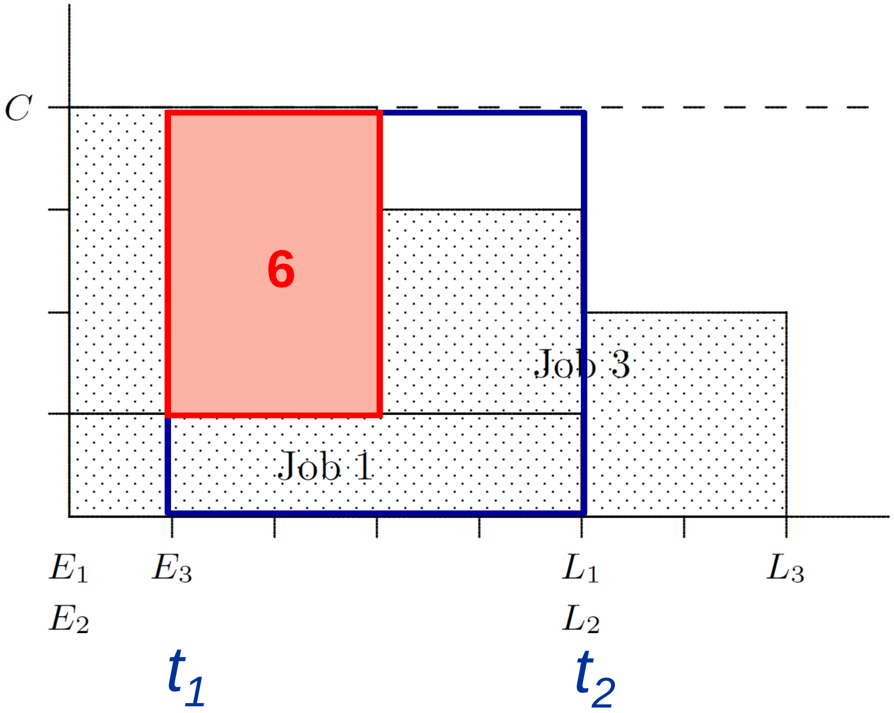
Energetic reasoning

Overlap area is 9



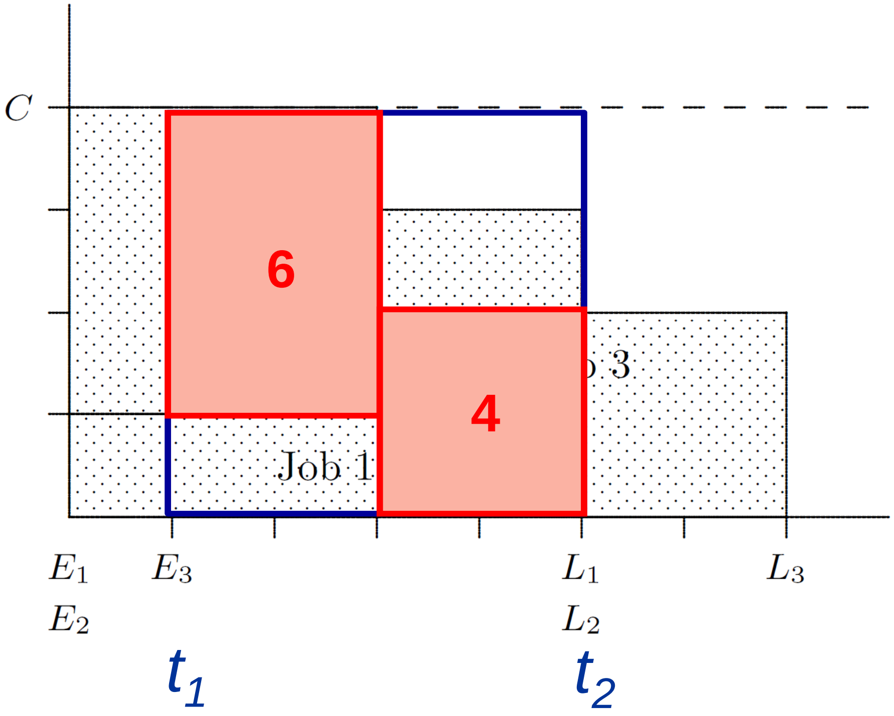
Energetic reasoning

Job 2 must use at least min{6,9} energy inside the interval [t_1,t_2]



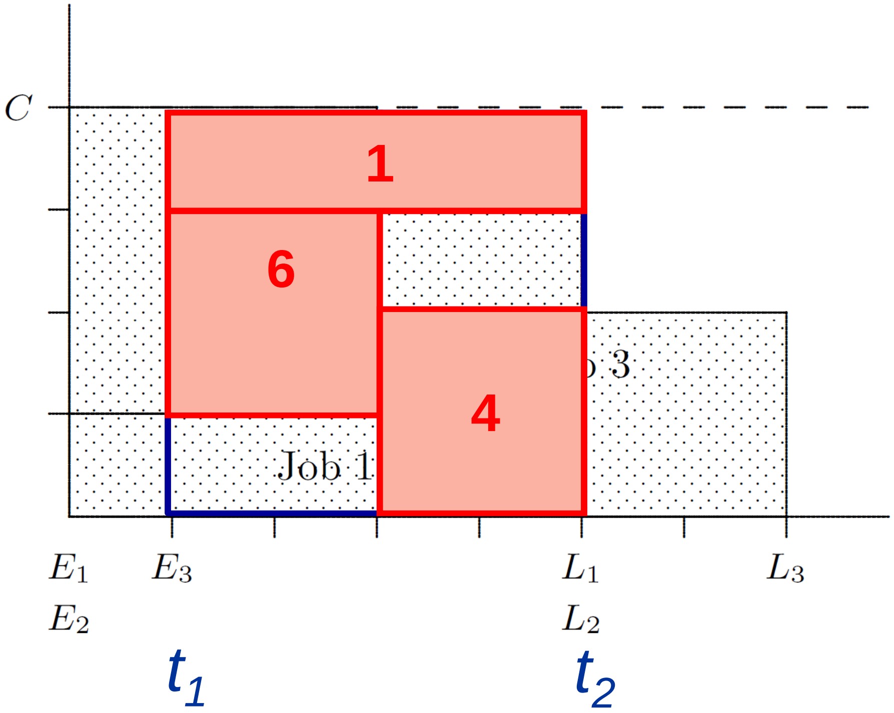
Energetic reasoning

Do the same for job 3.



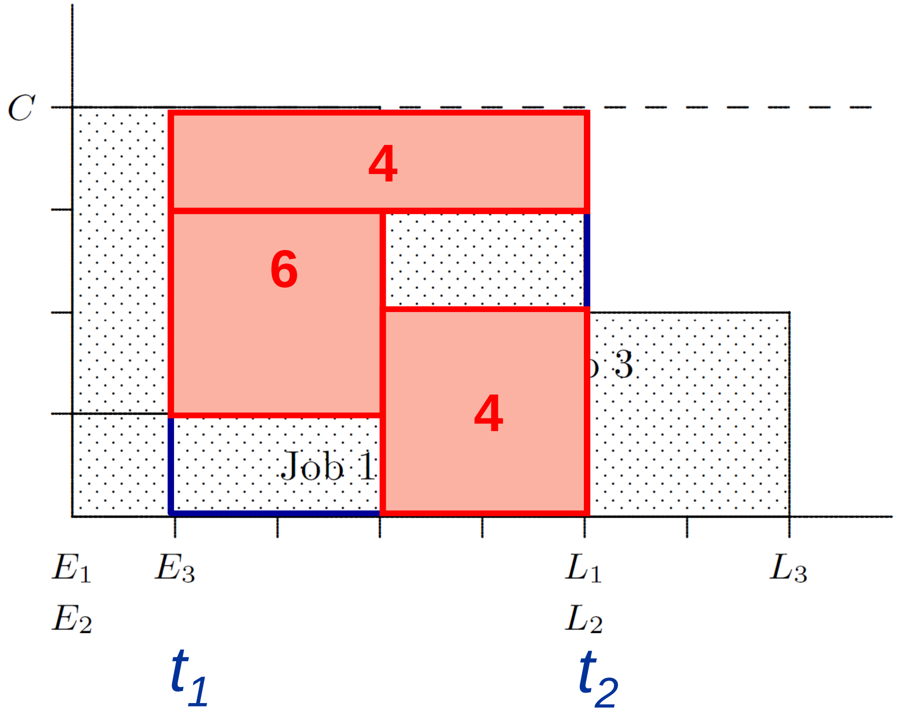
Energetic reasoning

And job 1.



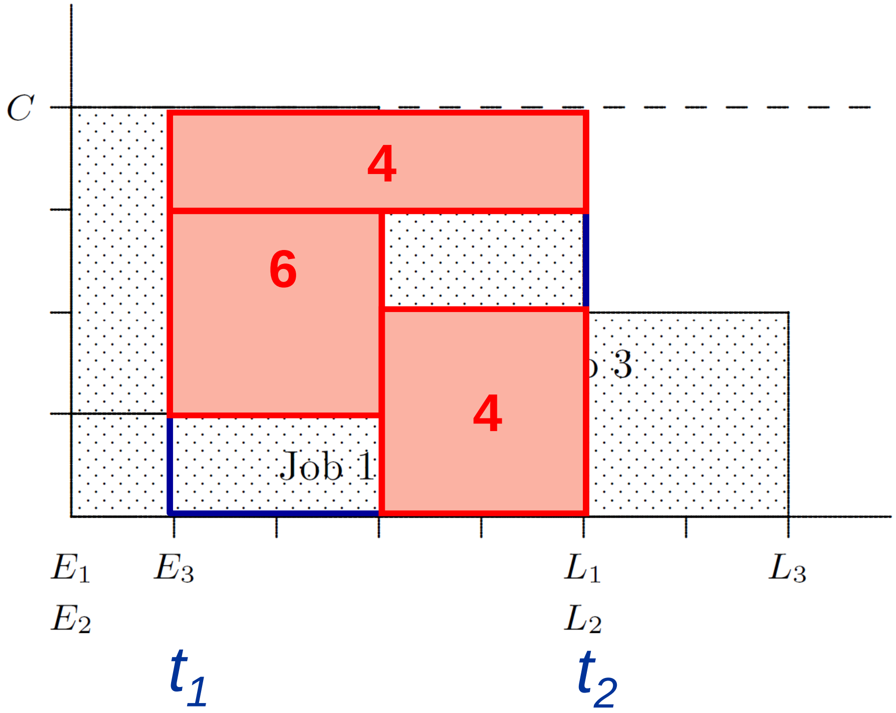
Energetic reasoning

Area required in the interval [t_1,t_2] is 6 + 4 + 4 = 14.
Area available is 16. So we are OK.



Energetic reasoning

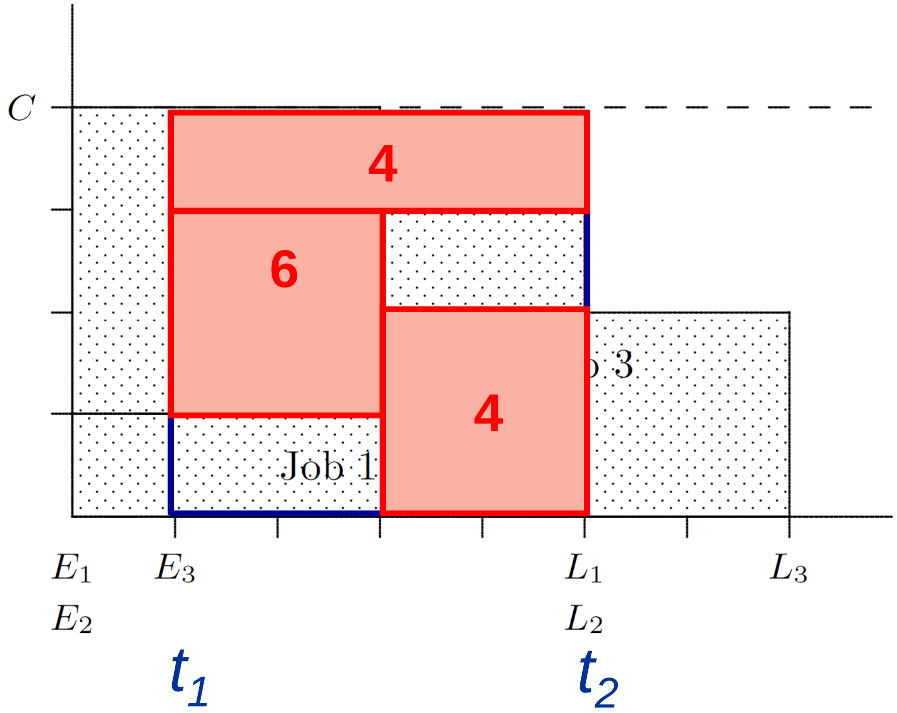
Energy required in the interval [t_1,t_2] is 6 + 4 + 4 = 14.
Area available is 16. So we are OK.



If energy required > area available, problem is infeasible.

Energetic reasoning

Energy required in the interval [t_1,t_2] is 6 + 4 + 4 = 14.
Area available is 16. So we are OK.



Similar principle can be used to update bounds.

Energetic reasoning

Theorem. It suffices to check pairs (t1,t2) in the union of sets

$$\{(t_1, t_2) \mid t_1 \in T_1,\ t_2 \in T_2,\ t_1 < t_2\}$$
$$\{(t_1, t_2) \mid t_1 \in T_1,\ t_2 \in T(t_1),\ t_1 < t_2\}$$
$$\{(t_1, t_2) \mid t_2 \in T_1,\ t_1 \in T(t_2),\ t_1 < t_2\}$$

where

$$T_1 = \{E_i, F_i, S_i \mid i = 1, \ldots, n\}$$
$$T_2 = \{F_i, S_i, L_i \mid i = 1, \ldots, n\}$$
$$T(t) = \{E_i + L_i - t \mid i = 1, \ldots, n\}$$

The SAT Problem

Propositional Logic

Conversion to CNF

Unit Resolution

DPLL

Implication Graph

Backdoors and Branching

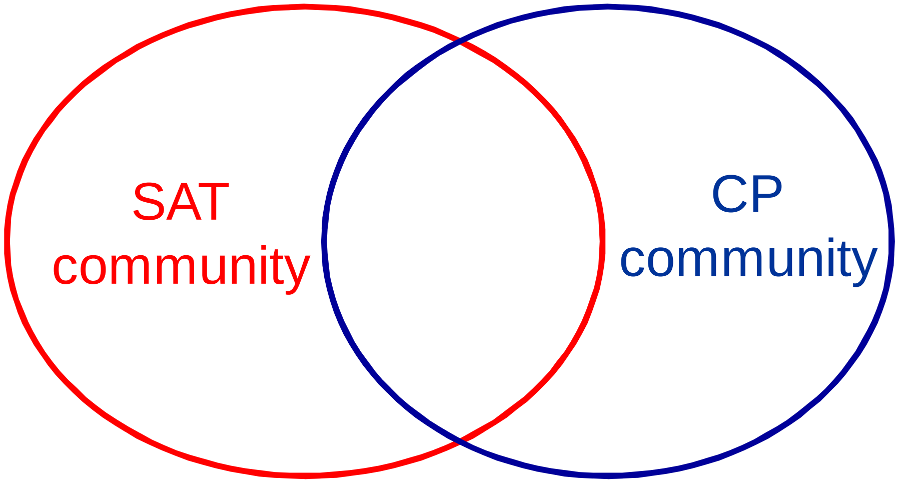
Propositional Satisfiability Problem

- A general approach to constraint solving when variables are discrete.
    - First reduce the problem to SAT.
    - Then solve it using a SAT solver.
    - The solvers are highly engineered and extremely **fast**.

SAT Solvers

- A SAT competition is held regularly.
    - About 50 solvers compete.
- Most solvers evolved from DPLL
    - Davis-Putnam-Loveland-Logemann algorithm
    - …and use CDCL (conflict-directed clause learning).
- Breakthrough solver was CHAFF.
    - A popular open-source solver is MiniSAT.

SAT and CP



- Similarities:
  - Focus on logical inference.
  - Use of branching and propagation.
- Difference:
  - SAT doesn’t use global constraints.
  - SAT uses atomistic modeling, like mixed integer programming.
- CP learned problem-solving ideas from SAT.

Propositional Logic

- Propositional formulas connect boolean variables with **and**, **or**, **not**, **implies**, etc.
    - There are no quantifiers.

x_j is a formula, where x_j is a boolean variable

$A \vee B$ is a formula (A or B), where A and B are formulas

$A \wedge B$ is a formula (A and B)

$\bar{A}$ is a formula (not A)

$A \rightarrow B$ is a formula defined as $\bar{A} \vee B$ (material implication)

$A \equiv B$ is a formula defined as $(A \rightarrow B) \wedge (B \rightarrow A)$

Propositional Logic

- A formula in **conjunctive normal form (CNF)** is a conjunction of clauses.
  - A **literal** is x_j or $\bar{x}_j$
  - A **clause** is a disjunction of literals, e.g. $\bar{x}_1 \vee x_2 \vee \bar{x}_3$
- Example of CNF:

$$(\bar{x}_1 \vee \bar{x}_3) \wedge (x_2 \vee x_1) \wedge (x_2 \vee \bar{x}_3)$$

Propositional Logic

- The **SAT** problem is to satisfy a formula in CNF.
    - That is, assign truth values (0 or 1) to the variables to make the formula true.

Propositional Logic

- The **SAT** problem is to satisfy a formula in CNF.
    - That is, assign truth values (0 or 1) to the variables to make the formula true.
- Some problems already have logical form
    - Circuit verification.
    - Product configuration.
    - These can be converted to CNF and solved as SAT problems.
- Most problems must be rewritten in logical form.

Propositional Logic

- Converting a problem to CNF is a key element of SAT-based problem solving.
    - General syntactic methods.
    - General semantic methods.
    - Problem-specific methods (growing literature).

Conversion to CNF

- **Syntactic** rules for converting a propositional formula to CNF.
    - These are useful if we already know how to write the constraints as a propositional formula.

$\overline{(A \vee B)} \equiv \overline{A} \wedge \overline{B}$ De Morgan's law

$\overline{(A \wedge B)} \equiv \overline{A} \vee \overline{B}$ De Morgan's law

$(A \vee (B \wedge C)) \equiv ((A \vee B) \wedge (B \vee C))$ distribution

Conversion to CNF

- Example

$$\overline{(x_1 \vee \overline{x}_2)} \vee (x_1 \wedge \overline{x}_3)$$

$$\equiv (\overline{x}_1 \wedge x_2) \vee (x_1 \wedge \overline{x}_3)$$ De Morgan

$$\equiv (\overline{x}_1 \vee x_1) \wedge (\overline{x}_1 \vee \overline{x}_3) \wedge (x_2 \vee x_1) \wedge (x_2 \vee \overline{x}_3)$$ distribution

$$\equiv (\overline{x}_1 \vee \overline{x}_3) \wedge (x_2 \vee x_1) \wedge (x_2 \vee \overline{x}_3)$$ remove tautology

Conversion to CNF

- Another example: Hiring problem
  - A company must hire some staff to complete a task and has workers 1, …, 6 to choose from.
  - Workers 3 and 4 are temporary workers.

Must hire at least 1 of workers 1,5,6

Cannot hire 6 unless it hires 1 or 5

Cannot hire 5 unless it hires 2 or 6

Must hire 2 if it hires 5 and 6.

Must hire a temporary worker if 1 or 2

Can hire neither 1 nor 2 if a temp worker

Conversion to CNF

- Another example: Hiring problem
  - A company must hire some staff to complete a task and has workers 1, …, 6 to choose from.
  - Workers 3 and 4 are temporary workers.

Must hire at least 1 of workers 1,5,6	$x_1 \vee x_5 \vee x_6$
Cannot hire 6 unless it hires 1 or 5	$x_6 \to (x_1 \vee x_5)$
Cannot hire 5 unless it hires 2 or 6	$x_5 \to (x_2 \vee x_6)$
Must hire 2 if it hires 5 and 6.	$(x_5 \wedge x_6) \to x_2$
Must hire a temporary worker if 1 or 2	$(x_1 \vee x_2) \to (x_3 \vee x_4)$
Can hire neither 1 nor 2 if a temp worker	$(x_3 \vee x_4) \to (\bar{x}_1 \wedge \bar{x}_2)$

Conversion to CNF

- This is easily converted to CNF.

$$
\begin{array}{lcl}
x_1 \vee x_5 \vee x_6 & \equiv & x_1 \vee x_5 \vee x_6 \\
x_6 \rightarrow (x_1 \vee x_5) & \equiv & \bar{x}_6 \vee x_1 \vee x_5 \\
x_5 \rightarrow (x_2 \vee x_6) & \equiv & \bar{x}_5 \vee x_2 \vee x_6 \\
(x_5 \wedge x_6) \rightarrow x_2 & \equiv & (\bar{x}_5 \vee x_2) \wedge (\bar{x}_6 \vee x_2) \\
(x_1 \vee x_2) \rightarrow (x_3 \vee x_4) & \equiv & (\bar{x}_1 \vee x_3 \vee x_4) \wedge (\bar{x}_2 \vee x_3 \vee x_4) \\
(x_3 \vee x_4) \rightarrow (\bar{x}_1 \wedge \bar{x}_2) & \equiv & (\bar{x}_3 \vee \bar{x}_1) \wedge (\bar{x}_3 \vee \bar{x}_2) \wedge (\bar{x}_4 \vee \bar{x}_1) \wedge (\bar{x}_4 \vee \bar{x}_2)
\end{array}
$$

Conversion to CNF

- However, this method can require exponential time and space.
  - For example,

$$(x_1 \vee y_2) \vee \cdots \vee (x_n \vee y_n)$$

converts to a conjunction of 2^n clauses of the form

$$F_1 \vee \cdots \vee F_n$$

where each F_j is x_j or y_j.

Conversion to CNF

- To avoid exponential blowup, lift into higher dimensional space.
    - Rather than distribute $F \vee G$, replace it with

$$(z_1 \vee z_2) \wedge (\bar{z}_1 \vee F) \wedge (\bar{z}_2 \vee G)$$

where z_1, z_2 are new variables.

Conversion to CNF

- To avoid exponential blowup, lift into higher dimensional space.
  - Rather than distribute $F \vee G$, replace it with

$$(z_1 \vee z_2) \wedge (\bar{z}_1 \vee F) \wedge (\bar{z}_2 \vee G)$$

where z_1, z_2 are new variables.

- For example, $(x_1 \vee y_2) \vee \cdots \vee (x_n \vee y_n)$

converts to the CNF formula

$$(z_1 \vee \cdots \vee z_n) \wedge \bigwedge_{j=1}^{n} (\bar{z}_j \vee x_j) \wedge (\bar{z}_j \vee y_j)$$

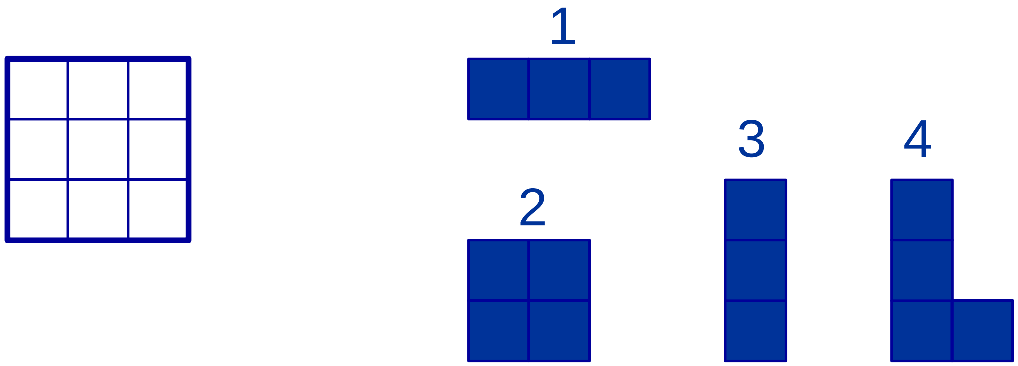
This requires linear time and space.

Conversion to CNF

- Semantic conversion can be used whenever a truth table is available.
    - However, it is exponential in time and space.

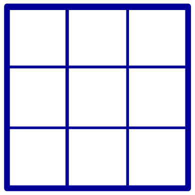
Conversion to CNF

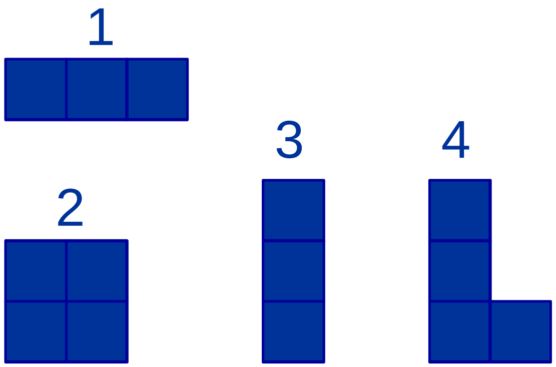
- Semantic conversion can be used whenever a truth table is available.
    - However, it is exponential in time and space.
- Example: The buildings assigned to the block on the left must fit:



Conversion to CNF

- Let $x_i = 1$ (true) when building *i* is assigned to the block.





Truth table:

x_1	x_2	x_3	x_4	
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Conversion to CNF

- Each false entry generates a clause

$x_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_4$

$\bar{x}_1 \vee x_2 \vee x_3 \vee \bar{x}_4$

$\bar{x}_1 \vee x_2 \vee \bar{x}_3 \vee x_4$

$\bar{x}_1 \vee x_2 \vee \bar{x}_3 \vee \bar{x}_4$

$\bar{x}_1 \vee \bar{x}_2 \vee x_3 \vee \bar{x}_4$

$\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee x_4$

$\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_4$

This says $(x_1,x_2,x_3,x_4) \neq (0,1,1,1)$, or

$$\overline{\bar{x}_1 \wedge x_2 \wedge x_3 \wedge x_4}$$

x_1	x_2	x_3	x_4	
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

Conversion to CNF

- Each false entry generates a clause

$$x_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_4$$
$$\bar{x}_1 \vee x_2 \vee x_3 \vee \bar{x}_4$$
$$\bar{x}_1 \vee x_2 \vee \bar{x}_3 \vee x_4$$
$$\bar{x}_1 \vee x_2 \vee \bar{x}_3 \vee \bar{x}_4$$
$$\bar{x}_1 \vee \bar{x}_2 \vee x_3 \vee \bar{x}_4$$
$$\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee x_4$$
$$\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_4$$

This says
(x_1,x_2,x_3,x_4)
$\neq (0,1,1,1)$,
or

$$\overline{\bar{x}_1 \wedge x_2 \wedge x_3 \wedge x_4}$$

We will simplify this later.

x_1	x_2	x_3	x_4	
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	0

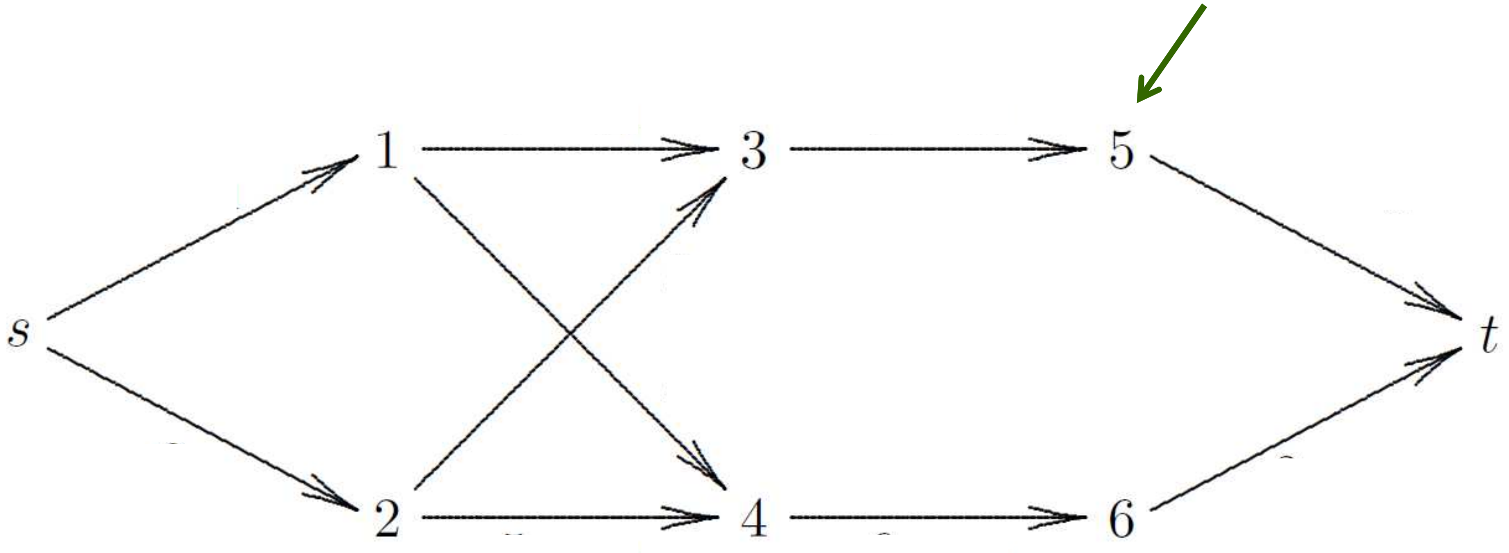
Conversion to CNF

- **Problem specific** conversion to CNF.
  - Sometimes, constraints in binary variables are easy to covert to CNF.
- Example: Airline crew rostering
  - Assign rosters (sequences of flights) to crews.
  - Each crew gets exactly one roster.
  - Each flight is staffed by at least one crew.

Conversion to CNF

- Small problem instance: 2 crews and 4 rosters.
  - Each *s*-*t* path below is a feasible sequence of flights (roster) for a crew.
  - Namely, 135, 146, 235, 246.

Flight number



Conversion to CNF

- Small problem instance: 2 crews and 4 rosters.
  - Rosters: 135, 146, 235, 246.
- Let x_{ij} = 1 when crew *i* is assigned to roster *j*.
- Two types of constraints:
  - Each crew is assigned exactly one roster.
  - Each flight is covered by at least one crew.

Conversion to CNF

- Small problem instance: 2 crews and 4 rosters.
  - Rosters: 135, 146, 235, 246.
- Let x_{ij} = 1 when crew *i* is assigned to roster *j*.
- Each crew is assigned exactly one roster.
  - Exactly one of x_{i1}, x_{i2}, x_{i3}, x_{i4} is true for each crew *i*.

$$\begin{array}{ll}
x_{11} \vee x_{11} \vee x_{11} \vee x_{14} & x_{21} \vee x_{21} \vee x_{21} \vee x_{24} \\
\overline{x}_{11} \vee \overline{x}_{12} & \overline{x}_{21} \vee \overline{x}_{22} \\
\overline{x}_{11} \vee \overline{x}_{13} & \overline{x}_{21} \vee \overline{x}_{23} \\
\overline{x}_{11} \vee \overline{x}_{14} & \overline{x}_{21} \vee \overline{x}_{24} \\
\overline{x}_{12} \vee \overline{x}_{13} & \overline{x}_{22} \vee \overline{x}_{23} \\
\overline{x}_{12} \vee \overline{x}_{14} & \overline{x}_{22} \vee \overline{x}_{24} \\
\overline{x}_{13} \vee \overline{x}_{14} & \overline{x}_{23} \vee \overline{x}_{24}
\end{array}$$

Conversion to CNF

- Small problem instance: 2 crews and 4 rosters.
    - Rosters: 135, 146, 235, 246.
- Let $x_{ij} = 1$ when crew *i* is assigned to roster *j*.
- Each flight is covered by at least one crew:

Flight 1 is in rosters 1 and 2

Flight 2 is in rosters 3 and 4

$$\begin{gathered}
x_{11} \vee x_{12} \vee x_{21} \vee x_{22} \\
x_{13} \vee x_{14} \vee x_{23} \vee x_{24} \\
x_{11} \vee x_{13} \vee x_{21} \vee x_{23} \\
x_{12} \vee x_{14} \vee x_{22} \vee x_{24} \\
x_{11} \vee x_{13} \vee x_{21} \vee x_{23} \\
x_{12} \vee x_{14} \vee x_{22} \vee x_{24}
\end{gathered}$$

Conversion to CNF

- Many problems are hard to encode in SAT.
    - Such as problems that include quantities.

Conversion to CNF

- Many problems are hard to encode in SAT.
  - Such as problems that include quantities.

- Example:
  - The 0-1 knapsack inequality

$$\begin{aligned}&300x_0+300x_1+285x_2+285x_3+265x_4+265x_5+230x_6+230x_7+190x_8+200x_9+\\&\quad 400x_{10}+200x_{11}+400x_{12}+200x_{13}+400x_{14}+200x_{15}+400x_{16}+200x_{17}+400x_{18}\geq 2701\end{aligned}$$

translates to 117,520 clauses.

Resolution Method

- **Resolution** is a simple but complete inference method for clauses.
    - Provably exponential (very hard proof).
    - Far too slow in practice to solve problems, but it has practical applications for simplifying expressions.
    - Invented by W. V. Quine in 1950s (“consensus” for DNF).
    - Achieves domain and *k*-consistency for CNF.

Resolution Method

- **Resolution** is a simple but complete inference method for clauses.
    - Provably exponential (very hard proof).
    - Far too slow in practice to solve problems, but it has practical applications for simplifying expressions.
    - Invented by W. V. Quine in 1950s (“consensus” for DNF).
    - Achieves domain and *k*-consistency for CNF.

- Important special cases:
    - Unit resolution
        - Linear-time propagation method
    - Parallel resolution

Resolution Method

- Resolution generates **resolvents** recursively.
    - Clause set is unsatisfiable if empty clause results.
    - If absorbed clauses removed, this generates all prime implications.
        - = strongest possible implications.

$$\begin{array}{lllll}
x_1 \vee & x_2 \vee & x_3 & & \\
\bar{x}_1 \vee & x_2 & & \vee & \bar{x}_4 \\
& x_2 \vee & x_3 & \vee & \bar{x}_4
\end{array}$$

Resolvent, obtained by resolving on x_1

Must be no other sign changes between clauses.

Resolution Method

- Example of refutation

$$x_1 \vee x_3$$

$$x_1 \vee x_2$$

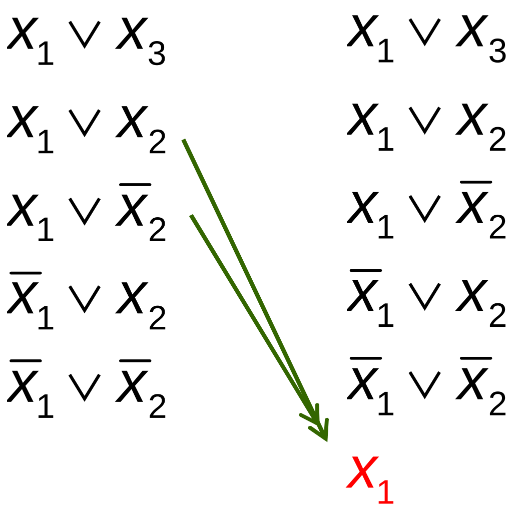
$$x_1 \vee \bar{x}_2$$

$$\bar{x}_1 \vee x_2$$

$$\bar{x}_1 \vee \bar{x}_2$$

Resolution Method

- Example of refutation



$x_1 \vee x_3$

$x_1 \vee x_2$

$x_1 \vee \bar{x}_2$

$\bar{x}_1 \vee x_2$

$\bar{x}_1 \vee \bar{x}_2$

$x_1 \vee x_3$

$x_1 \vee x_2$

$x_1 \vee \bar{x}_2$

$\bar{x}_1 \vee x_2$

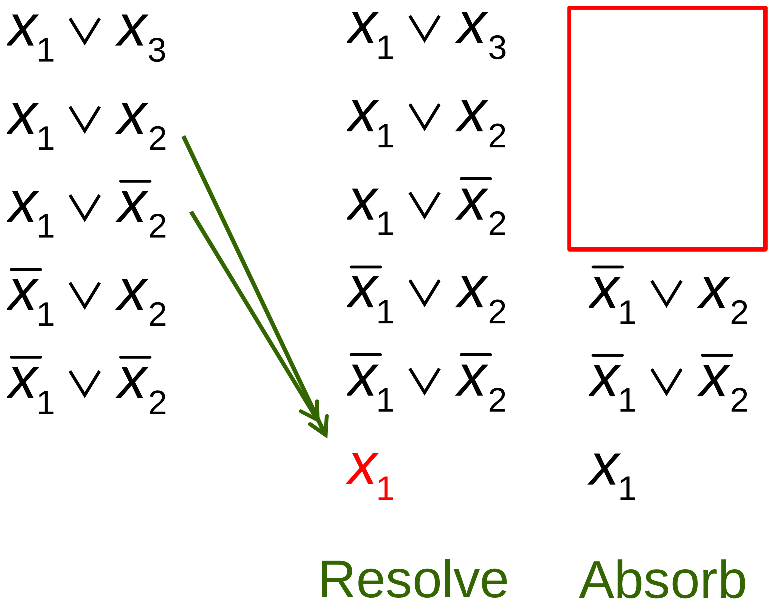
$\bar{x}_1 \vee \bar{x}_2$

x_1

Resolve

Resolution Method

- Example of refutation



$x_1 \vee x_3$

$x_1 \vee x_2$

$x_1 \vee \bar{x}_2$

$\bar{x}_1 \vee x_2$

$\bar{x}_1 \vee \bar{x}_2$

$x_1 \vee x_3$

$x_1 \vee x_2$

$x_1 \vee \bar{x}_2$

$\bar{x}_1 \vee x_2$

$\bar{x}_1 \vee \bar{x}_2$

x_1

Resolve

$\bar{x}_1 \vee x_2$

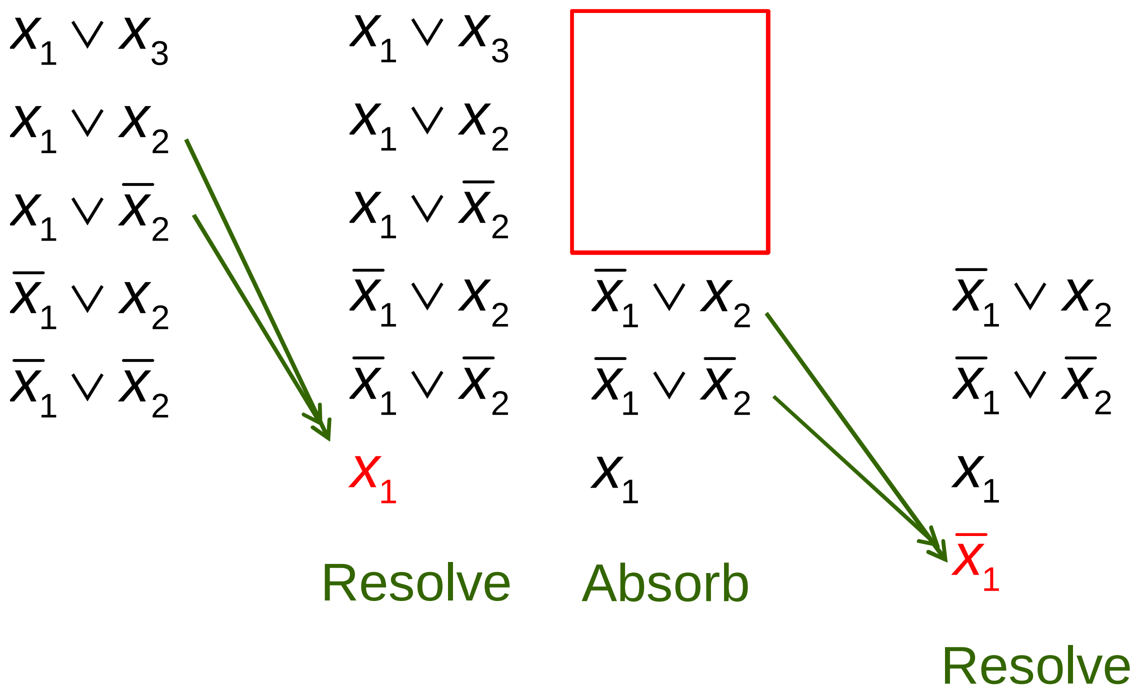
$\bar{x}_1 \vee \bar{x}_2$

x_1

Absorb

Resolution Method

- Example of refutation



$x_1 \vee x_3$

$x_1 \vee x_2$

$x_1 \vee \bar{x}_2$

$\bar{x}_1 \vee x_2$

$\bar{x}_1 \vee \bar{x}_2$

$x_1 \vee x_3$

$x_1 \vee x_2$

$x_1 \vee \bar{x}_2$

$\bar{x}_1 \vee x_2$

$\bar{x}_1 \vee \bar{x}_2$

x_1

Resolve

$\bar{x}_1 \vee x_2$

$\bar{x}_1 \vee \bar{x}_2$

x_1

Absorb

$\bar{x}_1 \vee x_2$

$\bar{x}_1 \vee \bar{x}_2$

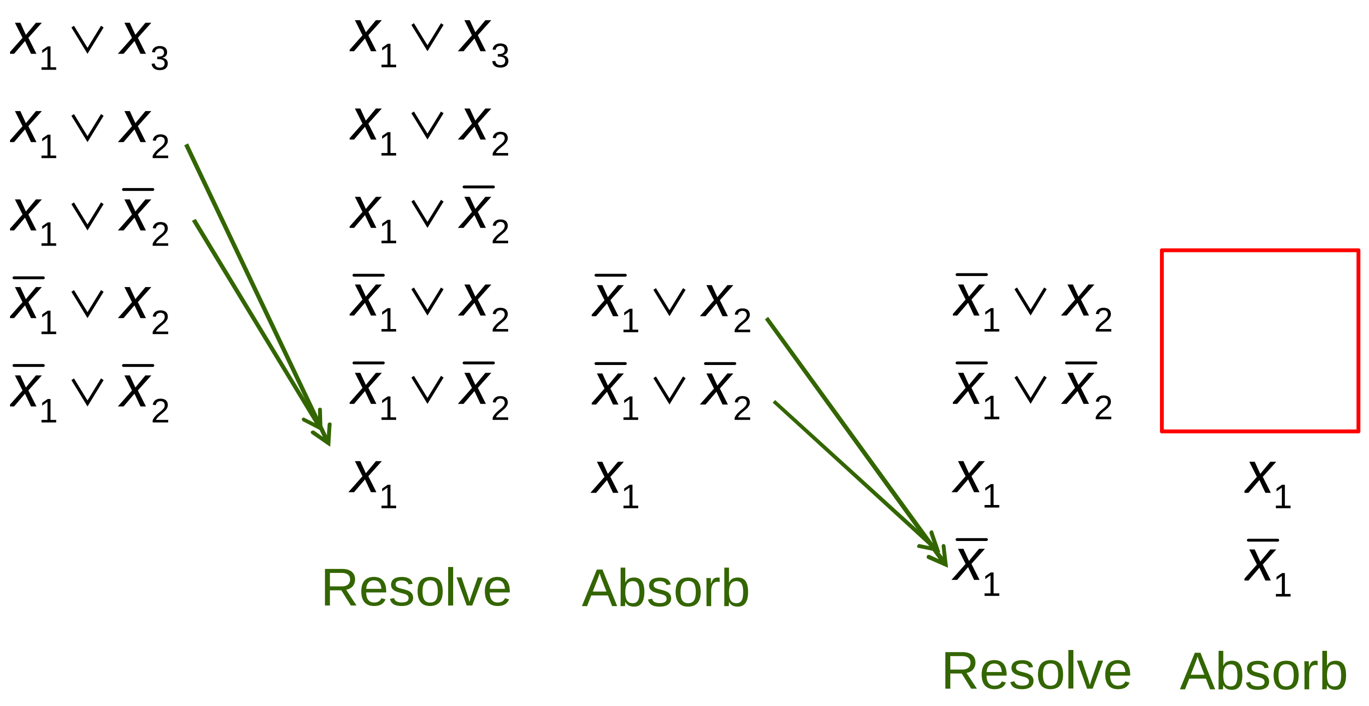
x_1

$\bar{x}_1$

Resolve

Resolution Method

- Example of refutation



$x_1 \vee x_3$

$x_1 \vee x_2$

$x_1 \vee \bar{x}_2$

$\bar{x}_1 \vee x_2$

$\bar{x}_1 \vee \bar{x}_2$

$x_1 \vee x_3$

$x_1 \vee x_2$

$x_1 \vee \bar{x}_2$

$\bar{x}_1 \vee x_2$

$\bar{x}_1 \vee \bar{x}_2$

x_1

Resolve

$\bar{x}_1 \vee x_2$

$\bar{x}_1 \vee \bar{x}_2$

x_1

Absorb

$\bar{x}_1 \vee x_2$

$\bar{x}_1 \vee \bar{x}_2$

x_1

$\bar{x}_1$

Resolve

$\boxed{}$

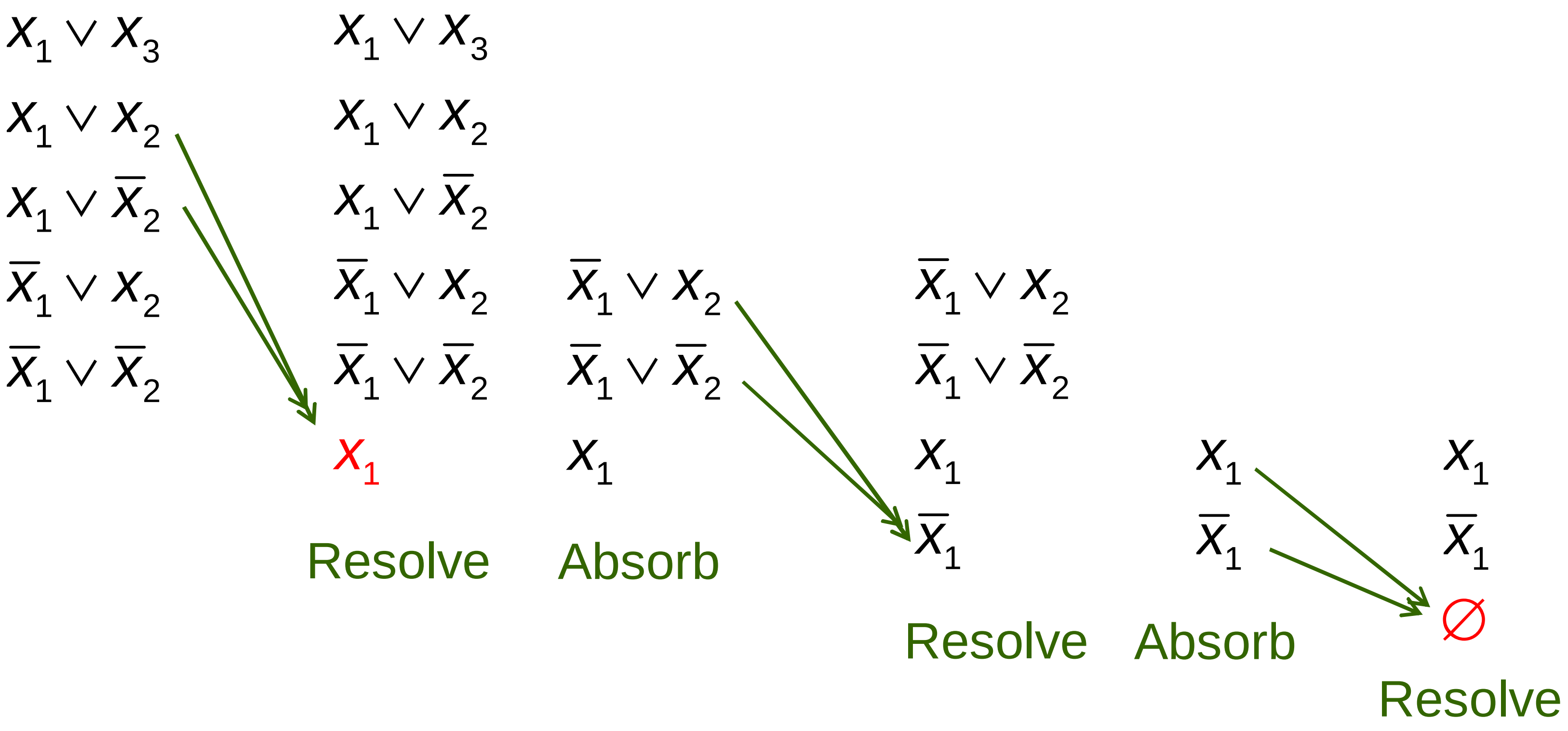
x_1

$\bar{x}_1$

Absorb

Resolution Method

- Example of refutation



$x_1 \vee x_3$
$x_1 \vee x_2$
$x_1 \vee \bar{x}_2$
$\bar{x}_1 \vee x_2$
$\bar{x}_1 \vee \bar{x}_2$

$x_1 \vee x_3$
$x_1 \vee x_2$
$x_1 \vee \bar{x}_2$
$\bar{x}_1 \vee x_2$
$\bar{x}_1 \vee \bar{x}_2$
x_1

Resolve

$\bar{x}_1 \vee x_2$
$\bar{x}_1 \vee \bar{x}_2$
x_1

Absorb

$\bar{x}_1 \vee x_2$
$\bar{x}_1 \vee \bar{x}_2$
x_1
$\bar{x}_1$

Resolve

x_1
$\bar{x}_1$

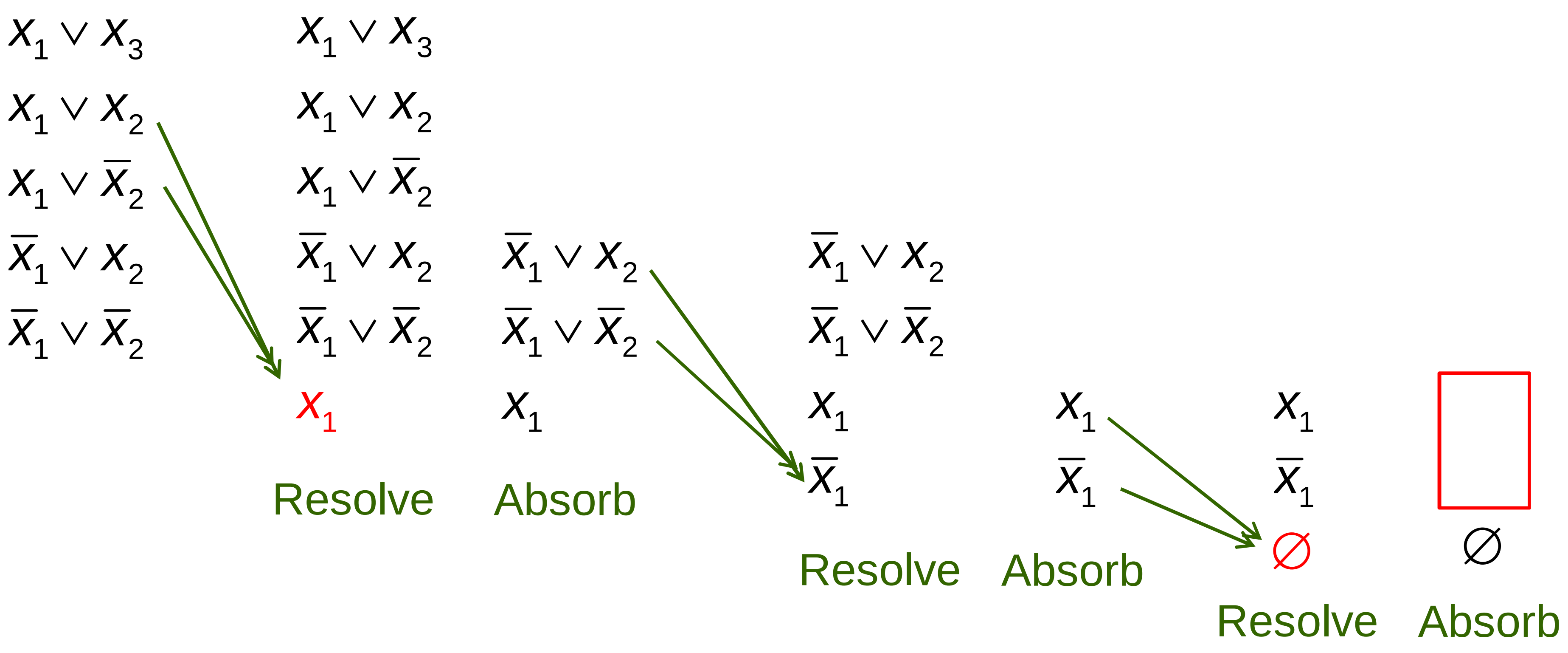
Absorb

x_1
$\bar{x}_1$
$\emptyset$

Resolve

Resolution Method

- Example of refutation



$x_1 \vee x_3$

$x_1 \vee x_2$

$x_1 \vee \bar{x}_2$

$\bar{x}_1 \vee x_2$

$\bar{x}_1 \vee \bar{x}_2$

$x_1 \vee x_3$

$x_1 \vee x_2$

$x_1 \vee \bar{x}_2$

$\bar{x}_1 \vee x_2$

$\bar{x}_1 \vee \bar{x}_2$

x_1

Resolve

$\bar{x}_1 \vee x_2$

$\bar{x}_1 \vee \bar{x}_2$

x_1

Absorb

$\bar{x}_1 \vee x_2$

$\bar{x}_1 \vee \bar{x}_2$

x_1

$\bar{x}_1$

Resolve

x_1

$\bar{x}_1$

Absorb

x_1

$\bar{x}_1$

$\varnothing$

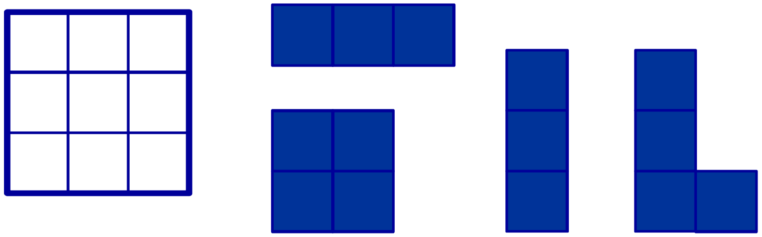
Resolve

$\varnothing$

Absorb

Resolution Method

- Example of prime implications
  - Simplify CNF expression derived earlier



$$x_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_4$$
$$\bar{x}_1 \vee x_2 \vee x_3 \vee \bar{x}_4$$
$$\bar{x}_1 \vee x_2 \vee \bar{x}_3 \vee x_4$$
$$\bar{x}_1 \vee x_2 \vee \bar{x}_3 \vee \bar{x}_4$$
$$\bar{x}_1 \vee \bar{x}_2 \vee x_3 \vee \bar{x}_4$$
$$\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee x_4$$
$$\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_4$$

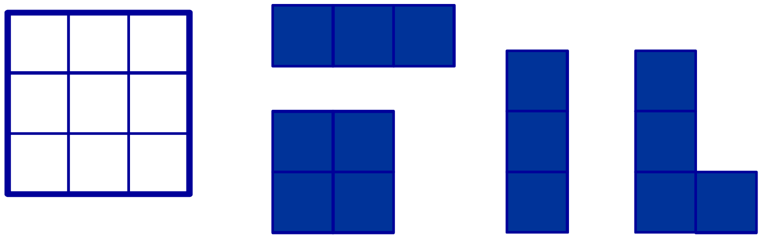
simplifies to

Prime implications

$$\bar{x}_1 \vee \bar{x}_3$$
$$\bar{x}_1 \vee \bar{x}_4$$
$$\bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_4$$

Resolution Method

- Example of prime implications
  - Simplify CNF expression derived earlier



$$x_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_4$$
$$\bar{x}_1 \vee x_2 \vee x_3 \vee \bar{x}_4$$
$$\bar{x}_1 \vee x_2 \vee \bar{x}_3 \vee x_4$$
$$\bar{x}_1 \vee x_2 \vee \bar{x}_3 \vee \bar{x}_4$$
$$\bar{x}_1 \vee \bar{x}_2 \vee x_3 \vee \bar{x}_4$$
$$\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee x_4$$
$$\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_4$$

simplifies to

Prime implications

$$\bar{x}_1 \vee \bar{x}_3$$
$$\bar{x}_1 \vee \bar{x}_4$$
$$\bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_4$$

Projection onto each x_i is {0,1}, because resolution fixes no variables. So the problem is domain consistent without reducing the domains {0,1}.

Resolution Method

- **Parallel resolution** resolves only on the last variable in each clause.

$$x_1 \vee x_2 \vee x_3$$

$$x_1 \vee x_2 \vee \bar{x}_3$$

Parallel resolvent → $x_1 \vee x_2$

$$x_1 \vee x_2 \vee x_3$$

$$x_1 \vee \bar{x}_2 \vee x_3$$

No parallel resolvent →

Resolution Method

- Parallel **absorption** will be used with parallel resolution.
  - Clause *C* **parallel-absorbs** *D* if: *C* is the empty clause, *C* = *D*, or the last literal of *C* occurs before last in *D*.

$$x_1 \vee x_2 \vee x_3$$

$$x_1 \vee x_2 \vee \bar{x}_3$$

$$x_1 \vee x_2$$

The parallel resolvent parallel-absorbs both parents because x_2 occurs before last in both.

Unit Resolution

- In **unit** resolution, at least one parent clause must be a **unit clause** (contains only 1 literal).
    - Runs in linear time.
    - Very efficient using **watched literals**.

Unit Resolution

- Example:

$$\begin{array}{lllllllll}
x_1 & & & & & & & & \\
 & & \bar{x}_2 & & & & & & \\
\bar{x}_1 & & & \vee & x_3 & & & & \\
\bar{x}_1 & \vee & x_2 & \vee & \bar{x}_3 & \vee & x_4 & & \\
\bar{x}_1 & \vee & x_2 & \vee & \bar{x}_3 & \vee & \bar{x}_4 & \vee & x_5 \\
 & & x_2 & \vee & \bar{x}_3 & & & \vee & \bar{x}_5
\end{array}$$

Unit Resolution

- Example:

$$
\begin{array}{ccccccccc}
x_1 & & & & & & & & \\
 & & \bar{x}_2 & & & & & & \\
\bar{x}_1 & & & \vee & x_3 & & & & \\
\bar{x}_1 & \vee & x_2 & \vee & \bar{x}_3 & \vee & x_4 & & \\
\bar{x}_1 & \vee & x_2 & \vee & \bar{x}_3 & \vee & \bar{x}_4 & \vee & x_5 \\
 & & x_2 & \vee & \bar{x}_3 & & & \vee & \bar{x}_5
\end{array}
$$

Unit Resolution

- Example:

$$\begin{array}{lllll}
x_1 & & & & \\
 & \bar{x}_2 & & & \\
\bar{x}_1 & & \vee\, x_3 & & \\
\bar{x}_1 & \vee\, x_2 & \vee\, \bar{x}_3 & \vee\, x_4 & \\
\bar{x}_1 & \vee\, x_2 & \vee\, \bar{x}_3 & \vee\, \bar{x}_4 & \vee\, x_5 \\
 & x_2 & \vee\, \bar{x}_3 & & \vee\, \bar{x}_5
\end{array}
\quad\longrightarrow\quad
\begin{array}{llll}
\bar{x}_2 & & & \\
 & \vee\, x_3 & & \\
x_2 & \vee\, \bar{x}_3 & \vee\, x_4 & \\
x_2 & \vee\, \bar{x}_3 & \vee\, \bar{x}_4 & \vee\, x_5 \\
x_2 & \vee\, \bar{x}_3 & & \vee\, \bar{x}_5
\end{array}$$

Unit Resolution

- Example:

$$\begin{array}{lllll}
\bar{x}_2 & & & & \\
 & \vee\, x_3 & & & \\
x_2 & \vee\, \bar{x}_3 & \vee\, x_4 & & \\
x_2 & \vee\, \bar{x}_3 & \vee\, \bar{x}_4 & \vee\, x_5 & \\
x_2 & \vee\, \bar{x}_3 & & \vee\, \bar{x}_5 &
\end{array}$$

Unit Resolution

- Example:

$$\begin{array}{lcl} & & \bar{x}_2 \\ x_3 & & \quad \vee x_3 \\ \bar{x}_3 \vee x_4 & \longleftarrow & x_2 \vee \bar{x}_3 \vee x_4 \\ \bar{x}_3 \vee \bar{x}_4 \vee x_5 & & x_2 \vee \bar{x}_3 \vee \bar{x}_4 \vee x_5 \\ \bar{x}_3 \qquad \vee \bar{x}_5 & & x_2 \vee \bar{x}_3 \qquad \vee \bar{x}_5 \end{array}$$

Unit Resolution

- Example:

$$\begin{array}{l}
x_3 \\
\bar{x}_3 \vee x_4 \\
\bar{x}_3 \vee \bar{x}_4 \vee x_5 \\
\bar{x}_3 \qquad\quad \vee \bar{x}_5
\end{array}$$

Unit Resolution

- Example:

$$\begin{array}{lllll} x_3 & & & & \\ \bar{x}_3 & \vee\, x_4 & & & x_4 \\ \bar{x}_3 & \vee\, \bar{x}_4 & \vee\, x_5 & \longrightarrow & \bar{x}_4 \vee x_5 \\ \bar{x}_3 & & \vee\, \bar{x}_5 & & \qquad \bar{x}_5 \end{array}$$

Unit Resolution

- Example:

$$\begin{array}{l} x_4 \\ \bar{x}_4 \vee x_5 \\ \qquad\;\; \bar{x}_5 \end{array}$$

Unit Resolution

- Example:

$$\emptyset \longleftarrow \begin{array}{l} x_4 \\ \bar{x}_4 \vee x_5 \\ \qquad\;\; \bar{x}_5 \end{array}$$

Unit Resolution

- Now use watched literals.

$$
\begin{array}{lllllllll}
x_1 & & & & & & & & (a) \\
 & & \bar{x}_2 & & & & & & (b) \\
\bar{x}_1 & & & \vee & x_3 & & & & (c) \\
\bar{x}_1 & \vee & x_2 & \vee & \bar{x}_3 & \vee & x_4 & & (d) \\
\bar{x}_1 & \vee & x_2 & \vee & \bar{x}_3 & \vee & \bar{x}_4 & \vee\; x_5 & (e) \\
 & & x_2 & \vee & \bar{x}_3 & & & \vee\; \bar{x}_5 & (f)
\end{array}
$$

Unit Resolution

- Now use watched literals.

$$\begin{array}{lllllllll}
x_1 & & & & & & & & (a) \\
 & & \bar{x}_2 & & & & & & (b) \\
\bar{x}_1 & & & \vee & x_3 & & & & (c) \\
\bar{x}_1 & \vee & x_2 & \vee & \bar{x}_3 & \vee & x_4 & & (d) \\
\bar{x}_1 & \vee & x_2 & \vee & \bar{x}_3 & \vee & \bar{x}_4 & \vee\ x_5 & (e) \\
 & & x_2 & \vee & \bar{x}_3 & & & \vee\ \bar{x}_5 & (f)
\end{array}$$

Arbitrarily select 2 watched literals in each clause

Unit Resolution

- Now use watched literals.

$$\begin{array}{lllllr}
x_1 & & & & & (a) \\
 & \bar{x}_2 & & & & (b) \\
\bar{x}_1 & & \vee\, x_3 & & & (c) \\
\bar{x}_1 & \vee\, x_2 & \vee\, \bar{x}_3 & \vee\, x_4 & & (d) \\
\bar{x}_1 & \vee\, x_2 & \vee\, \bar{x}_3 & \vee\, \bar{x}_4 & \vee\, x_5 & (e) \\
 & x_2 & \vee\, \bar{x}_3 & & \vee\, \bar{x}_5 & (f)
\end{array}$$

Arbitrarily select 2 watched literals in each clause.

If unit resolution reduces a clause to a single literal, it must at some point fix **one** of the watched literals.

So it suffices to examine a clause only when one of its watched literals is fixed.

Unit Resolution

- Now use watched literals.

$$\begin{array}{ll}
x_1 & (a) \\
\bar{x}_2 & (b) \\
\bar{x}_1 \vee x_3 & (c) \\
\bar{x}_1 \vee x_2 \vee \bar{x}_3 \vee x_4 & (d) \\
\bar{x}_1 \vee x_2 \vee \bar{x}_3 \vee \bar{x}_4 \vee x_5 & (e) \\
x_2 \vee \bar{x}_3 \vee \bar{x}_5 & (f)
\end{array}$$

Keep list of watched literals:

x_1	–	a	$\bar{x}_1$	–	c,d
x_2	–	d,f	$\bar{x}_2$	–	b
x_3	–	c	$\bar{x}_3$	–	e
x_4	–		$\bar{x}_4$	–	e
x_5	–		$\bar{x}_5$	–	f

Unit Resolution

- Now use watched literals.

$$\begin{array}{ll}
x_1 & (a) \\
\bar{x}_2 & (b) \\
\bar{x}_1 \vee x_3 & (c) \\
\bar{x}_1 \vee x_2 \vee \bar{x}_3 \vee x_4 & (d) \\
\bar{x}_1 \vee x_2 \vee \bar{x}_3 \vee \bar{x}_4 \vee x_5 & (e) \\
x_2 \vee \bar{x}_3 \vee \bar{x}_5 & (f)
\end{array}$$

To resolve on x_1, examine only the clauses in which $\bar{x}_1$ is a watched literal (enormous savings).

For absorption, check clauses in which x_1 is a watched literal (none here)

x_1	–	a	$\bar{x}_1$	–	c,d
x_2	–	d,f	$\bar{x}_2$	–	b
x_3	–	c	$\bar{x}_3$	–	e
x_4	–		$\bar{x}_4$	–	e
x_5	–		$\bar{x}_5$	–	f

Unit Resolution

- Now use watched literals.

$$
\begin{array}{lr}
x_1 & (a) \\
\bar{x}_2 & (b) \\
x_3 & (c) \\
x_2 \vee \bar{x}_3 \vee x_4 & (d) \\
\bar{x}_1 \vee x_2 \vee \bar{x}_3 \vee \bar{x}_4 \vee x_5 & (e) \\
x_2 \vee \bar{x}_3 \vee \bar{x}_5 & (f)
\end{array}
$$

Arbitrarily select a new watched literal in clause *d.*

x_1	–	*a*	$\bar{x}_1$	–	*c*,*d*
x_2	–	*d*,*f*	$\bar{x}_2$	–	*b*
x_3	–	*c*	$\bar{x}_3$	–	*e*
x_4	–		$\bar{x}_4$	–	*e*
x_5	–		$\bar{x}_5$	–	*f*

Unit Resolution

- Now use watched literals.

$$(a)$$

$$\bar{x}_2 \qquad (b)$$

$$x_3 \qquad (c)$$

$$x_2 \vee \bar{x}_3 \vee x_4 \qquad (d)$$

$$\bar{x}_1 \vee x_2 \vee \bar{x}_3 \vee \bar{x}_4 \vee x_5 \qquad (e)$$

$$x_2 \vee \bar{x}_3 \qquad \vee \bar{x}_5 \qquad (f)$$

$\bar{x}_1$

Keep list of fixed variables:

x_1

x_1	–	a	$\bar{x}_1$	–	c,d
x_2	–	d,f	$\bar{x}_2$	–	b
x_3	–	c	$\bar{x}_3$	–	e
x_4	–		$\bar{x}_4$	–	e
x_5	–		$\bar{x}_5$	–	f

Unit Resolution

- Now use watched literals.

$$(a)$$

$$\bar{x}_2 \qquad (b)$$

$\bar{x}_1$

$$x_3 \qquad (c)$$

$$x_2 \vee \bar{x}_3 \vee x_4 \qquad (d)$$

$$\bar{x}_1 \vee x_2 \vee \bar{x}_3 \vee \bar{x}_4 \vee x_5 \qquad (e)$$

$$x_2 \vee \bar{x}_3 \vee \bar{x}_5 \qquad (f)$$

Update list of watched literals:

x_1	–		$\bar{x}_1$	–	
x_2	–	d,f	$\bar{x}_2$	–	b
x_3	–	c	$\bar{x}_3$	–	e
x_4	–	d	$\bar{x}_4$	–	e
x_5	–		$\bar{x}_5$	–	f

Keep list of fixed variables:

x_1

Unit Resolution

- Now use watched literals.

	(a)
$\bar{x}_2$	(b)
x_3	(c)
$x_2 \vee \bar{x}_3 \vee x_4$	(d)
$\bar{x}_1 \vee x_2 \vee \bar{x}_3 \vee \bar{x}_4 \vee x_5$	(e)
$x_2 \vee \bar{x}_3 \vee \bar{x}_5$	(f)

Resolve on x_2

$\bar{x}_1$

Keep list of fixed variables:

x_1

x_1	–		$\bar{x}_1$	–	
x_2	–	d,f	$\bar{x}_2$	–	b
x_3	–	c	$\bar{x}_3$	–	e
x_4	–	d	$\bar{x}_4$	–	e
x_5	–		$\bar{x}_5$	–	f

Unit Resolution

- Now use watched literals.

$$\begin{array}{rr}
 & (a) \\
 & (b) \\
x_3 & (c) \\
\bar{x}_3 \vee x_4 & (d) \\
\bar{x}_1 \vee x_2 \vee \bar{x}_3 \vee \bar{x}_4 \vee x_5 & (e) \\
\bar{x}_3 \qquad\quad \vee \bar{x}_5 & (f)
\end{array}$$

Resolve on x_2

Update list of watched literals:

x_1	–		$\bar{x}_1$	–	
x_2	–		$\bar{x}_2$	–	
x_3	–	c	$\bar{x}_3$	–	d,e,f
x_4	–	d	$\bar{x}_4$	–	e
x_5	–		$\bar{x}_5$	–	f

Keep list of fixed variables:

$x_1, \bar{x}_2$

Unit Resolution

- Now use watched literals.

	(a)
	(b)
x_3	(c)
$\bar{x}_3 \vee x_4$	(d)
$\bar{x}_1 \vee x_2 \vee \bar{x}_3 \vee \bar{x}_4 \vee x_5$	(e)
$\bar{x}_3 \vee \bar{x}_5$	(f)

Resolve on x_3

Update list of watched literals:

x_1	–		$\bar{x}_1$	–	
x_2	–		$\bar{x}_2$	–	
x_3	–	c	$\bar{x}_3$	–	d,e,f
x_4	–	d	$\bar{x}_4$	–	e
x_5	–		$\bar{x}_5$	–	f

Keep list of fixed variables:

$x_1, \bar{x}_2$

Unit Resolution

- Now use watched literals.

(a)

(b)

(c)

$x_4 \qquad (d)$

$\bar{x}_1 \vee x_2 \qquad \vee \bar{x}_4 \vee x_5 \qquad (e)$

$\vee \bar{x}_5 \qquad (f)$

Resolve on x_3

Update list of watched literals:

x_1	–		$\bar{x}_1$	–	
x_2	–		$\bar{x}_2$	–	
x_3	–		$\bar{x}_3$	–	
x_4	–	d	$\bar{x}_4$	–	e
x_5	–	e	$\bar{x}_5$	–	f

Keep list of fixed variables:

$x_1, \bar{x}_2, x_3$

Unit Resolution

- Now use watched literals.

(a)

(b)

(c)

$x_4 \qquad (d)$

$\bar{x}_1 \vee x_2 \qquad \vee \bar{x}_4 \vee x_5 \qquad (e)$

$\vee \bar{x}_5 \qquad (f)$

Resolve on x_4

We know that (e) becomes a unit clause because of list of fixed variables

Keep list of fixed variables:

$x_1, \bar{x}_2, x_3$

x_1	–		$\bar{x}_1$	–	
x_2	–		$\bar{x}_2$	–	
x_3	–		$\bar{x}_3$	–	
x_4	–	d	$\bar{x}_4$	–	e
x_5	–	e	$\bar{x}_5$	–	f

Unit Resolution

- Now use watched literals.

	(a)
	(b)
	(c)
	(d)
x_5	(e)
$\bar{x}_5$	(f)

Resolve on x_4

Update list of watched literals:

x_1	–		$\bar{x}_1$	–	
x_2	–		$\bar{x}_2$	–	
x_3	–		$\bar{x}_3$	–	
x_4	–		$\bar{x}_4$	–	
x_5	–	e	$\bar{x}_5$	–	f

Keep list of fixed variables:

$x_1, \bar{x}_2, x_3, x_4$

Unit Resolution

- Now use watched literals.

(a)

(b)

(c)

(d)

x_5 (e)

$\bar{x}_5$ (f)

Resolve on x_5 and derive empty clause.

Keep list of fixed variables:

$x_1, \bar{x}_2, x_3, x_4$

x_1	–		$\bar{x}_1$	–	
x_2	–		$\bar{x}_2$	–	
x_3	–		$\bar{x}_3$	–	
x_4	–		$\bar{x}_4$	–	
x_5	–	e	$\bar{x}_5$	–	f

DPLL

- The **DPLL** (Davis-Putnam-Loveland-Logemann) algorithm combines branching with unit resolution.
    - Unit resolution serves as a propagation algorithm at each node of the search tree.

DPLL

- The **DPLL** (Davis-Putnam-Loveland-Logemann) algorithm combines branching with unit resolution.
    - Unit resolution serves as a propagation algorithm at each node of the search tree.
- **CDCL** (conflict-directed clause learning) uses nogoods to direct the search and reduce backtracking.
    - An old idea in AI.
    - The best solvers generally use DPLL + CDCL (and many tricks).

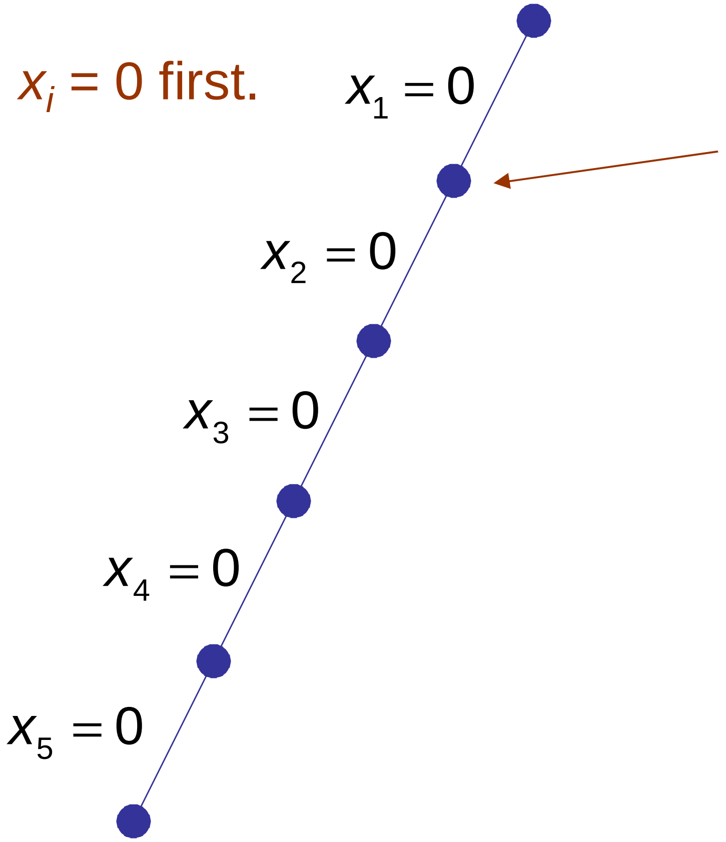
DPLL

- Example: Hiring problem

$$
\begin{array}{lcl}
x_1 \vee x_5 \vee x_6 & \equiv & x_1 \vee x_5 \vee x_6 \\
x_6 \rightarrow (x_1 \vee x_5) & \equiv & \overline{x}_6 \vee x_1 \vee x_5 \\
x_5 \rightarrow (x_2 \vee x_6) & \equiv & \overline{x}_5 \vee x_2 \vee x_6 \\
(x_5 \wedge x_6) \rightarrow x_2 & \equiv & (\overline{x}_5 \vee x_2) \wedge (\overline{x}_6 \vee x_2) \\
(x_1 \vee x_2) \rightarrow (x_3 \vee x_4) & \equiv & (\overline{x}_1 \vee x_3 \vee x_4) \wedge (\overline{x}_2 \vee x_3 \vee x_4) \\
(x_3 \vee x_4) \rightarrow (\overline{x}_1 \wedge \overline{x}_2) & \equiv & (\overline{x}_3 \vee \overline{x}_1) \wedge (\overline{x}_3 \vee \overline{x}_2) \wedge (\overline{x}_4 \vee \overline{x}_1) \wedge (\overline{x}_4 \vee \overline{x}_2)
\end{array}
$$

Simple DPLL

Branch by trying $x_i = 0$ first.

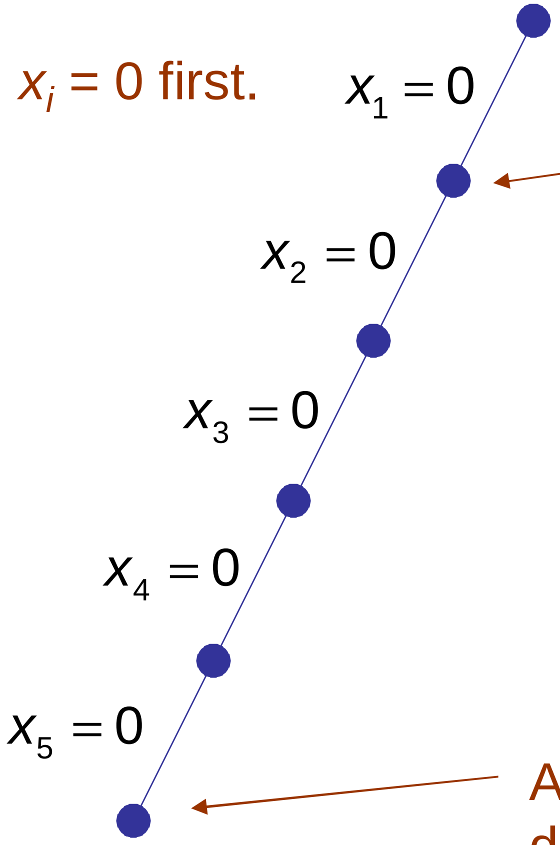


Apply unit resolution after adding unit clause $\bar{x}_1$

Pass simplified clause set to child node.

Simple DPLL

Branch by trying $x_i = 0$ first.

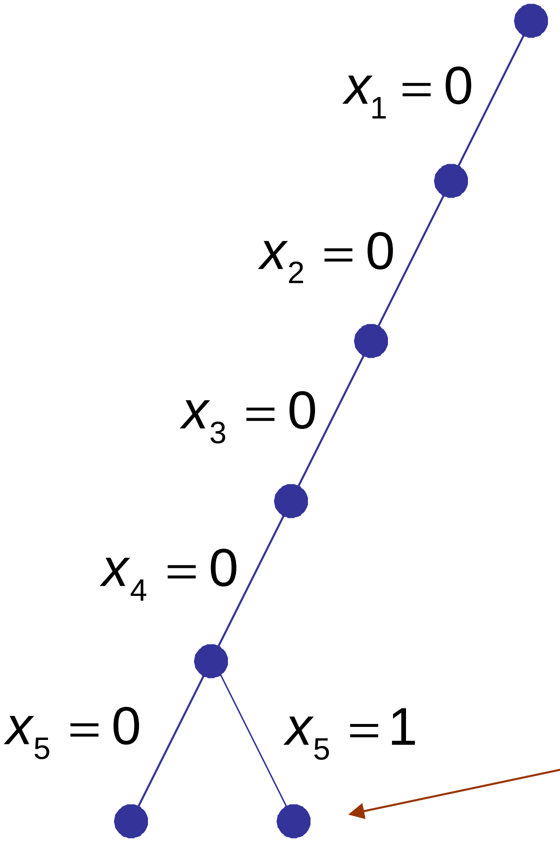


Apply unit resolution after adding unit clause $\bar{x}_1$

Pass simplified clause set to child node.

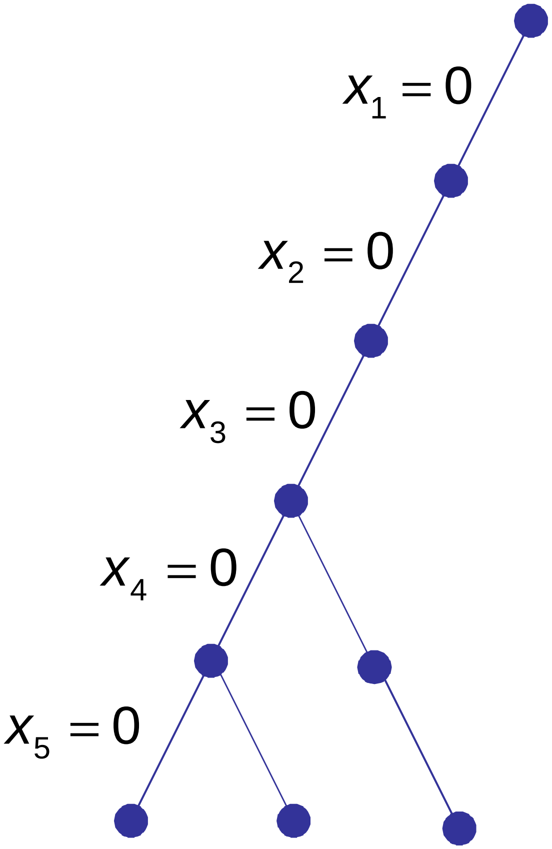
At this point, unit resolution derives the empty clause.

Simple DPLL



Backtrack and take this branch (depth-first search)

Simple DPLL



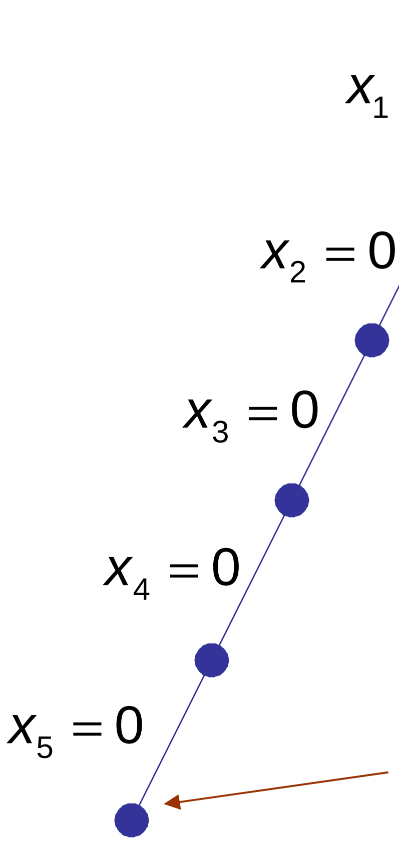
Continue in this fashion until search is exhaustive.

Solution is never found.

DPLL with Conflict Clauses

- Use **conflict clauses** to direct the search.
  - A conflict clause is a nogood that rules out a partial assignment that caused infeasibility.

DPLL with Conflict Clauses

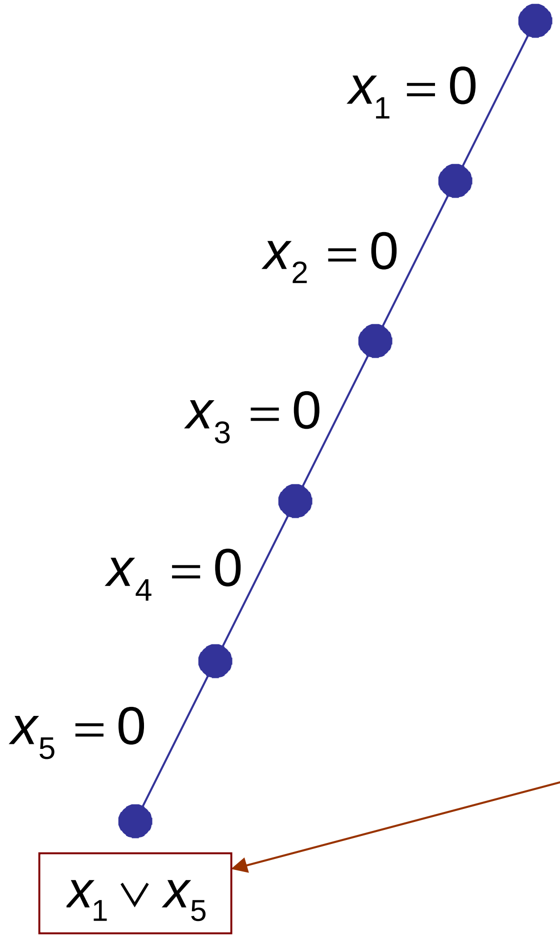


Again branch to here. Unit resolution proves infeasibility.

Setting $(x_1,x_5) = (0,0)$ is enough for unit resolution to prove infeasibility.

How do we know? To be discussed… .

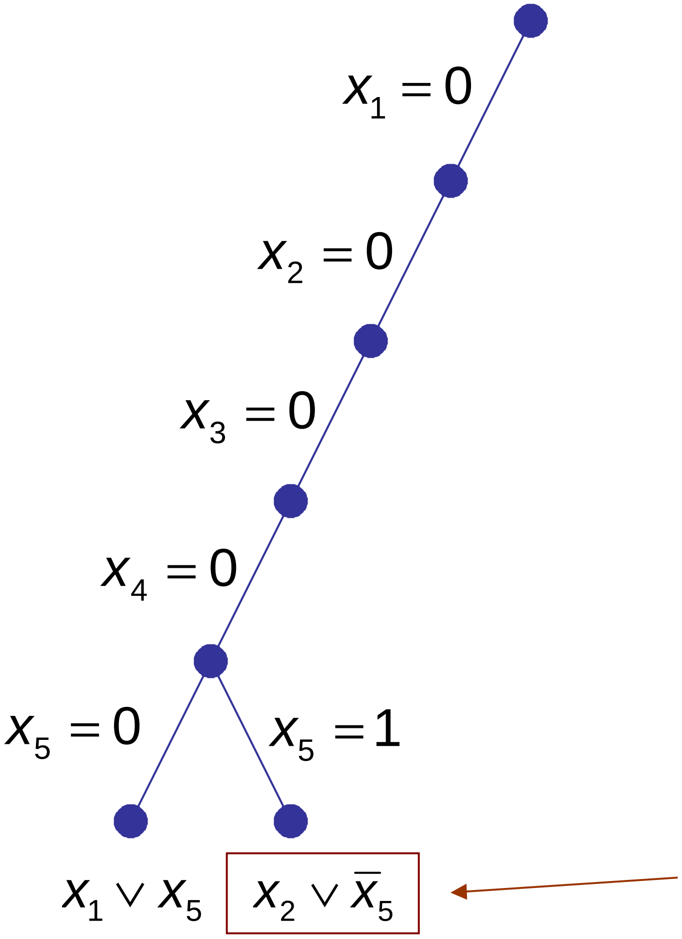
DPLL with Conflict Clauses



Generate **conflict clause** to rule out partial assignment that created infeasibility.

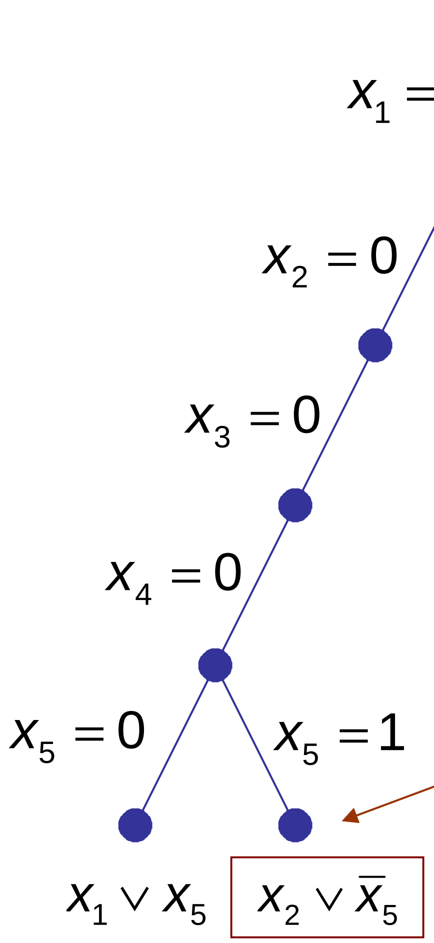
Future branching must satisfy the conflict clause.

DPLL with Conflict Clauses



Branch to here and generate another conflict clause

DPLL with Conflict Clauses



Actually, we can forget about branching and simply solve the **nogood set** $\{x_1 \vee x_5\}$.

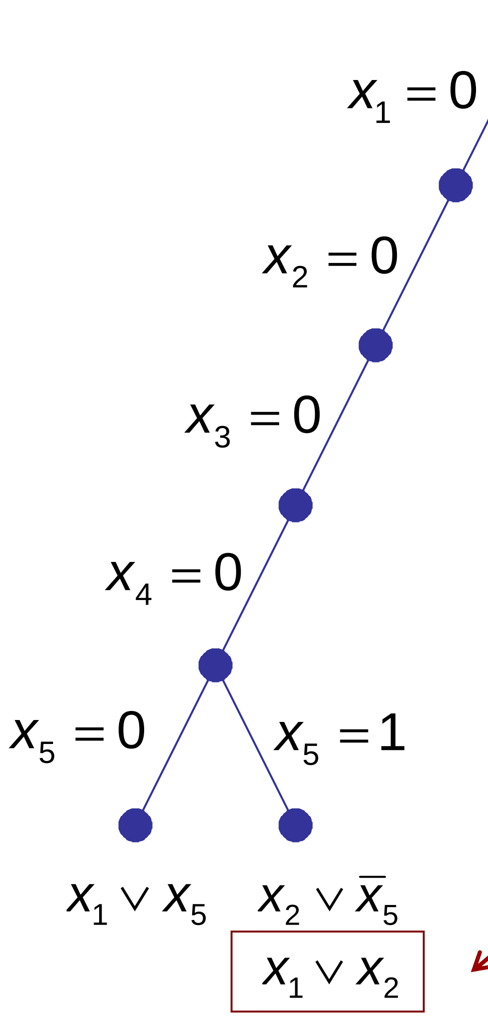
We will make sure the nogood set can always be solved by forward checking.

Here, we try $x_i = 0$ first. This yields the next leaf node.

Branch to here and generate another conflict clause

Slide 436

DPLL with Conflict Clauses

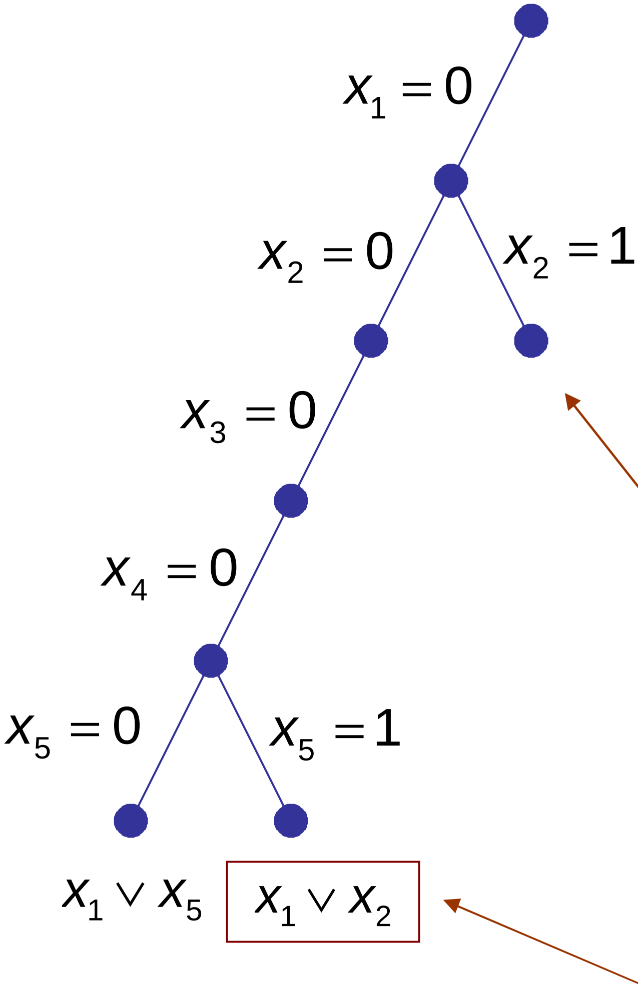


Now the nogood set contains

$x_1 \vee x_5 \qquad x_2 \vee \bar{x}_5$

Apply **parallel resolution** and **parallel absorption** to obtain simplified nogood set $x_1 \vee x_2$

DPLL with Conflict Clauses



Now solve nogood set by forward checking.

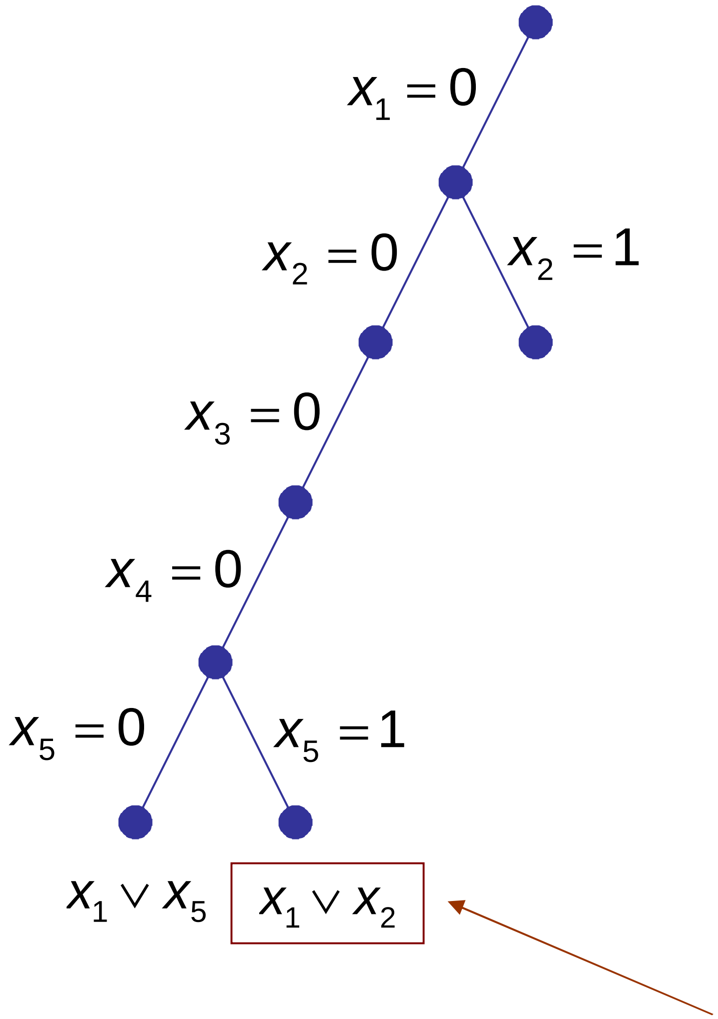
Because we processed nogoods with parallel resolution, we can solve it by forward checking (if feasible).

Perform unit resolution after each variable is fixed, which yields empty clause after fixing 2 variables.

Nogood set

x_1

DPLL with Conflict Clauses



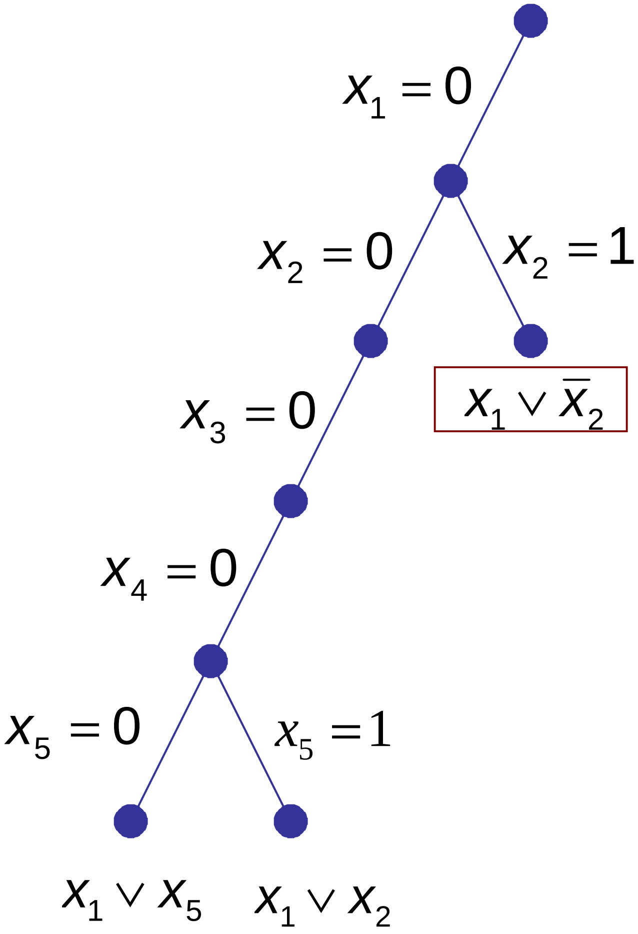
When backtracking, there is no need to retrace how watched literals were assigned.

This is a **lazy** data structure.

Nogood set

x_1

DPLL with Conflict Clauses

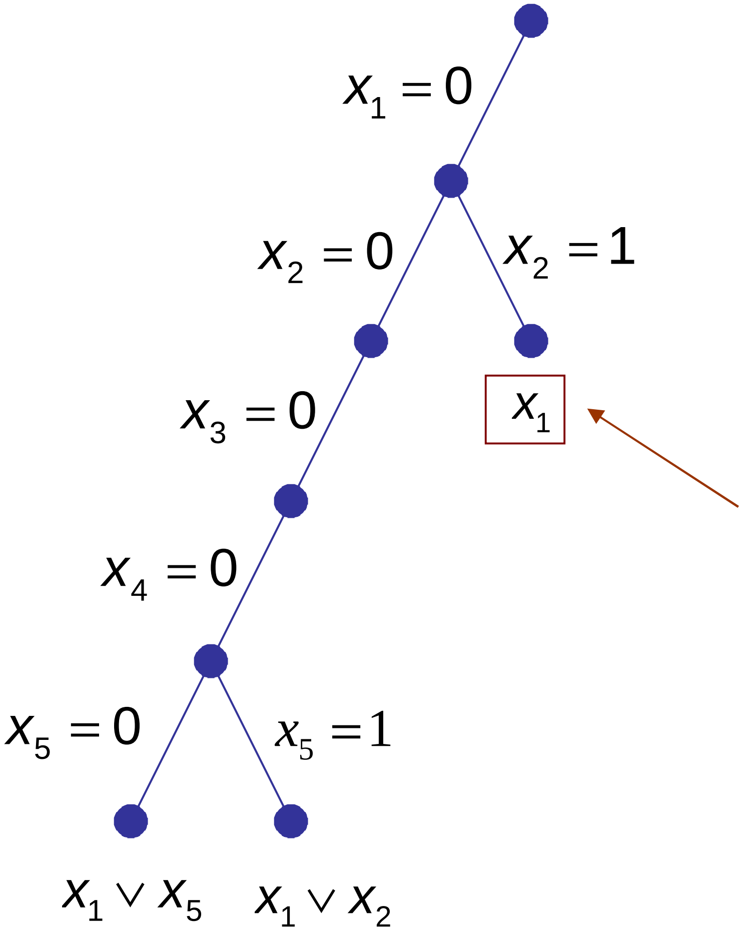


General new nogood to obtain nogood set

$x_1 \vee \bar{x}_2$ $x_1 \vee x_2$

x_1

DPLL with Conflict Clauses



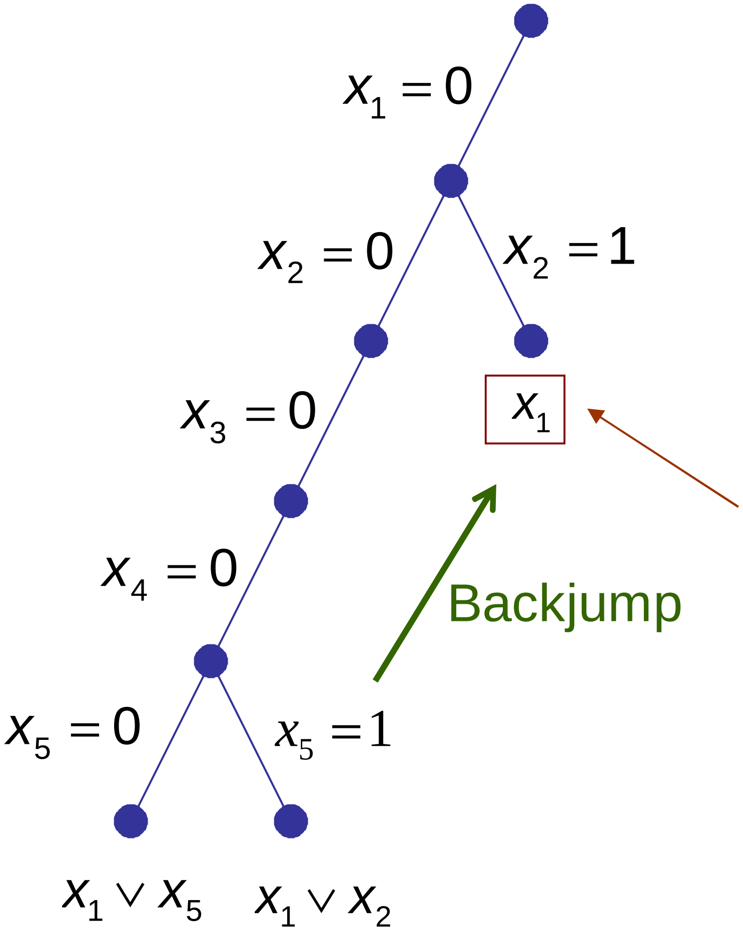
General new nogood to obtain nogood set

$x_1 \vee \bar{x}_2 \qquad x_1 \vee x_2$

Apply parallel resolution to obtain simplified nogood set

x_1

DPLL with Conflict Clauses



General new nogood to obtain nogood set

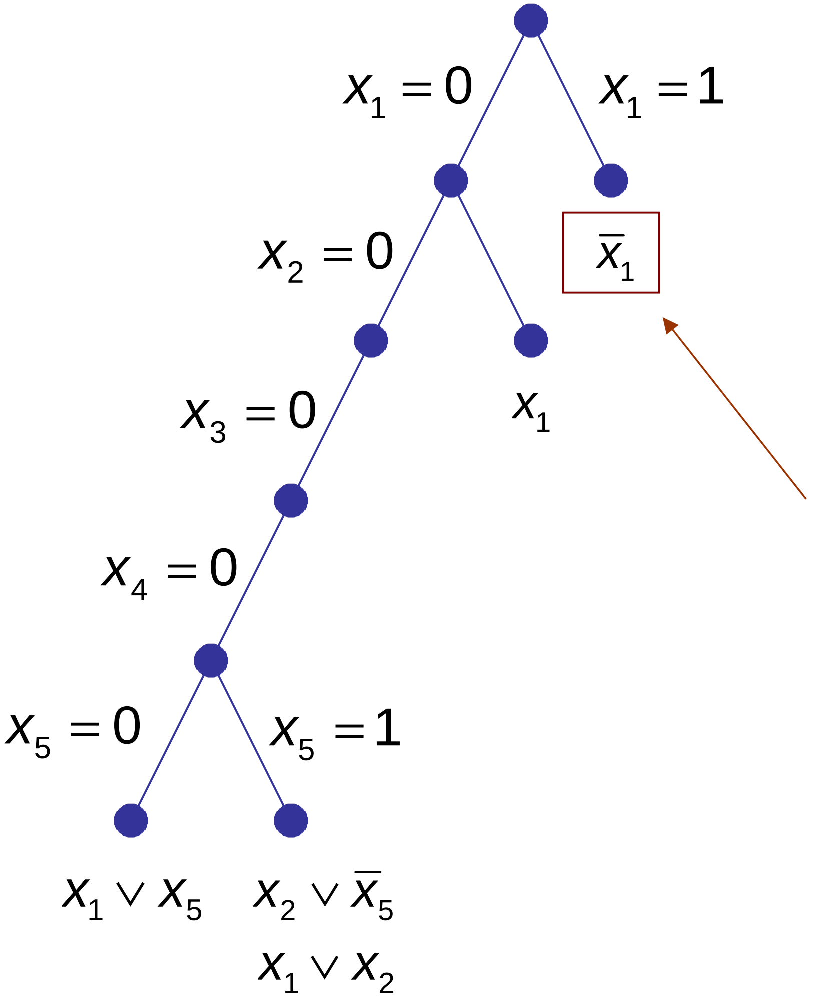
$x_1 \vee \bar{x}_2 \qquad x_1 \vee x_2$

Apply parallel resolution to obtain simplified nogood set.

x_1

Parallel resolution is always fast in this context.

DPLL with Conflict Clauses

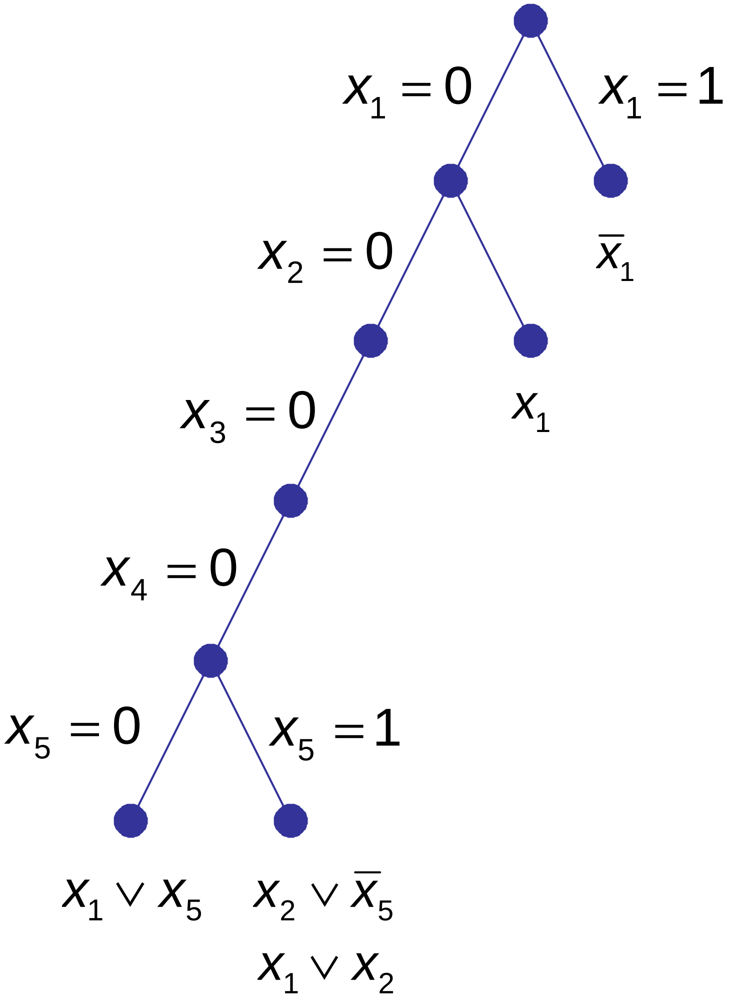


Again solve nogood set.

Unit resolution derives empty clause after fixing only x_1

Generate nogood.

DPLL with Conflict Clauses



Now the nogood set is

$x_1 \qquad \bar{x}_1$

Parallel resolution derives the empty clause.

Forward checking cannot solve the nogood set, so the search is complete.

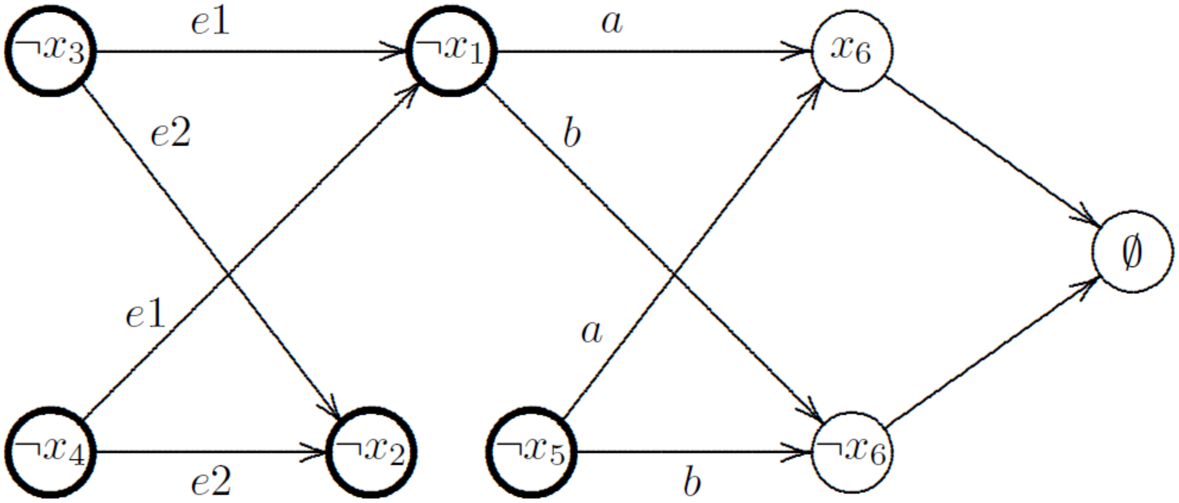
There is no solution.

Implication Graph

- Conflict clauses are identified by analyzing the **implication graph**.

Implication Graph

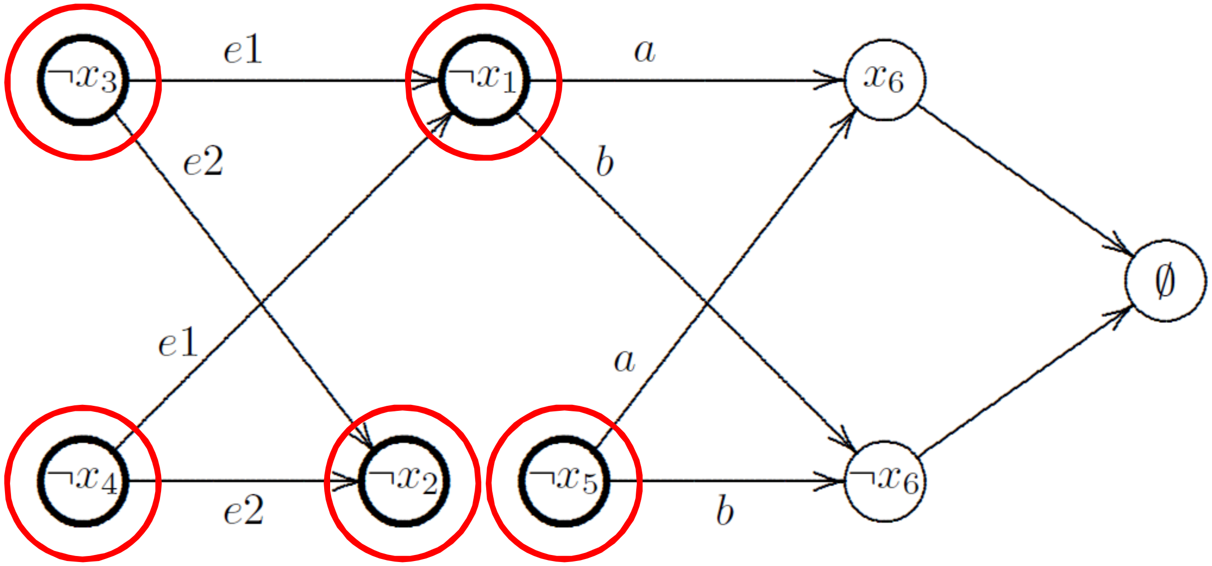
- Hiring example: Build conflict graph at first leaf node.



$x_1 \vee x_5 \vee x_6$	(a)
$x_1 \vee x_5 \vee \bar{x}_6$	(b)
$x_2 \vee \bar{x}_5 \vee x_6$	(c)
$x_2 \vee \bar{x}_5 \vee \bar{x}_6$	(d)
$\bar{x}_1 \vee x_3 \vee x_4$	$(e1)$
$\bar{x}_2 \vee x_3 \vee x_4$	$(e2)$
$\bar{x}_3 \vee \bar{x}$	$(f1)$
$\bar{x}_3 \vee \bar{x}_2$	$(f2)$
$\bar{x}_4 \vee \bar{x}_1$	$(f3)$
$\bar{x}_4 \vee \bar{x}_2$	$(f4)$

Implication Graph

- Hiring example: Build conflict graph at first leaf node.

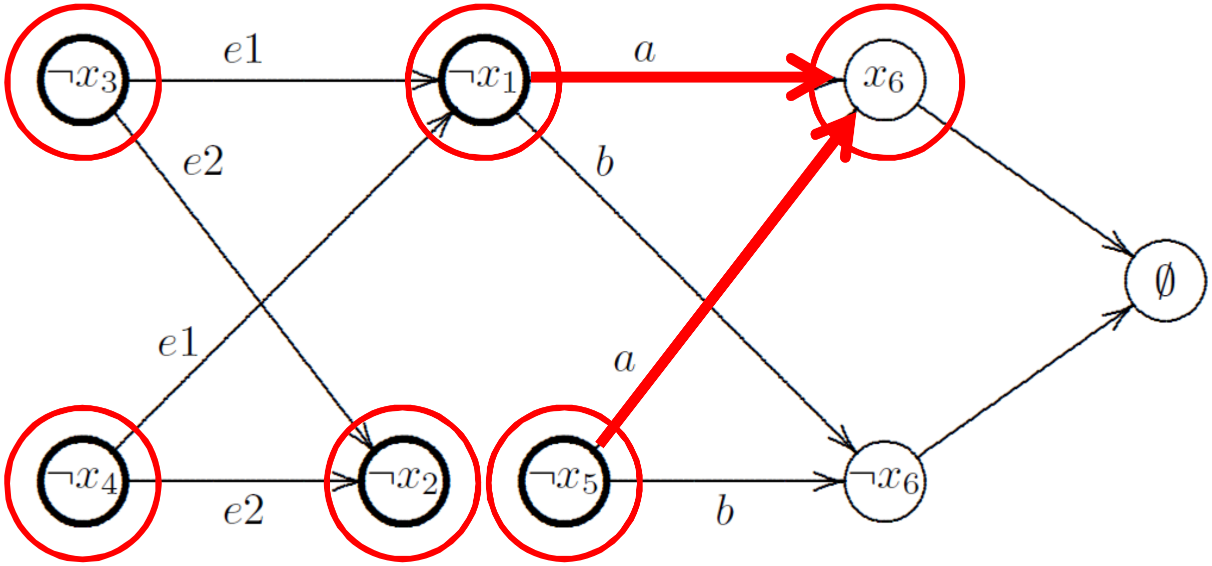


$x_1 \vee x_5 \vee x_6$	(a)
$x_1 \vee x_5 \vee \bar{x}_6$	(b)
$x_2 \vee \bar{x}_5 \vee x_6$	(c)
$x_2 \vee \bar{x}_5 \vee \bar{x}_6$	(d)
$\bar{x}_1 \vee x_3 \vee x_4$	$(e1)$
$\bar{x}_2 \vee x_3 \vee x_4$	$(e2)$
$\bar{x}_3 \vee \bar{x}$	$(f1)$
$\bar{x}_3 \vee \bar{x}_2$	$(f2)$
$\bar{x}_4 \vee \bar{x}_1$	$(f3)$
$\bar{x}_4 \vee \bar{x}_2$	$(f4)$

Add a vertex for every branching literal.

Implication Graph

- Hiring example: Build conflict graph at first leaf node.

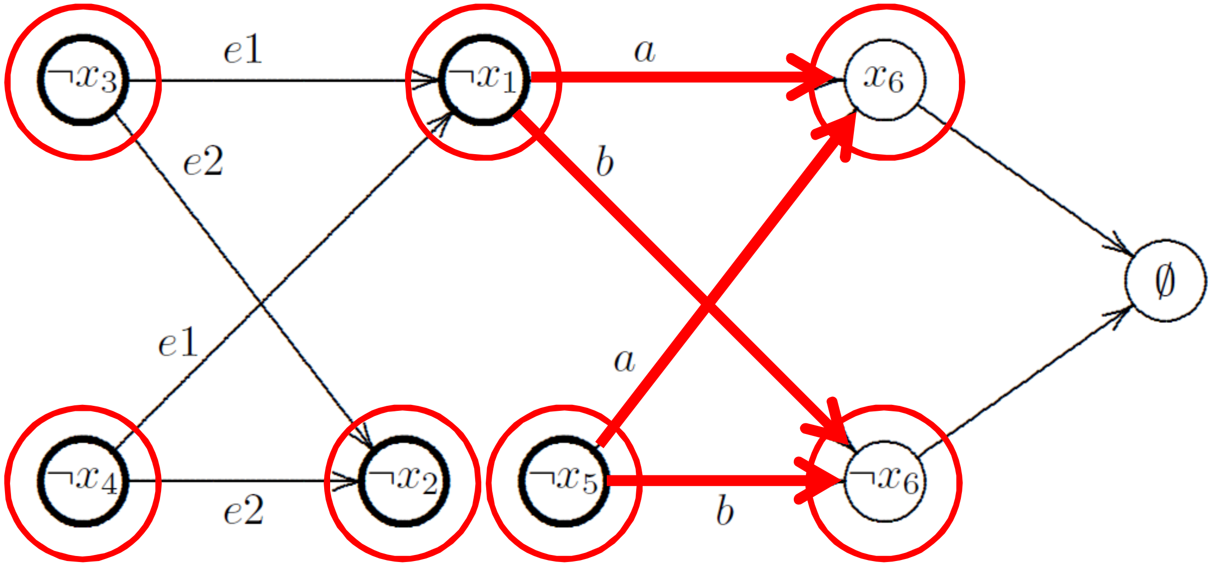


$x_1 \vee x_5 \vee x_6$	(a)
$x_1 \vee x_5 \vee \bar{x}_6$	(b)
$x_2 \vee \bar{x}_5 \vee x_6$	(c)
$x_2 \vee \bar{x}_5 \vee \bar{x}_6$	(d)
$\bar{x}_1 \vee x_3 \vee x_4$	$(e1)$
$\bar{x}_2 \vee x_3 \vee x_4$	$(e2)$
$\bar{x}_3 \vee \bar{x}$	$(f1)$
$\bar{x}_3 \vee \bar{x}_2$	$(f2)$
$\bar{x}_4 \vee \bar{x}_1$	$(f3)$
$\bar{x}_4 \vee \bar{x}_2$	$(f4)$

Add edges for clause (a), which is $(\bar{x}_1 \wedge \bar{x}_5) \rightarrow x_6$
Both antecedents are vertices.

Implication Graph

- Hiring example: Build conflict graph at first leaf node.

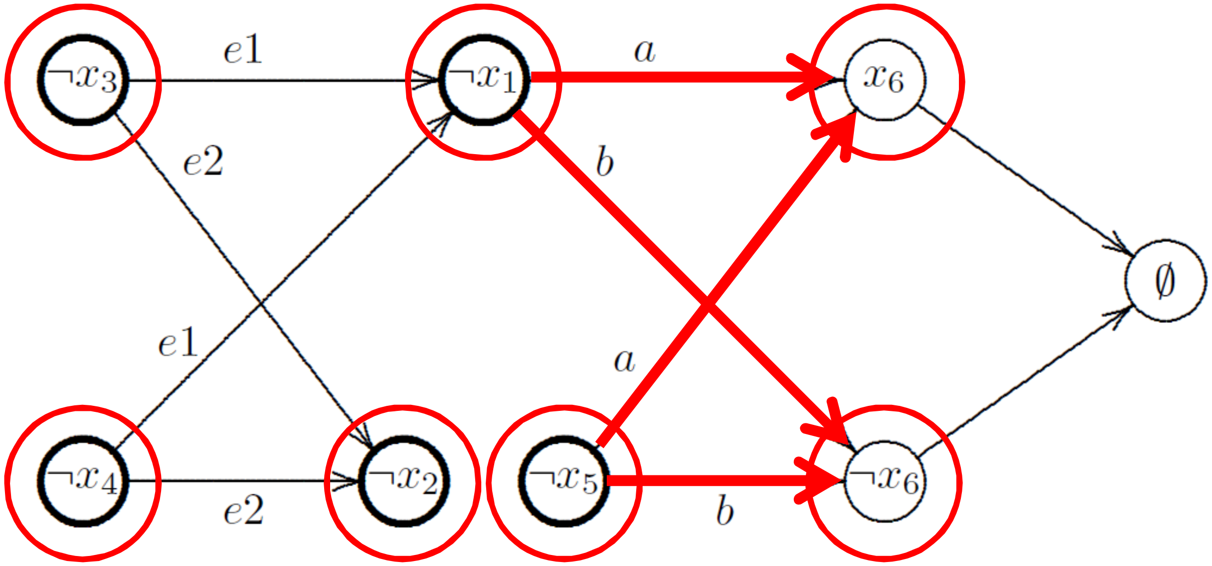


Add edges for clause (b), which is $(\bar{x}_1 \wedge \bar{x}_5) \rightarrow \bar{x}_6$

$x_1 \vee x_5 \vee x_6$	(a)
$x_1 \vee x_5 \vee \bar{x}_6$	(b)
$x_2 \vee \bar{x}_5 \vee x_6$	(c)
$x_2 \vee \bar{x}_5 \vee \bar{x}_6$	(d)
$\bar{x}_1 \vee x_3 \vee x_4$	$(e1)$
$\bar{x}_2 \vee x_3 \vee x_4$	$(e2)$
$\bar{x}_3 \vee \bar{x}$	$(f1)$
$\bar{x}_3 \vee \bar{x}_2$	$(f2)$
$\bar{x}_4 \vee \bar{x}_1$	$(f3)$
$\bar{x}_4 \vee \bar{x}_2$	$(f4)$

Implication Graph

- Hiring example: Build conflict graph at first leaf node.

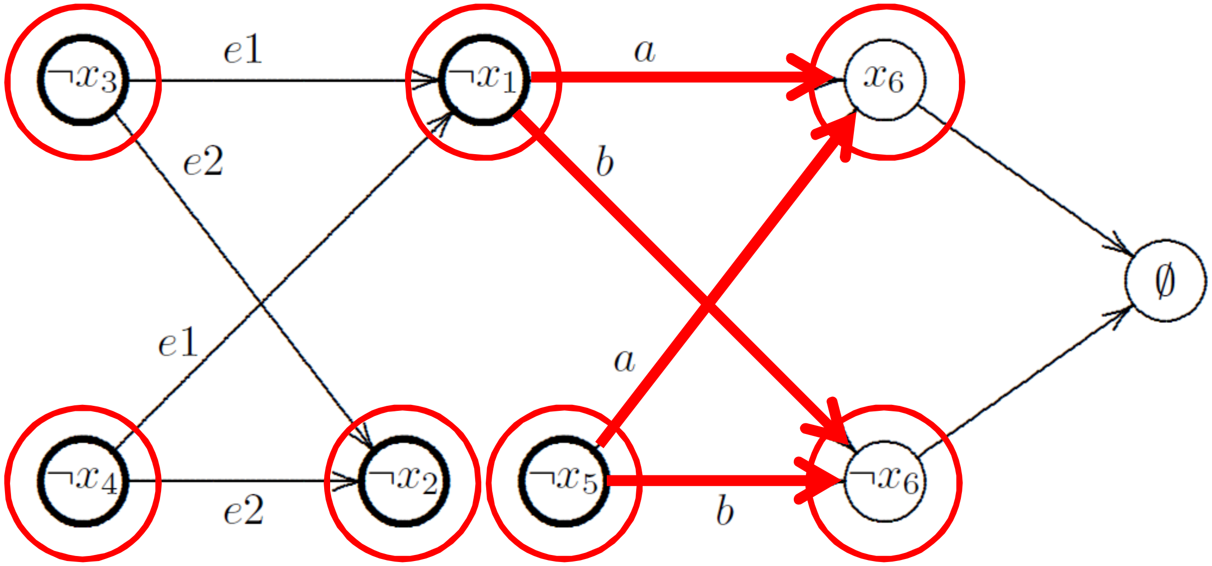


No edges for clause (c)

$x_1 \vee x_5 \vee x_6$	(a)
$x_1 \vee x_5 \vee \bar{x}_6$	(b)
$x_2 \vee \bar{x}_5 \vee x_6$	(c)
$x_2 \vee \bar{x}_5 \vee \bar{x}_6$	(d)
$\bar{x}_1 \vee x_3 \vee x_4$	$(e1)$
$\bar{x}_2 \vee x_3 \vee x_4$	$(e2)$
$\bar{x}_3 \vee \bar{x}$	$(f1)$
$\bar{x}_3 \vee \bar{x}_2$	$(f2)$
$\bar{x}_4 \vee \bar{x}_1$	$(f3)$
$\bar{x}_4 \vee \bar{x}_2$	$(f4)$

Implication Graph

- Hiring example: Build conflict graph at first leaf node.



$x_1 \vee x_5 \vee x_6$ (a)

$x_1 \vee x_5 \vee \bar{x}_6$ (b)

$x_2 \vee \bar{x}_5 \vee x_6$ (c)

$x_2 \vee \bar{x}_5 \vee \bar{x}_6$ (d)

$\bar{x}_1 \vee x_3 \vee x_4$ (e1)

$\bar{x}_2 \vee x_3 \vee x_4$ (e2)

$\bar{x}_3 \vee \bar{x}$ (f1)

$\bar{x}_3 \vee \bar{x}_2$ (f2)

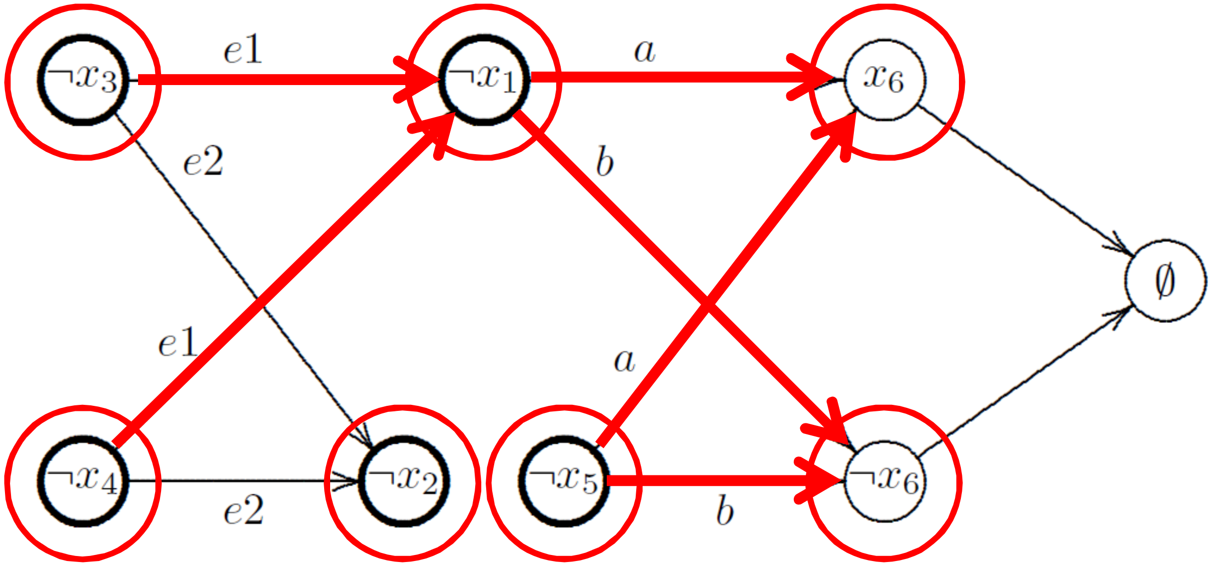
$\bar{x}_4 \vee \bar{x}_1$ (f3)

$\bar{x}_4 \vee \bar{x}_2$ (f4)

No edges for clause (d)

Implication Graph

- Hiring example: Build conflict graph at first leaf node.

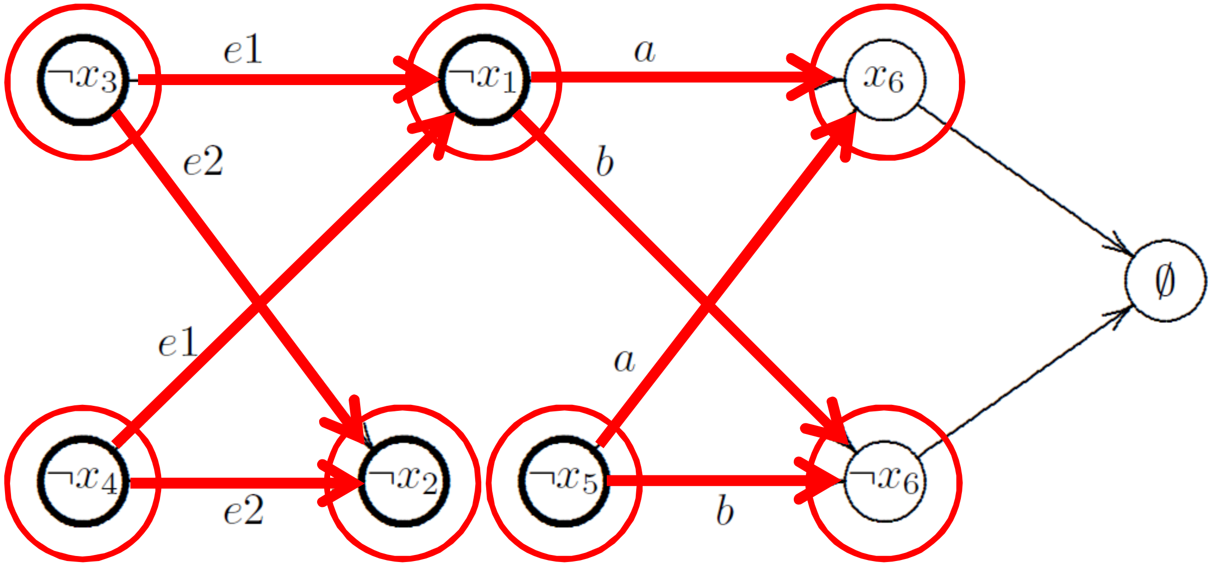


Add edges for clause (e1), which is $(\bar{x}_3 \wedge \bar{x}_4) \rightarrow \bar{x}_1$

$x_1 \vee x_5 \vee x_6$	(a)
$x_1 \vee x_5 \vee \bar{x}_6$	(b)
$x_2 \vee \bar{x}_5 \vee x_6$	(c)
$x_2 \vee \bar{x}_5 \vee \bar{x}_6$	(d)
$\bar{x}_1 \vee x_3 \vee x_4$	$(e1)$
$\bar{x}_2 \vee x_3 \vee x_4$	$(e2)$
$\bar{x}_3 \vee \bar{x}$	$(f1)$
$\bar{x}_3 \vee \bar{x}_2$	$(f2)$
$\bar{x}_4 \vee \bar{x}_1$	$(f3)$
$\bar{x}_4 \vee \bar{x}_2$	$(f4)$

Implication Graph

- Hiring example: Build conflict graph at first leaf node.



$x_1 \vee x_5 \vee x_6$ (*a*)

$x_1 \vee x_5 \vee \bar{x}_6$ (*b*)

$x_2 \vee \bar{x}_5 \vee x_6$ (*c*)

$x_2 \vee \bar{x}_5 \vee \bar{x}_6$ (*d*)

$\bar{x}_1 \vee x_3 \vee x_4$ (*e*1)

$\bar{x}_2 \vee x_3 \vee x_4$ (*e*2)

$\bar{x}_3 \vee \bar{x}$ (*f*1)

$\bar{x}_3 \vee \bar{x}_2$ (*f*2)

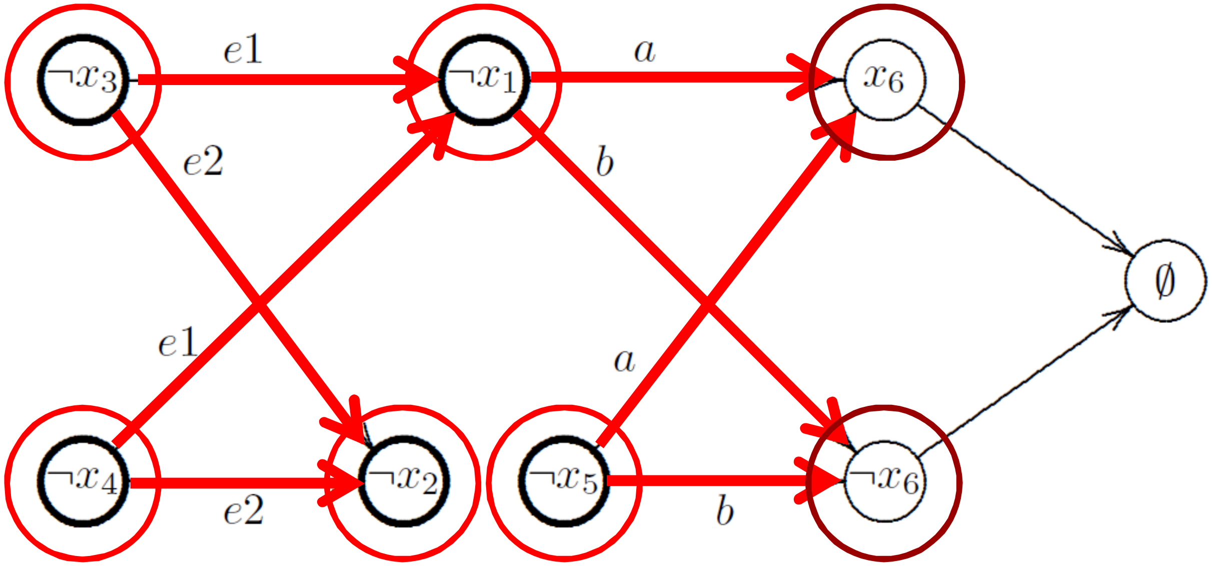
$\bar{x}_4 \vee \bar{x}_1$ (*f*3)

$\bar{x}_4 \vee \bar{x}_2$ (*f*4)

Add edges for clause (e2), which is $(\bar{x}_3 \wedge \bar{x}_4) \rightarrow \bar{x}_2$

Implication Graph

- Hiring example: Build conflict graph at first leaf node.

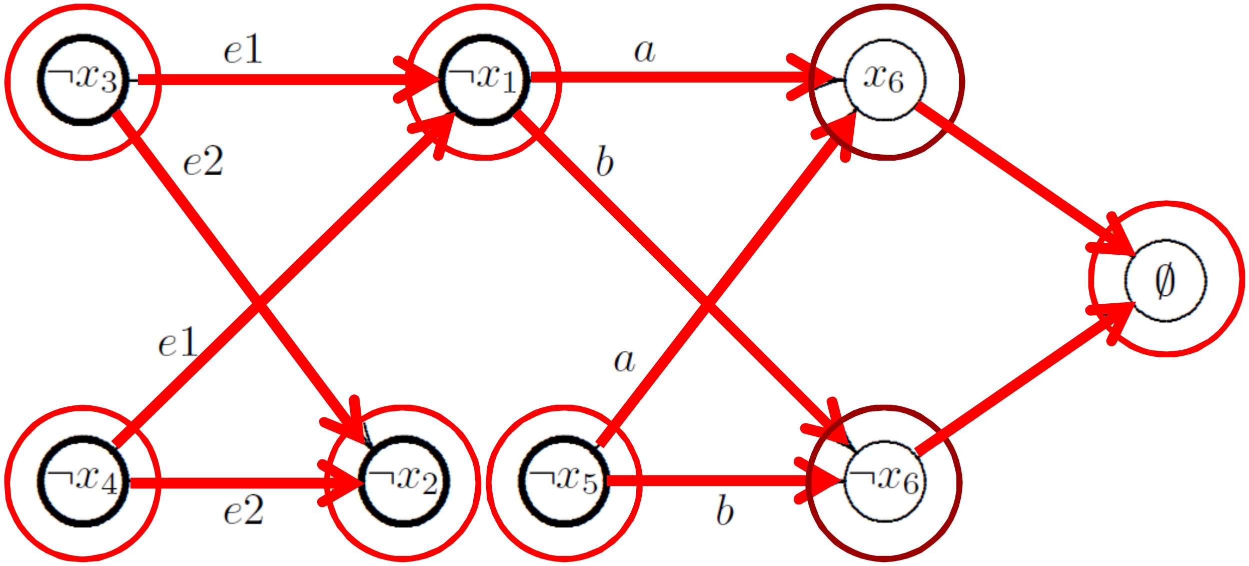


Identify **conflict literals**, i.e., both x_i and $\bar{x}_i$ are present.

$x_1 \vee x_5 \vee x_6$	(a)
$x_1 \vee x_5 \vee \bar{x}_6$	(b)
$x_2 \vee \bar{x}_5 \vee x_6$	(c)
$x_2 \vee \bar{x}_5 \vee \bar{x}_6$	(d)
$\bar{x}_1 \vee x_3 \vee x_4$	$(e1)$
$\bar{x}_2 \vee x_3 \vee x_4$	$(e2)$
$\bar{x}_3 \vee \bar{x}$	$(f1)$
$\bar{x}_3 \vee \bar{x}_2$	$(f2)$
$\bar{x}_4 \vee \bar{x}_1$	$(f3)$
$\bar{x}_4 \vee \bar{x}_2$	$(f4)$

Implication Graph

- Hiring example: Build conflict graph at first leaf node.

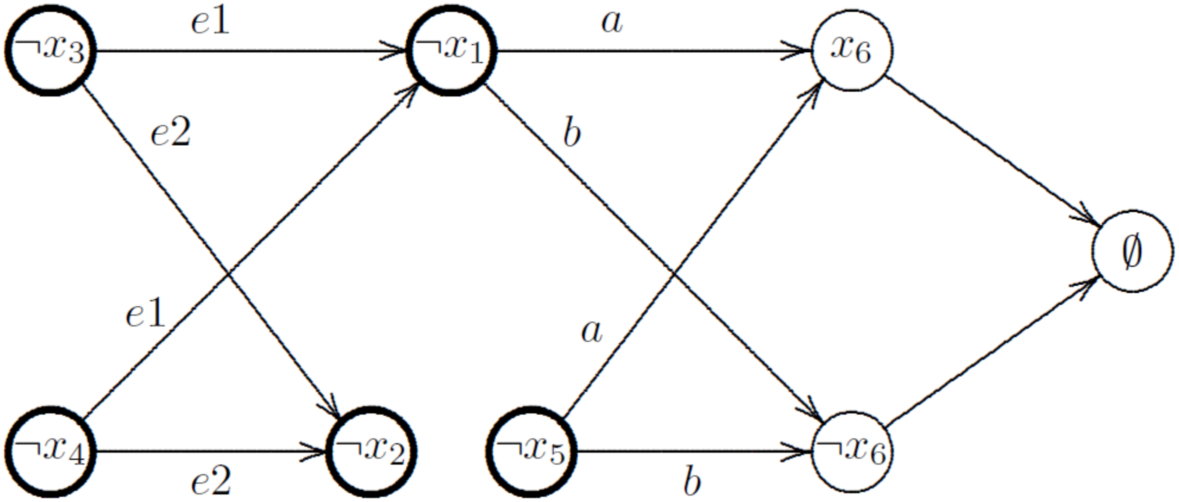


Add arcs from conflict literals to $\emptyset$.

$x_1 \vee x_5 \vee x_6$	(a)
$x_1 \vee x_5 \vee \bar{x}_6$	(b)
$x_2 \vee \bar{x}_5 \vee x_6$	(c)
$x_2 \vee \bar{x}_5 \vee \bar{x}_6$	(d)
$\bar{x}_1 \vee x_3 \vee x_4$	$(e1)$
$\bar{x}_2 \vee x_3 \vee x_4$	$(e2)$
$\bar{x}_3 \vee \bar{x}$	$(f1)$
$\bar{x}_3 \vee \bar{x}_2$	$(f2)$
$\bar{x}_4 \vee \bar{x}_1$	$(f3)$
$\bar{x}_4 \vee \bar{x}_2$	$(f4)$

Implication Graph

- A proof of infeasibility is represented by a **conflict graph** from the implication graph.
    - There may be several proofs.



$x_1 \vee x_5 \vee x_6$ (a)

$x_1 \vee x_5 \vee \bar{x}_6$ (b)

$x_2 \vee \bar{x}_5 \vee x_6$ (c)

$x_2 \vee \bar{x}_5 \vee \bar{x}_6$ (d)

$\bar{x}_1 \vee x_3 \vee x_4$ $(e1)$

$\bar{x}_2 \vee x_3 \vee x_4$ $(e2)$

$\bar{x}_3 \vee \bar{x}$ $(f1)$

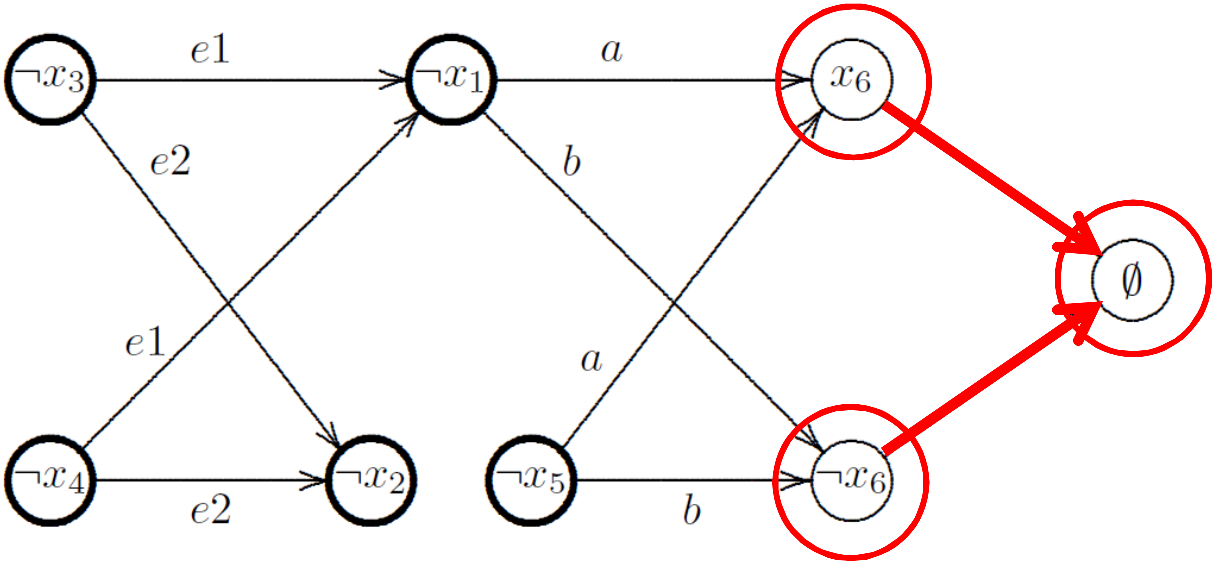
$\bar{x}_3 \vee \bar{x}_2$ $(f2)$

$\bar{x}_4 \vee \bar{x}_1$ $(f3)$

$\bar{x}_4 \vee \bar{x}_2$ $(f4)$

Implication Graph

- Build a **conflict graph** *G* from the implication graph.

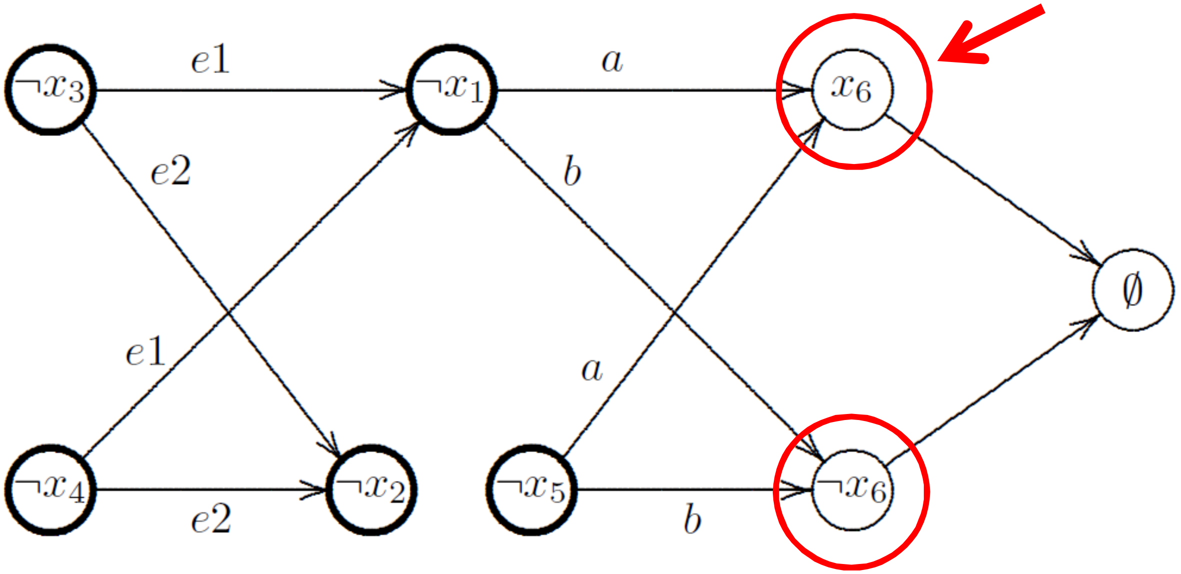


$x_1 \vee x_5 \vee x_6$	(a)
$x_1 \vee x_5 \vee \overline{x}_6$	(b)
$x_2 \vee \overline{x}_5 \vee x_6$	(c)
$x_2 \vee \overline{x}_5 \vee \overline{x}_6$	(d)
$\overline{x}_1 \vee x_3 \vee x_4$	$(e1)$
$\overline{x}_2 \vee x_3 \vee x_4$	$(e2)$
$\overline{x}_3 \vee \overline{x}$	$(f1)$
$\overline{x}_3 \vee \overline{x}_2$	$(f2)$
$\overline{x}_4 \vee \overline{x}_1$	$(f3)$
$\overline{x}_4 \vee \overline{x}_2$	$(f4)$

Create edges in *G* for any two conflict literals and ∅.

Implication Graph

- Build a **conflict graph** *G* from the implication graph.



$x_1 \vee x_5 \vee x_6$ (*a*)

$x_1 \vee x_5 \vee \bar{x}_6$ (*b*)

$x_2 \vee \bar{x}_5 \vee x_6$ (*c*)

$x_2 \vee \bar{x}_5 \vee \bar{x}_6$ (*d*)

$\bar{x}_1 \vee x_3 \vee x_4$ (*e*1)

$\bar{x}_2 \vee x_3 \vee x_4$ (*e*2)

$\bar{x}_3 \vee \bar{x}$ (*f*1)

$\bar{x}_3 \vee \bar{x}_2$ (*f*2)

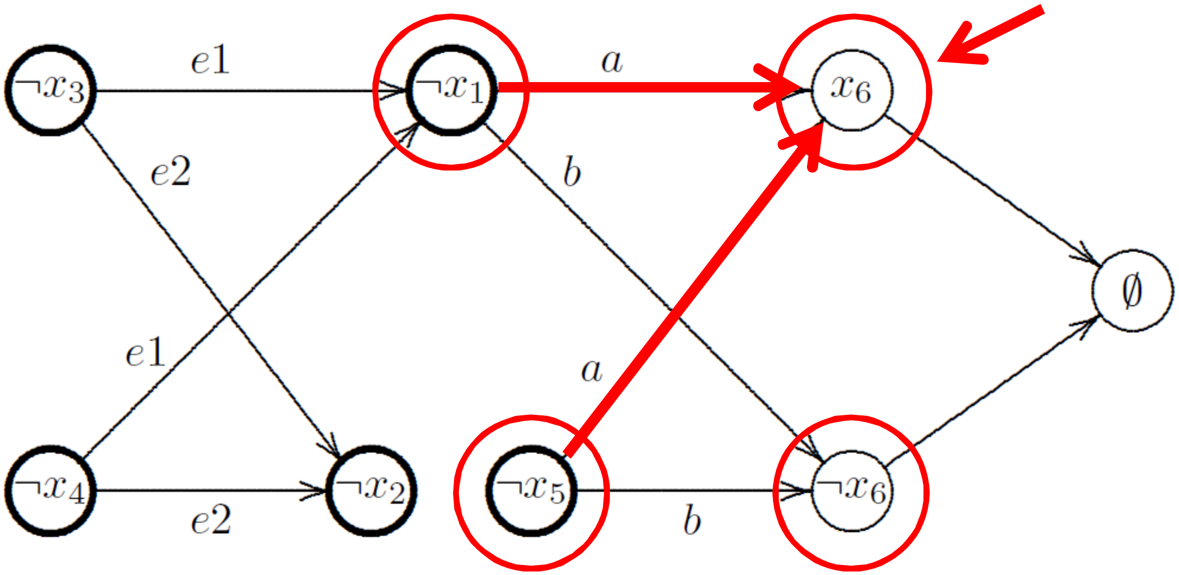
$\bar{x}_4 \vee \bar{x}_1$ (*f*3)

$\bar{x}_4 \vee \bar{x}_2$ (*f*4)

Select a non-branching vertex in *G* for which there are no incoming edges in *G*.

Implication Graph

- Build a **conflict graph** from the implication graph.

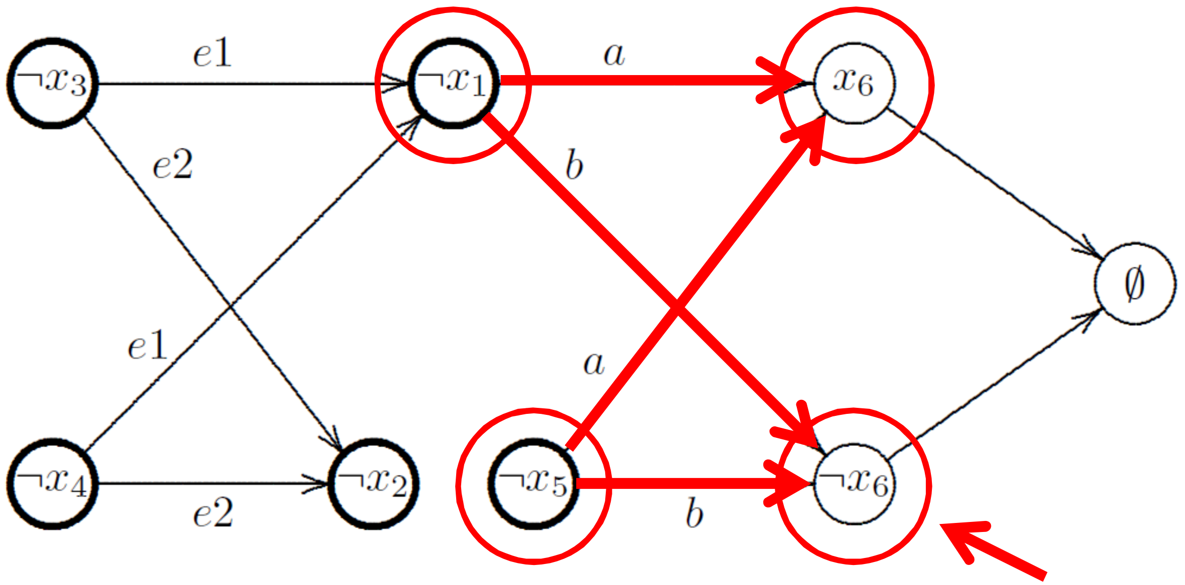


$x_1 \vee x_5 \vee x_6$	(a)
$x_1 \vee x_5 \vee \bar{x}_6$	(b)
$x_2 \vee \bar{x}_5 \vee x_6$	(c)
$x_2 \vee \bar{x}_5 \vee \bar{x}_6$	(d)
$\bar{x}_1 \vee x_3 \vee x_4$	$(e1)$
$\bar{x}_2 \vee x_3 \vee x_4$	$(e2)$
$\bar{x}_3 \vee \bar{x}$	$(f1)$
$\bar{x}_3 \vee \bar{x}_2$	$(f2)$
$\bar{x}_4 \vee \bar{x}_1$	$(f3)$
$\bar{x}_4 \vee \bar{x}_2$	$(f4)$

Select a label on some incoming edge and create in *G* all edges bearing this label.

Implication Graph

- Build a **conflict graph** from the implication graph.

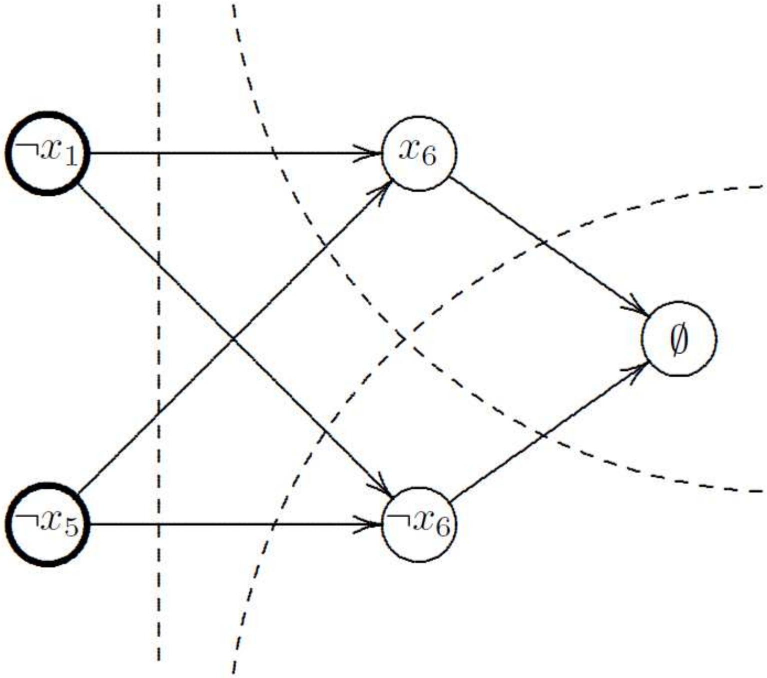


$x_1 \vee x_5 \vee x_6$	(a)
$x_1 \vee x_5 \vee \overline{x}_6$	(b)
$x_2 \vee \overline{x}_5 \vee x_6$	(c)
$x_2 \vee \overline{x}_5 \vee \overline{x}_6$	(d)
$\overline{x}_1 \vee x_3 \vee x_4$	$(e1)$
$\overline{x}_2 \vee x_3 \vee x_4$	$(e2)$
$\overline{x}_3 \vee \overline{x}$	$(f1)$
$\overline{x}_3 \vee \overline{x}_2$	$(f2)$
$\overline{x}_4 \vee \overline{x}_1$	$(f3)$
$\overline{x}_4 \vee \overline{x}_2$	$(f4)$

Repeat.

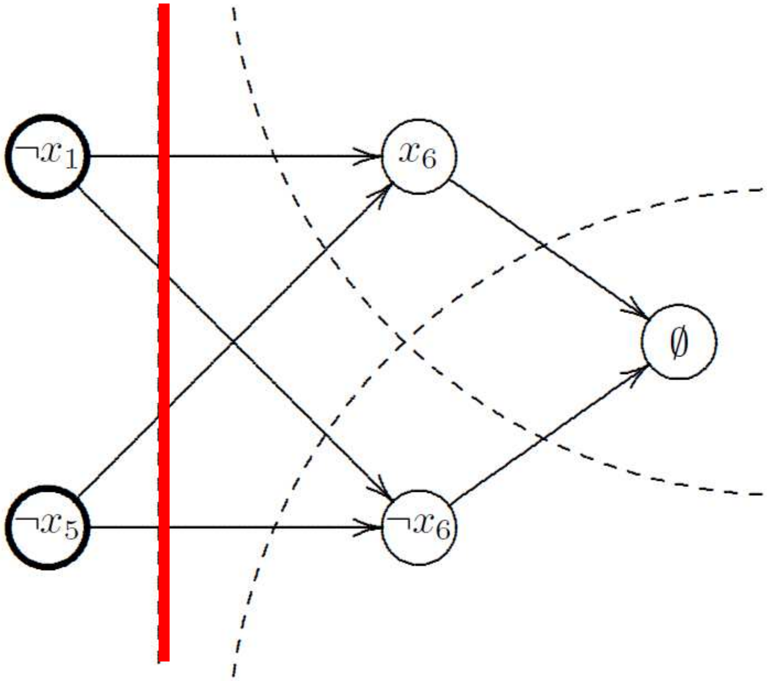
Implication Graph

- Now we have a conflict graph that represents a proof of infeasibility.



Implication Graph

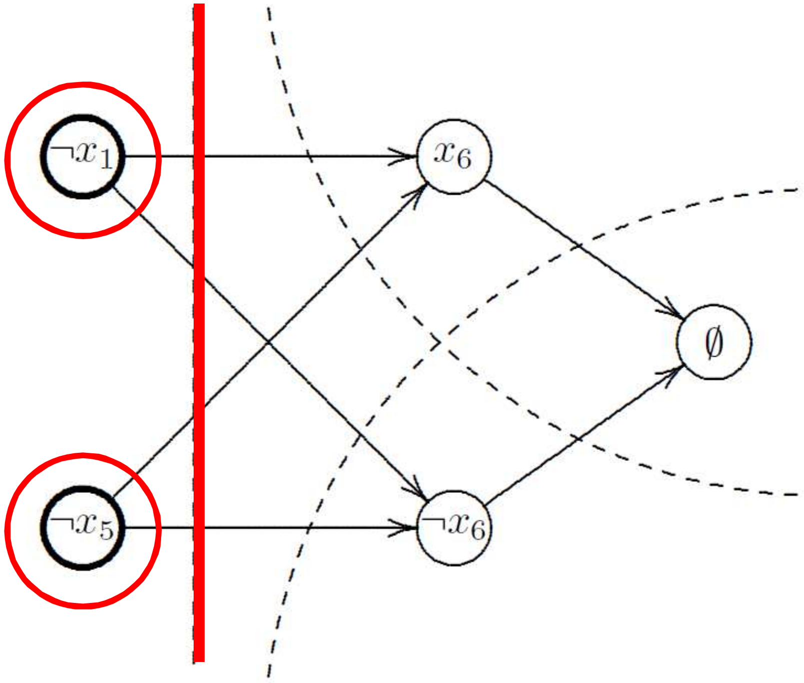
- Now we have a conflict graph that represents a proof of infeasibility.



Identify a **cut** such that:
all branching literals are on one side (the *reason side*)
and at least one conflict literal on the other side (the *conflict side*).

Implication Graph

- Now we have a conflict graph that represents a proof of infeasibility.



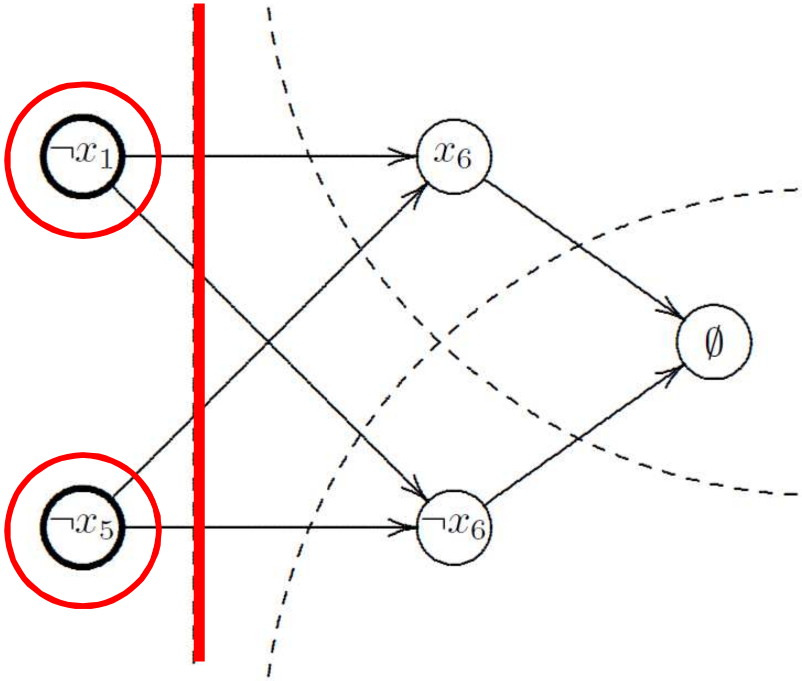
Identify **frontier** of the cut:

all vertices having at least one outgoing edge that crosses the cut

Implication Graph

- Now we have a conflict graph that represents a proof of infeasibility.

$x_1 \vee x_5$

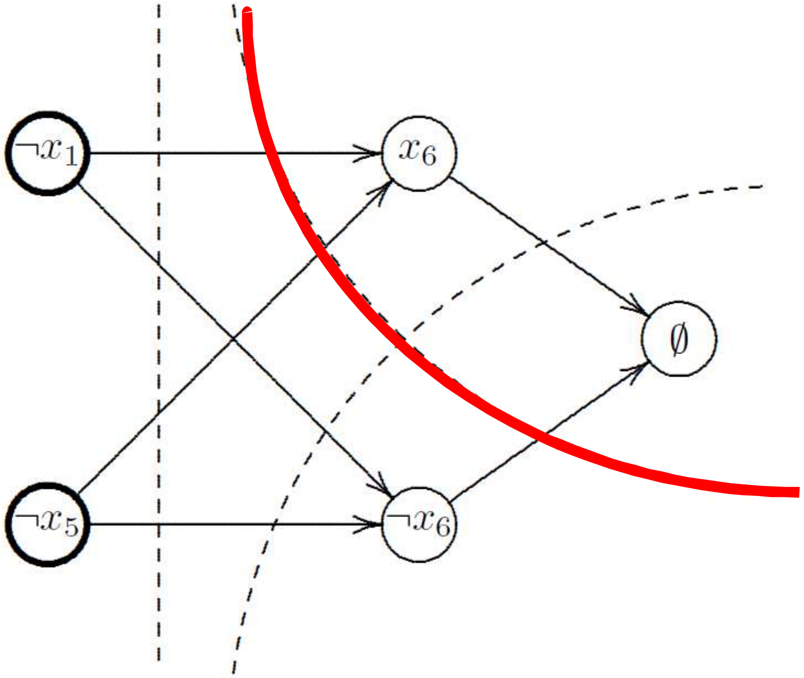


Negate these literals to obtain a conflict clause.

Implication Graph

- Now we have a conflict graph that represents a proof of infeasibility.

$x_1 \vee x_5 \vee x_6$

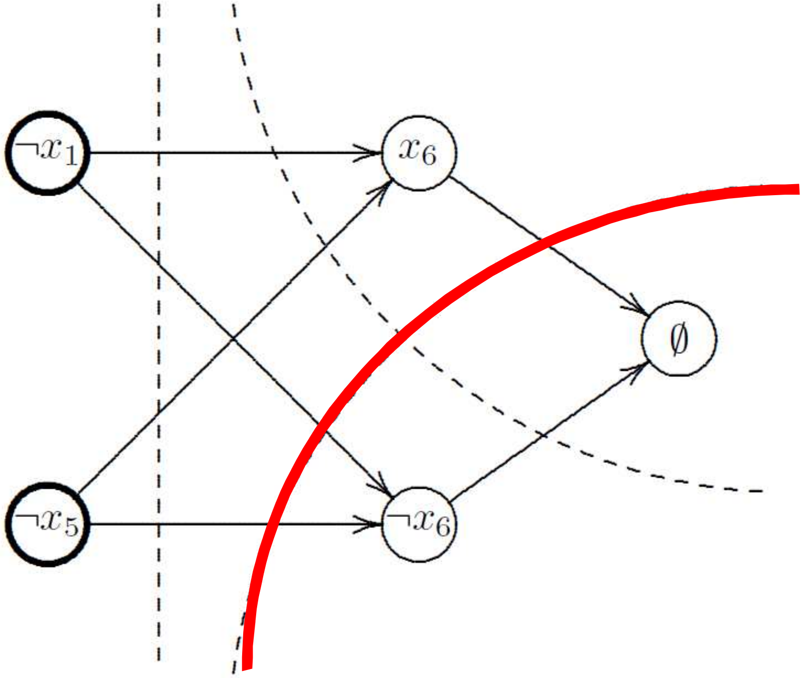


Another conflict clause (absorbed by the first).

Implication Graph

- Now we have a conflict graph that represents a proof of infeasibility.

$x_1 \vee x_5 \vee \bar{x}_6$



Another conflict clause (absorbed by the first).

Assessment of SAT Solvers

- Solvers are extremely efficient.
  - Can deal with **millions** of variables.
  - These are **complete** solvers (not heuristic methods).
    - They find a solution if one exists
    - And prove infeasibility otherwise.

Assessment of SAT Solvers

- Solvers are extremely efficient.
    - Can deal with **millions** of variables.
    - These are **complete** solvers (not heuristic methods).
        - They find a solution if one exists
        - And prove infeasibility otherwise.
- Most industrial problems are easy for their size.
    - They are nearly **renamable Horn**.
    - This teaches some important lessons.

Renamable Horn Problems

- A clause set is **Horn** if each clause contains at most one positive literal.
  - It is **renamable Horn** if it becomes Horn after complementing zero or more variables.

Renamable Horn

$$x_1 \vee x_2 \vee x_3$$
$$\bar{x}_1 \vee \bar{x}_2 \vee x_3$$

$$\bar{x}_1 \vee x_2 \vee \bar{x}_3$$
$$x_1 \vee \bar{x}_2 \vee \bar{x}_3$$

Not renamable Horn

$$x_1 \vee x_2 \vee x_3$$
$$\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3$$

Backdoors and Branching

- A renamable Horn sat problem can be solved by unit resolution.
    - Very fast.
- Industrial SAT problems tend to be nearly renamable Horn.
    - They become renamable Horn after fixing a few variables.
    - Such a variable set is known as a **backdoor**.
- This suggests a branching strategy.
    - Branch first on backdoor variables.
    - Then problems at leaf nodes are easy.

Lesson 1

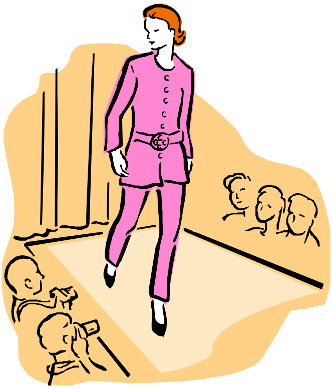
- The branching order can make a huge difference.
  - Try to identify a small backdoor.
    - This is a max clique problem, NP-hard.
    - Can use heuristics.
  - Try random restarts.
    - This may find a smaller backdoor.

Lesson 2

- NP-complete problems can be easy.
  - SAT is NP-complete.
  - But the class contains many easy problems
    - For example, almost all random instances of 3-SAT are easy.
    - Except when ratio of number of clauses to number of variables is about 4.3
    - This is known as a **phase transition**.

Lesson 2

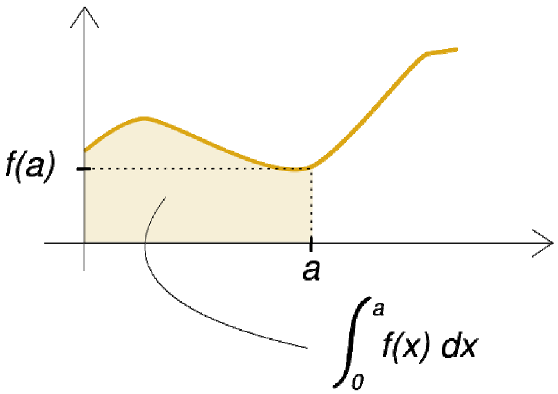
- NP-complete problems can be easy.
  - SAT is NP-complete.
  - But the class contains many easy problems
    - For example, almost all random instances of 3-SAT are easy.
    - Except when ratio of number of clauses to number of variables is about 4.3
    - This is known as a **phase transition**.
  - Think about it: The class NP is NP-complete (trivially).
    - Even though it contains all the easy problems in the world!



Advanced Modeling

Advanced modeling

See slides by Helmut Simonis.



Integrating OR and CP

Complementary strengths

Simple Example

Comparison

CP vs. Mathematical Programming

MP	CP
Numerical calculation	Logic processing
Relaxation	Inference (filtering, constraint propagation)
Atomistic modeling (linear inequalities)	High-level modeling (global constraints)
Branching	Branching
Independence of model and algorithm	Constraint-based processing

CP vs. MP

- In **mathematical programming**, equations (constraints) describe the problem but don’t tell how to solve it.

- In **constraint programming**, each constraint invokes a procedure that screens out unacceptable solutions.
    - Much as each line of a computer program invokes an operation.

Advantages of CP

- Better at sequencing and scheduling
  - …where MP methods have weak relaxations.
- Adding messy constraints makes the problem easier.
  - The more constraints, the better.
- More powerful modeling language.
  - Global constraints lead to succinct models.
  - Constraints convey problem structure to the solver.

Disdvantages of CP

- Weaker for continuous variables.
    - Due to lack of numerical techniques
- May fail when constraints contain many variables.
    - These constraints don't propagate well.
- Not robust
    - Lack of relaxation technology

Obvious solution…

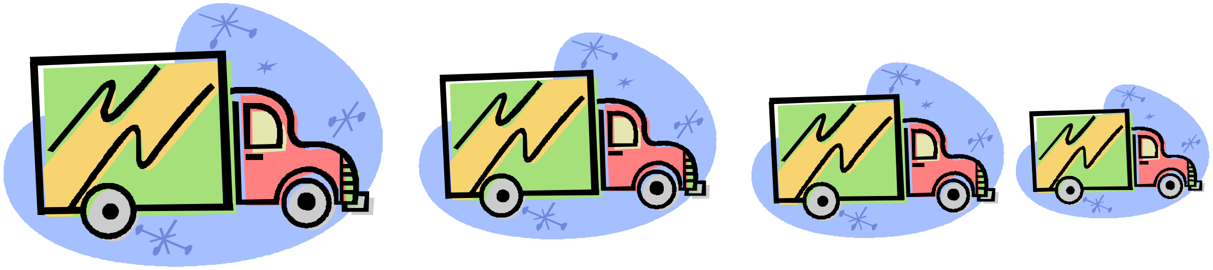
- Integrate CP and MP.

Software for Integrated Methods

- ECLiPSe
  - Exchanges information between ECLiPSEe solver, Xpress-MP
- OPL Studio
  - Combines CPLEX and ILOG CP Optimizer with script language
- Mosel
  - Combines Xpress-MP, Xpress-Kalis with low-level modeling
- BARON
  - Global optimization with relaxation + domain reduction
- SIMPL
  - Full integration with high-level modeling (prototype)
- SCIP
  - Combines MILP and CP-based propagation

Example: Freight Transfer

- Transport 42 tons of freight using 8 trucks, which come in 4 sizes…



Truck size	Number available	Capacity (tons)	Cost per truck
1	3	7	90
2	3	5	60
3	3	4	50
4	3	3	40

Number of trucks of type 1

$$\begin{aligned}
&\min\ 90x_1 + 60x_2 + 50x_3 + 40x_4 \\
&7x_1 + 5x_2 + 4x_3 + 3x_4 \geq 42 \\
&x_1 + x_2 + x_3 + x_4 \leq 8 \\
&x_i \in \{0,1,2,3\}
\end{aligned}$$

Knapsack covering constraint

Knapsack packing constraint

Truck type	Number available	Capacity (tons)	Cost per truck
1	3	7	90
2	3	5	60
3	3	4	50
4	3	3	40

Bounds propagation

$$\min\ 90x_1 + 60x_2 + 50x_3 + 40x_4$$

$$7x_1 + 5x_2 + 4x_3 + 3x_4 \geq 42$$

$$x_1 + x_2 + x_3 + x_4 \leq 8$$

$$x_i \in \{0,1,2,3\}$$

$$x_1 \geq \left\lceil \frac{42 - 5 \cdot 3 - 4 \cdot 3 - 3 \cdot 3}{7} \right\rceil = 1$$

Bounds propagation

$$\min\ 90x_1 + 60x_2 + 50x_3 + 40x_4$$

$$7x_1 + 5x_2 + 4x_3 + 3x_4 \geq 42$$

$$x_1 + x_2 + x_3 + x_4 \leq 8$$

$$x_1 \in \{1,2,3\},\quad x_2, x_3, x_4 \in \{0,1,2,3\}$$

Reduced domain

$$x_1 \geq \left\lceil \frac{42 - 5\cdot 3 - 4\cdot 3 - 3\cdot 3}{7} \right\rceil = 1$$

Cutting Planes

Begin with continuous relaxation

$$\min\ 90x_1 + 60x_2 + 50x_3 + 40x_4$$
$$7x_1 + 5x_2 + 4x_3 + 3x_4 \geq 42$$
$$x_1 + x_2 + x_3 + x_4 \leq 8$$
$$0 \leq x_i \leq 3, \quad x_1 \geq 1$$

Replace domains with bounds

This is a linear programming problem, which is easy to solve.

Its optimal value provides a lower bound on optimal value of original problem.

Cutting planes (valid inequalities)

$$\min\ 90x_1 + 60x_2 + 50x_3 + 40x_4$$
$$7x_1 + 5x_2 + 4x_3 + 3x_4 \geq 42$$
$$x_1 + x_2 + x_3 + x_4 \leq 8$$
$$0 \leq x_i \leq 3, \quad x_1 \geq 1$$

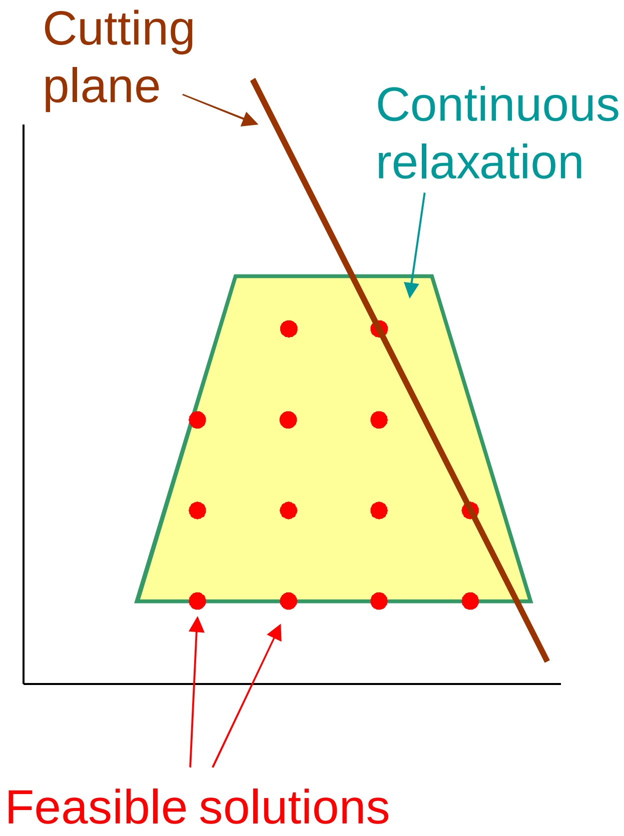
We can create a **tighter** relaxation (larger minimum value) with the addition of **cutting planes**.

Cutting planes (valid inequalities)

$$\min\ 90x_1 + 60x_2 + 50x_3 + 40x_4$$
$$7x_1 + 5x_2 + 4x_3 + 3x_4 \geq 42$$
$$x_1 + x_2 + x_3 + x_4 \leq 8$$
$$0 \leq x_i \leq 3, \quad x_1 \geq 1$$

All feasible solutions of the original problem satisfy a cutting plane (i.e., it is **valid**).

But a cutting plane may exclude (“**cut off**”) solutions of the continuous relaxation.



Cutting planes (valid inequalities)

$$\min\ 90x_1 + 60x_2 + 50x_3 + 40x_4$$

$$7x_1 + 5x_2 + 4x_3 + 3x_4 \geq 42$$

$$x_1 + x_2 + x_3 + x_4 \leq 8$$

$$0 \leq x_i \leq 3, \quad x_1 \geq 1$$

{1,2} is a **packing**

…because $7x_1 + 5x_2$ alone cannot satisfy the inequality, even with $x_1 = x_2 = 3$.

Cutting planes (valid inequalities)

$$\min\ 90x_1 + 60x_2 + 50x_3 + 40x_4$$
$$7x_1 + 5x_2 + 4x_3 + 3x_4 \geq 42$$
$$x_1 + x_2 + x_3 + x_4 \leq 8$$
$$0 \leq x_i \leq 3, \quad x_1 \geq 1$$

{1,2} is a **packing**

So, $4x_3 + 3x_4 \geq 42 - (7\cdot 3 + 5\cdot 3)$

Knapsack cut

which implies

$$x_3 + x_4 \geq \left\lceil \frac{42 - (7\cdot 3 + 5\cdot 3)}{\max\{4,3\}} \right\rceil = 2$$

Cutting planes (valid inequalities)

Let x_i have domain $[L_i, U_i]$ and let $a \geq 0$.

In general, a **packing** P for $ax \geq a_0$ satisfies

$$\sum_{i \notin P} a_i x_i \geq a_0 - \sum_{i \in P} a_i U_i$$

and generates a **knapsack cut**

$$\sum_{i \notin P} x_i \geq \left\lceil \frac{a_0 - \sum_{i \in P} a_i U_i}{\max_{i \notin P} \{a_i\}} \right\rceil$$

Cutting planes (valid inequalities)

$$\min\ 90x_1 + 60x_2 + 50x_3 + 40x_4$$

$$7x_1 + 5x_2 + 4x_3 + 3x_4 \geq 42$$

$$x_1 + x_2 + x_3 + x_4 \leq 8$$

$$0 \leq x_i \leq 3, \quad x_1 \geq 1$$

Maximal Packings	**Knapsack cuts**
{1,2}	$x_3 + x_4 \geq 2$
{1,3}	$x_2 + x_4 \geq 2$
{1,4}	$x_2 + x_3 \geq 3$

Knapsack cuts corresponding to nonmaximal packings can be nonredundant.

Continuous relaxation with cuts

$$\min\ 90x_1 + 60x_2 + 50x_3 + 40x_4$$

$$7x_1 + 5x_2 + 4x_3 + 3x_4 \geq 42$$

$$x_1 + x_2 + x_3 + x_4 \leq 8$$

$$0 \leq x_i \leq 3, \quad x_1 \geq 1$$

$$\boxed{\begin{array}{l} x_3 + x_4 \geq 2 \\ x_2 + x_4 \geq 2 \\ x_2 + x_3 \geq 3 \end{array}}$$

Knapsack cuts

Optimal value of 523.3 is a lower bound on optimal value of original problem.

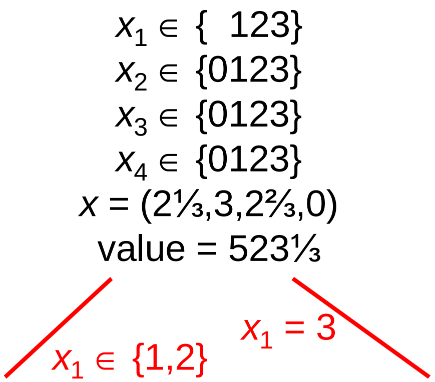
Branch-infer-and-relax tree

Propagate bounds and solve relaxation of original problem.

$x_1 \in \{\ \ 123\}$
$x_2 \in \{0123\}$
$x_3 \in \{0123\}$
$x_4 \in \{0123\}$
$x = (2⅓,3,2⅔,0)$
value = 523⅓

Branch-infer-and-relax tree

Branch on a variable with nonintegral value in the relaxation.



$x_1 \in \{\ \ 123\}$

$x_2 \in \{0123\}$

$x_3 \in \{0123\}$

$x_4 \in \{0123\}$

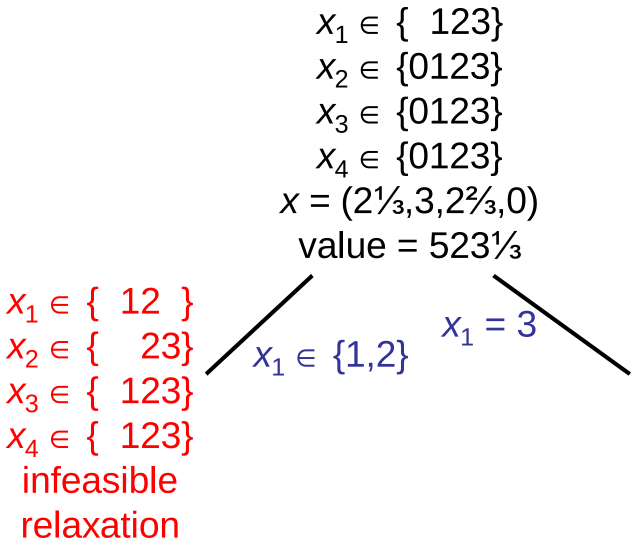
$x = (2\frac{1}{3},3,2\frac{2}{3},0)$

value = $523\frac{1}{3}$

$x_1 \in \{1,2\}$

$x_1 = 3$

Branch-infer-and-relax tree



$x_1 \in \{\ \ 123\}$
$x_2 \in \{0123\}$
$x_3 \in \{0123\}$
$x_4 \in \{0123\}$
$x = (2⅓,3,2⅔,0)$
value = 523⅓

$x_1 \in \{1,2\}$

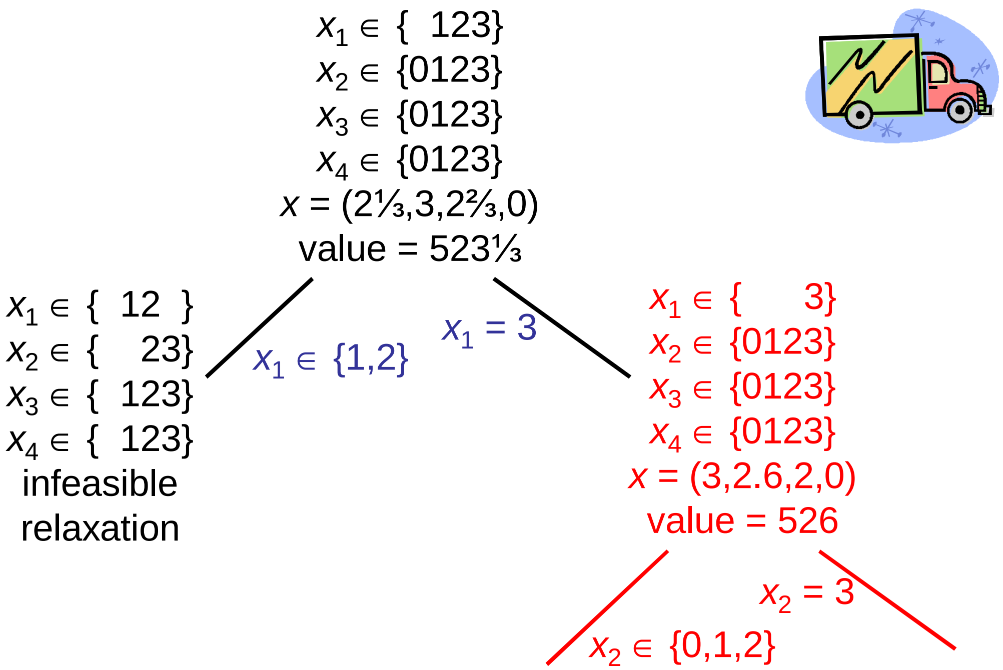
$x_1 = 3$

$x_1 \in \{\ \ 12\ \ \}$
$x_2 \in \{\ \ \ \ 23\}$
$x_3 \in \{\ \ 123\}$
$x_4 \in \{\ \ 123\}$
infeasible
relaxation

Propagate bounds and solve relaxation.

Since relaxation is infeasible, backtrack.

Branch-infer-and-relax tree



$x_1 \in \{\ \ 123\}$
$x_2 \in \{0123\}$
$x_3 \in \{0123\}$
$x_4 \in \{0123\}$
$x = (2⅓,3,2⅔,0)$
value = 523⅓

$x_1 \in \{1,2\}$

$x_1 = 3$

$x_1 \in \{\ 12\ \}$
$x_2 \in \{\ \ \ 23\}$
$x_3 \in \{\ 123\}$
$x_4 \in \{\ 123\}$
infeasible
relaxation

$x_1 \in \{\ \ \ \ \ 3\}$
$x_2 \in \{0123\}$
$x_3 \in \{0123\}$
$x_4 \in \{0123\}$
$x = (3,2.6,2,0)$
value = 526

$x_2 \in \{0,1,2\}$

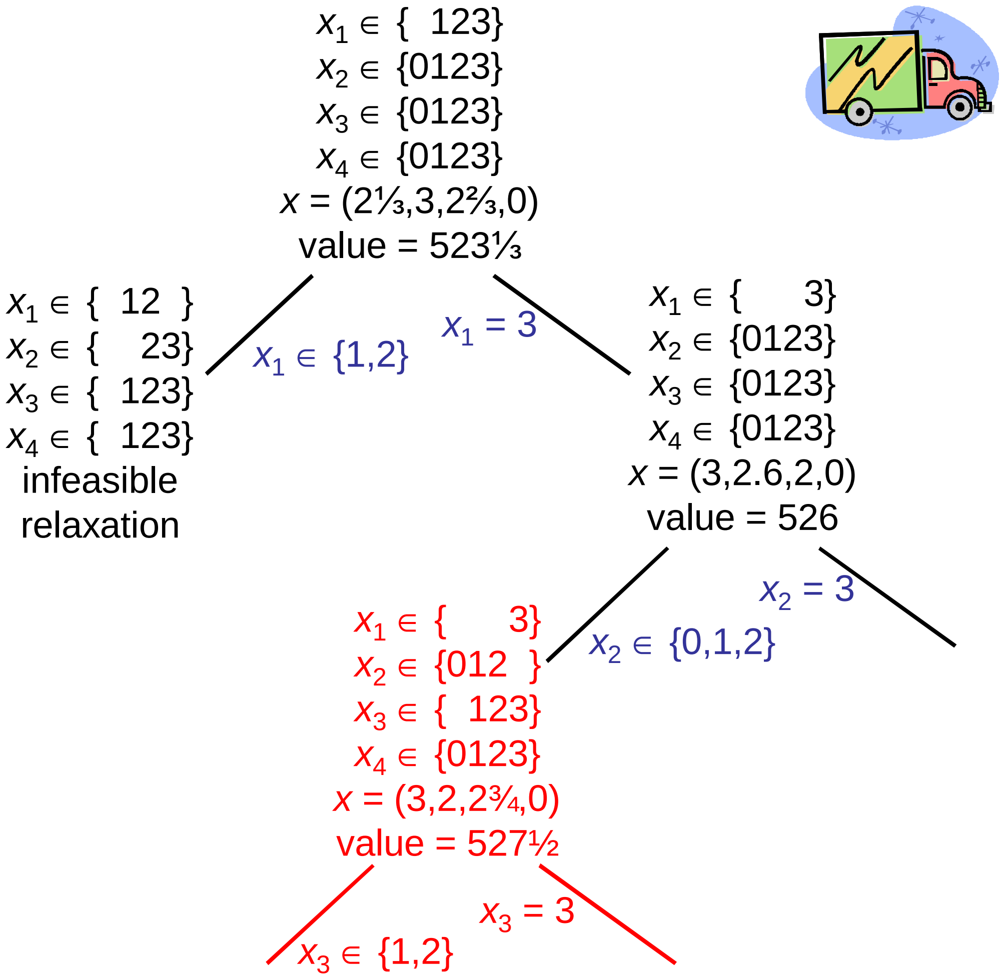
$x_2 = 3$

Propagate bounds and solve relaxation.

Branch on nonintegral variable.

Branch-infer-and-relax tree

Branch again.



$x_1 \in \{\ \ 123\}$
$x_2 \in \{0123\}$
$x_3 \in \{0123\}$
$x_4 \in \{0123\}$
x = (2⅓,3,2⅔,0)
value = 523⅓

$x_1 \in \{1,2\}$

$x_1 \in \{\ \ 12\ \ \}$
$x_2 \in \{\ \ \ \ 23\}$
$x_3 \in \{\ \ 123\}$
$x_4 \in \{\ \ 123\}$
infeasible
relaxation

$x_1 = 3$

$x_1 \in \{\ \ \ \ \ \ 3\}$
$x_2 \in \{0123\}$
$x_3 \in \{0123\}$
$x_4 \in \{0123\}$
x = (3,2.6,2,0)
value = 526

$x_2 \in \{0,1,2\}$

$x_1 \in \{\ \ \ \ \ \ 3\}$
$x_2 \in \{012\ \ \}$
$x_3 \in \{\ \ 123\}$
$x_4 \in \{0123\}$
x = (3,2,2¾,0)
value = 527½

$x_3 \in \{1,2\}$

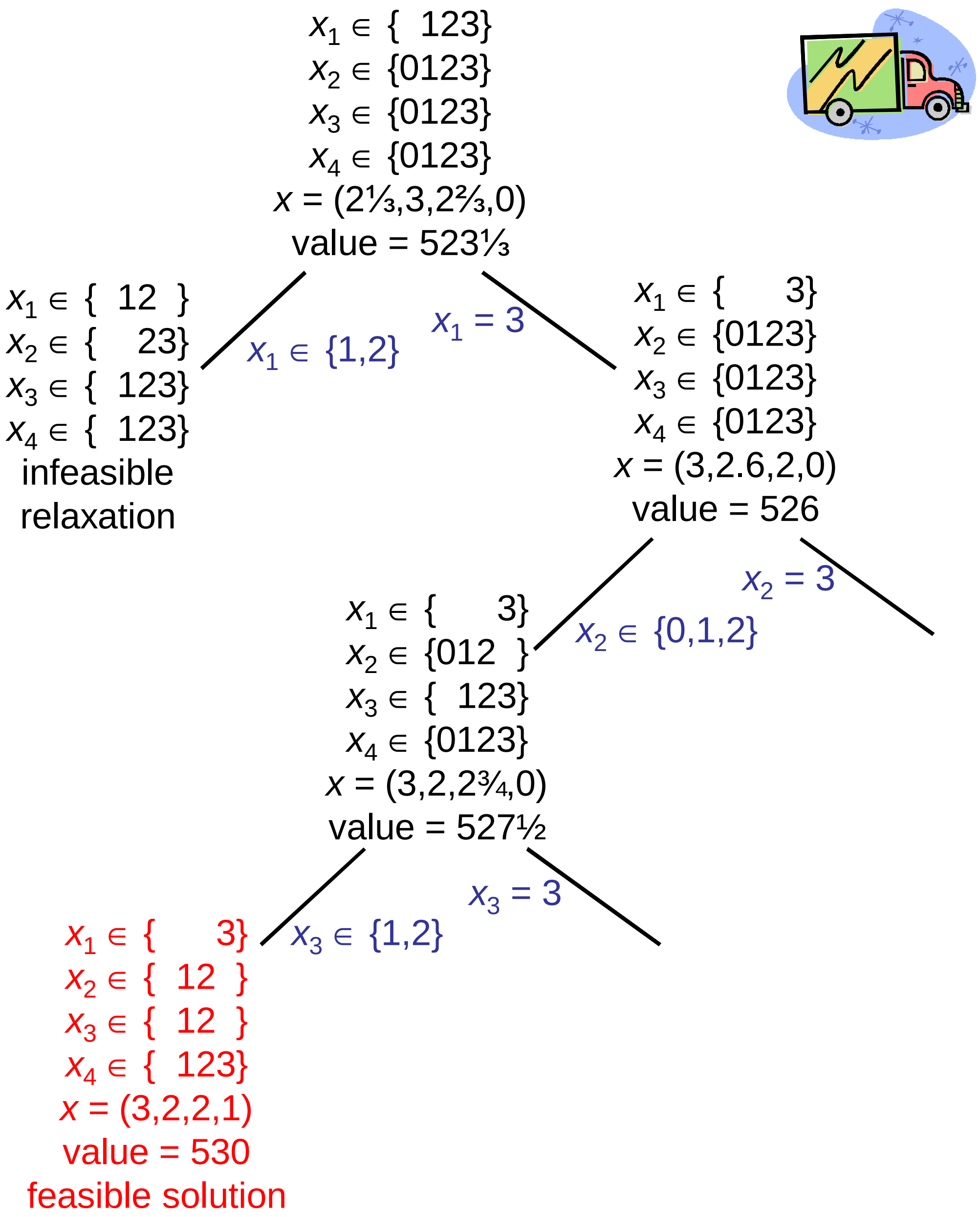
$x_3 = 3$

$x_2 = 3$

Branch-infer-and-relax tree

Solution of relaxation is integral and therefore feasible in the original problem.

This becomes the **incumbent solution.**



$x_1 \in \{\ \ 123\}$
$x_2 \in \{0123\}$
$x_3 \in \{0123\}$
$x_4 \in \{0123\}$
$x = (2\frac{1}{3},3,2\frac{2}{3},0)$
value = $523\frac{1}{3}$

$x_1 \in \{1,2\}$

$x_1 \in \{\ \ 12\ \ \}$
$x_2 \in \{\ \ \ \ 23\}$
$x_3 \in \{\ \ 123\}$
$x_4 \in \{\ \ 123\}$
infeasible relaxation

$x_1 = 3$

$x_1 \in \{\ \ \ \ \ \ 3\}$
$x_2 \in \{0123\}$
$x_3 \in \{0123\}$
$x_4 \in \{0123\}$
$x = (3,2.6,2,0)$
value = 526

$x_2 \in \{0,1,2\}$

$x_1 \in \{\ \ \ \ \ \ 3\}$
$x_2 \in \{012\ \ \}$
$x_3 \in \{\ \ 123\}$
$x_4 \in \{0123\}$
$x = (3,2,2\frac{3}{4},0)$
value = $527\frac{1}{2}$

$x_2 = 3$

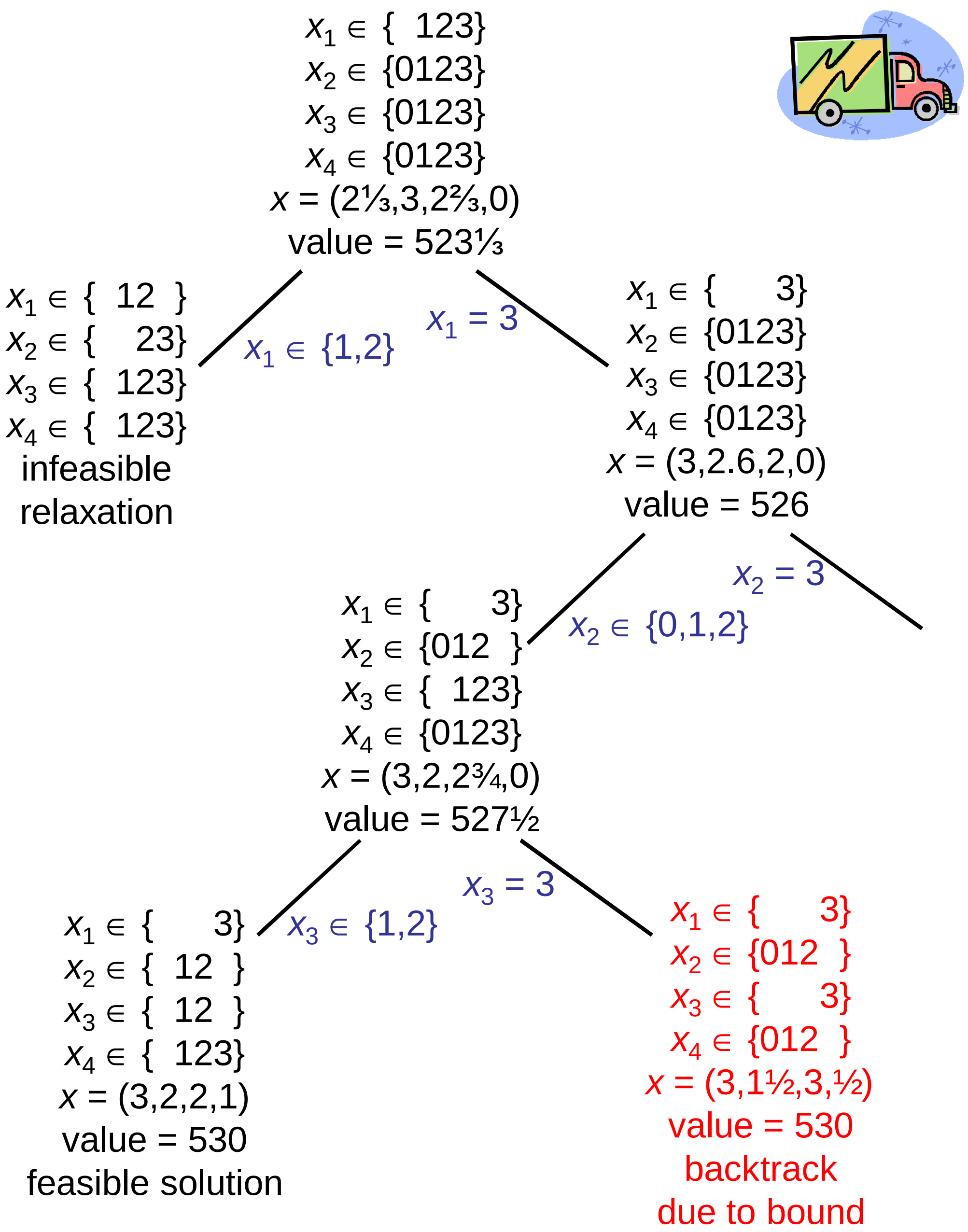
$x_3 \in \{1,2\}$

$x_1 \in \{\ \ \ \ \ \ 3\}$
$x_2 \in \{\ \ 12\ \ \}$
$x_3 \in \{\ \ 12\ \ \}$
$x_4 \in \{\ \ 123\}$
$x = (3,2,2,1)$
value = 530
feasible solution

$x_3 = 3$

Branch-infer-and-relax tree

Solution is nonintegral, but we can backtrack because value of relaxation is no better than incumbent solution.



$x_1 \in \{\ \ 123\}$
$x_2 \in \{0123\}$
$x_3 \in \{0123\}$
$x_4 \in \{0123\}$
$x = (2⅓,3,2⅔,0)$
value = 523⅓

$x_1 \in \{1,2\}$

$x_1 \in \{\ 12\ \ \}$
$x_2 \in \{\ \ \ 23\}$
$x_3 \in \{\ 123\}$
$x_4 \in \{\ 123\}$
infeasible
relaxation

$x_1 = 3$

$x_1 \in \{\ \ \ \ \ 3\}$
$x_2 \in \{0123\}$
$x_3 \in \{0123\}$
$x_4 \in \{0123\}$
$x = (3,2.6,2,0)$
value = 526

$x_2 \in \{0,1,2\}$

$x_1 \in \{\ \ \ \ \ 3\}$
$x_2 \in \{012\ \ \}$
$x_3 \in \{\ 123\}$
$x_4 \in \{0123\}$
$x = (3,2,2¾,0)$
value = 527½

$x_2 = 3$

$x_3 \in \{1,2\}$

$x_1 \in \{\ \ \ \ \ 3\}$
$x_2 \in \{\ 12\ \ \}$
$x_3 \in \{\ 12\ \ \}$
$x_4 \in \{\ 123\}$
$x = (3,2,2,1)$
value = 530
feasible solution

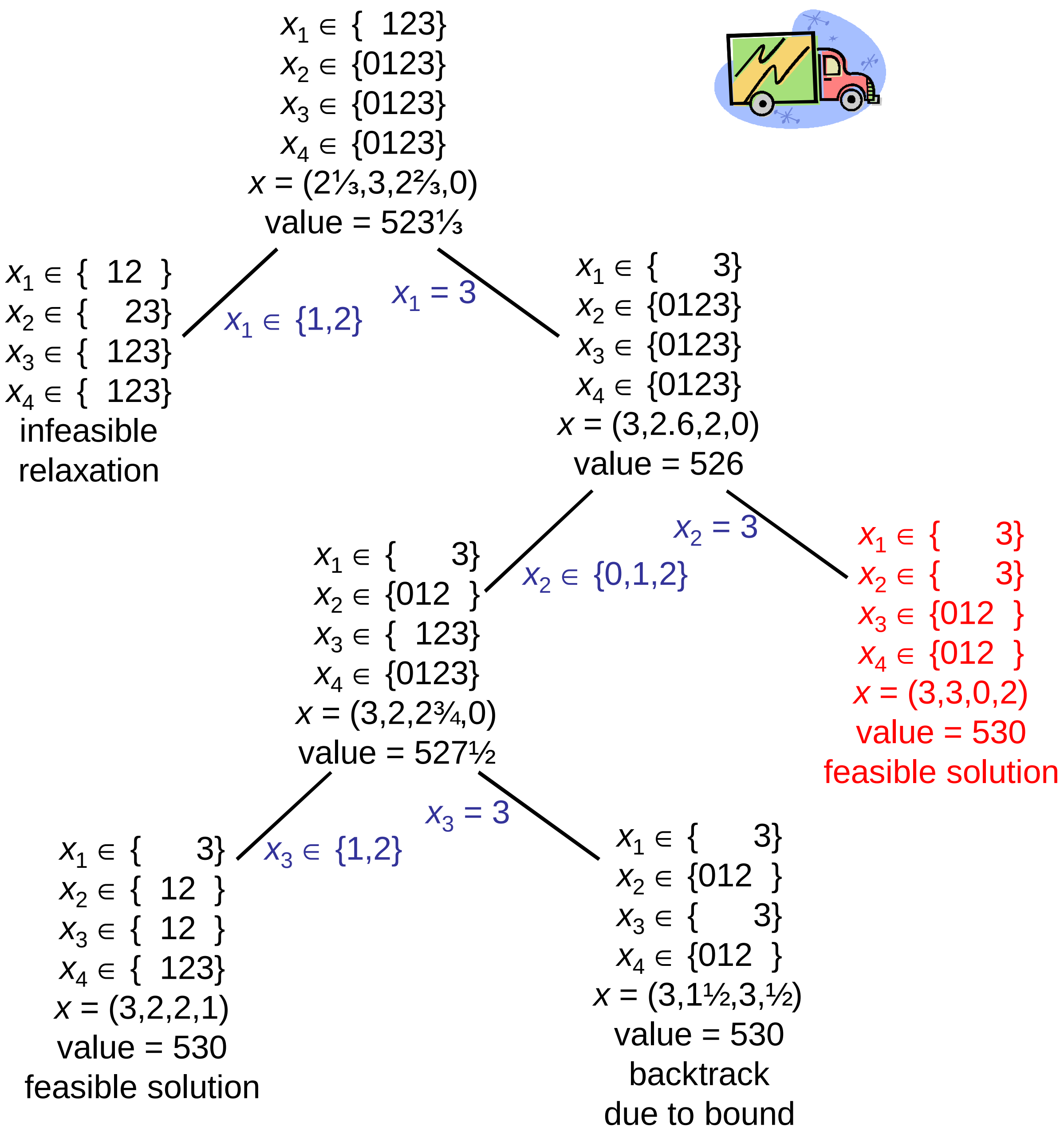
$x_3 = 3$

$x_1 \in \{\ \ \ \ \ 3\}$
$x_2 \in \{012\ \ \}$
$x_3 \in \{\ \ \ \ \ 3\}$
$x_4 \in \{012\ \ \}$
$x = (3,1½,3,½)$
value = 530
backtrack
due to bound

Branch-infer-and-relax tree

Another feasible solution found.

No better than incumbent solution, which is optimal because search has finished.



$x_1 \in \{\ \ 123\}$
$x_2 \in \{0123\}$
$x_3 \in \{0123\}$
$x_4 \in \{0123\}$
$x = (2⅓,3,2⅔,0)$
value = 523⅓

$x_1 \in \{1,2\}$

$x_1 \in \{\ \ 12\ \ \}$
$x_2 \in \{\ \ \ 23\}$
$x_3 \in \{\ \ 123\}$
$x_4 \in \{\ \ 123\}$
infeasible relaxation

$x_1 = 3$

$x_1 \in \{\ \ \ \ \ 3\}$
$x_2 \in \{0123\}$
$x_3 \in \{0123\}$
$x_4 \in \{0123\}$
$x = (3,2.6,2,0)$
value = 526

$x_2 \in \{0,1,2\}$

$x_1 \in \{\ \ \ \ \ 3\}$
$x_2 \in \{012\ \ \}$
$x_3 \in \{\ \ 123\}$
$x_4 \in \{0123\}$
$x = (3,2,2¾,0)$
value = 527½

$x_3 \in \{1,2\}$

$x_1 \in \{\ \ \ \ \ 3\}$
$x_2 \in \{\ \ 12\ \ \}$
$x_3 \in \{\ \ 12\ \ \}$
$x_4 \in \{\ \ 123\}$
$x = (3,2,2,1)$
value = 530
feasible solution

$x_3 = 3$

$x_1 \in \{\ \ \ \ \ 3\}$
$x_2 \in \{012\ \ \}$
$x_3 \in \{\ \ \ \ \ 3\}$
$x_4 \in \{012\ \ \}$
$x = (3,1½,3,½)$
value = 530
backtrack
due to bound

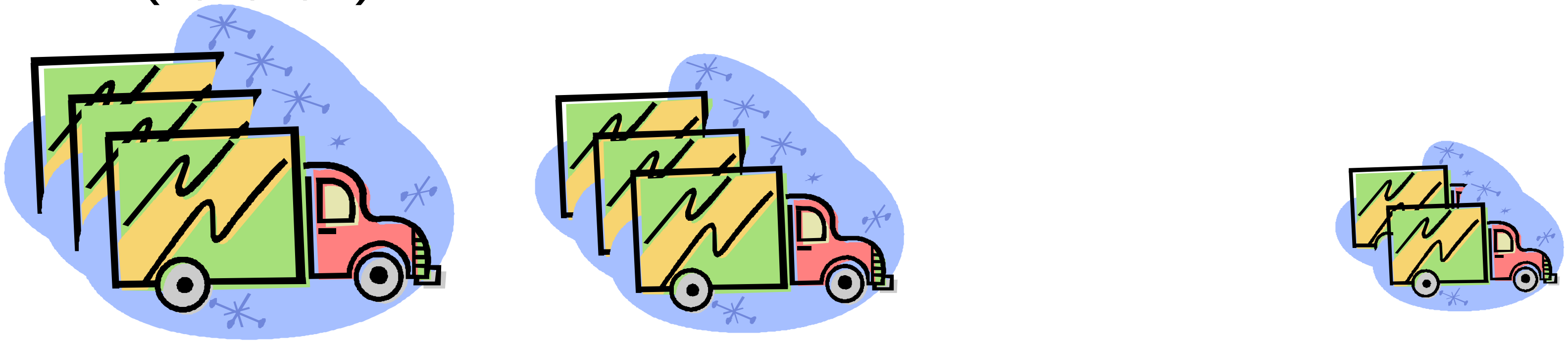
$x_2 = 3$

$x_1 \in \{\ \ \ \ \ 3\}$
$x_2 \in \{\ \ \ \ \ 3\}$
$x_3 \in \{012\ \ \}$
$x_4 \in \{012\ \ \}$
$x = (3,3,0,2)$
value = 530
feasible solution

Two optimal solutions…

$x = (3, 2, 2, 1)$

$x = (3, 3, 0, 2)$



Linear Relaxation

Why Relax?

Algebraic Analysis of LP

Linear Programming Duality

LP-Based Domain Filtering

Example: Single-Vehicle Routing

Why Relax?

Solving a relaxation of a problem can:

- Tighten variable bounds.
- Possibly solve original problem.
- Guide the search in a promising direction.
- Filter domains using reduced costs or Lagrange multipliers.
- Prune the search tree using a bound on the optimal value.
- Provide a more global view, because a single OR relaxation can pool relaxations of several constraints.

Some OR models that can provide relaxations:

- Linear programming (LP).
- Mixed integer linear programming (MILP)
  - Can itself be relaxed as an LP.
  - LP relaxation can be strengthened with cutting planes.
- Lagrangean relaxation.
- Specialized relaxations.
  - For particular problem classes.
  - For global constraints.

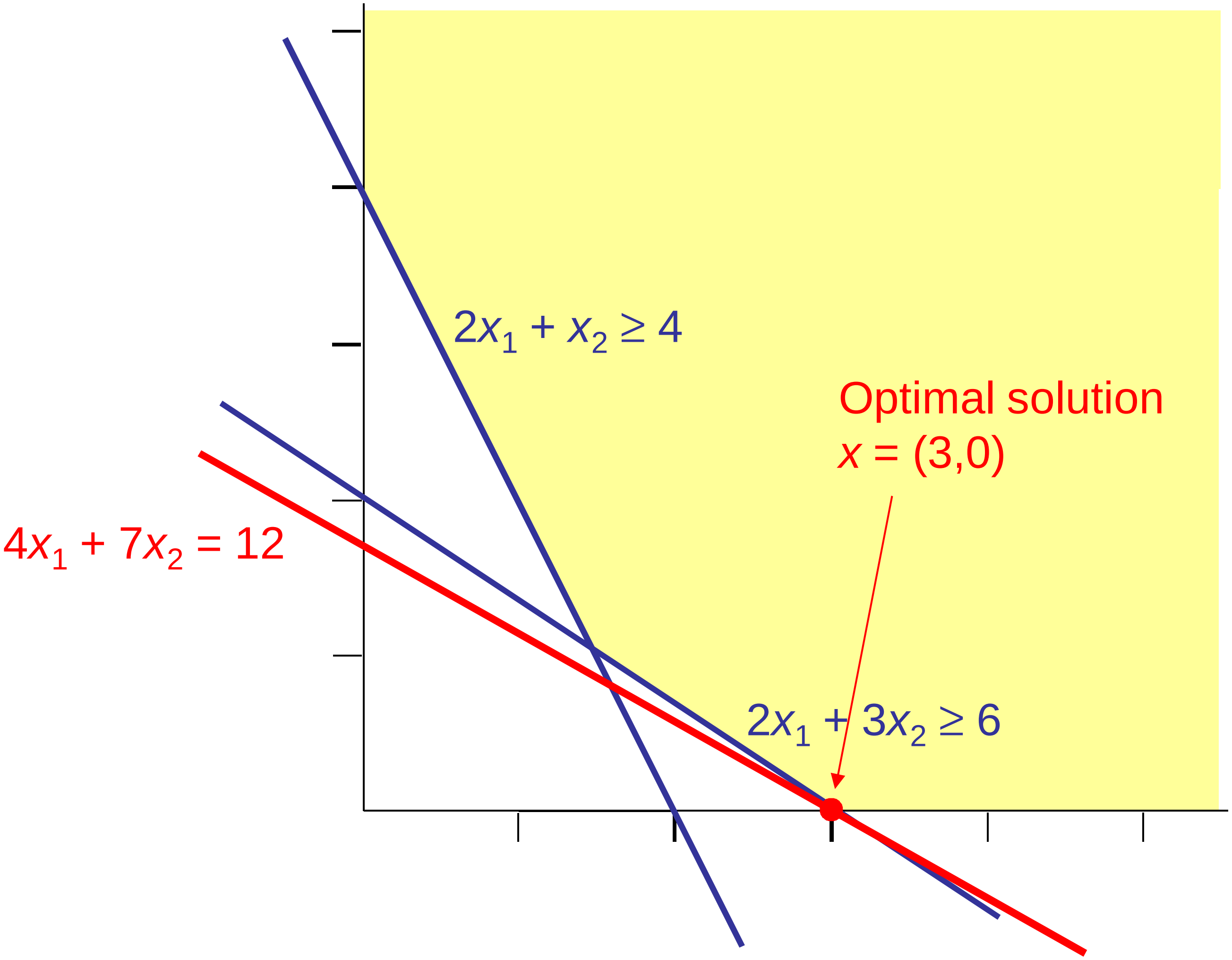
Motivation

- **Linear programming** is remarkably versatile for representing real-world problems.
- LP is by far the most widely used tool for **relaxation**.
- LP relaxations can be strengthened by **cutting planes.**
    - Based on polyhedral analysis.
- LP has an elegant and powerful **duality theory**.
    - Useful for domain filtering, and much else.
- The LP problem is **extremely well solved**.

Algebraic Analysis of LP

An example…

$$\min\ 4x_1 + 7x_2$$
$$2x_1 + 3x_2 \geq 6$$
$$2x_1 + x_2 \geq 4$$
$$x_1, x_2 \geq 0$$



Algebraic Analysis of LP

Rewrite as

$$\begin{aligned} &\min\ 4x_1 + 7x_2 \\ &2x_1 + 3x_2 \ge 6 \\ &2x_1 + x_2 \ge 4 \\ &x_1, x_2 \ge 0 \end{aligned}$$

$$\begin{aligned} &\min\ 4x_1 + 7x_2 \\ &2x_1 + 3x_2 - x_3 = 6 \\ &2x_1 + x_2 - x_4 = 4 \\ &x_1, x_2, x_3, x_4 \ge 0 \end{aligned}$$

In general an LP has the form

$$\begin{aligned} &\min\ cx \\ &Ax = b \\ &x \ge 0 \end{aligned}$$

Algebraic analysis of LP

Write

$$\begin{aligned} &\min cx \\ &\boxed{A}x = b \\ &x \geq 0 \end{aligned}$$

$m \times n$ matrix

as

$$\begin{aligned} &\min c_B x_B + c_N x_N \\ &B x_B + N x_N = b \\ &\boxed{x_B}, \boxed{x_N} \geq 0 \end{aligned}$$

Basic variables

Nonbasic variables

where

$$A = [\boxed{B}\, N]$$

Any set of *m* linearly independent columns of A.

These form a **basis** for the space spanned by the columns.

Algebraic analysis of LP

Write
$$\begin{aligned} &\min cx \\ &Ax = b \\ &x \geq 0 \end{aligned}$$
as
$$\begin{aligned} &\min c_B x_B + c_N x_N \\ &\boxed{Bx_B + Nx_N = b} \\ &x_B, x_N \geq 0 \end{aligned}$$
where
$$A = [B\ N]$$

Solve constraint equation for x_B: $$x_B = B^{-1}b - B^{-1}N\boxed{x_N}$$

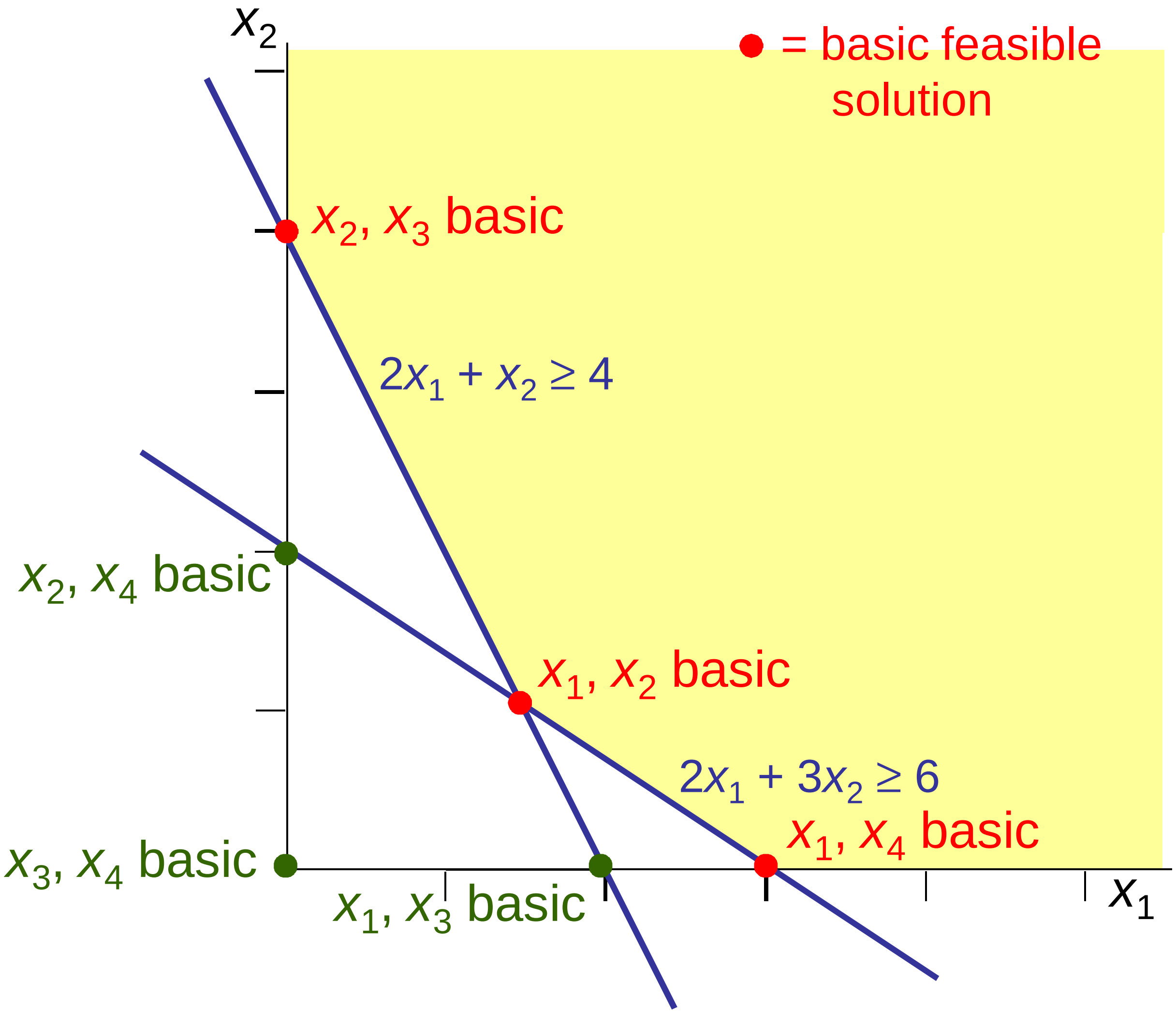
All solutions can be obtained by setting x_N to some value.

The solution is **basic** if $x_N = 0$.

It is a **basic feasible solution** if $x_N = 0$ and $x_B \geq 0$.

Example…

$$\min \ 4x_1 + 7x_2$$
$$2x_1 + 3x_2 - x_3 = 6$$
$$2x_1 + x_2 - x_4 = 4$$
$$x_1, x_2, x_3, x_4 \geq 0$$



Algebraic analysis of LP

Write

$$\begin{aligned} &\min cx \\ &Ax = b \\ &x \geq 0 \end{aligned}$$

as

$$\begin{aligned} &\min \boxed{c_B x_B + c_N x_N} \\ &B x_B + N x_N = b \\ &x_B, x_N \geq 0 \end{aligned}$$

where

$$A = [B\ N]$$

Solve constraint equation for x_B: $x_B = B^{-1}b - B^{-1}Nx_N$

Express cost in terms of nonbasic variables:

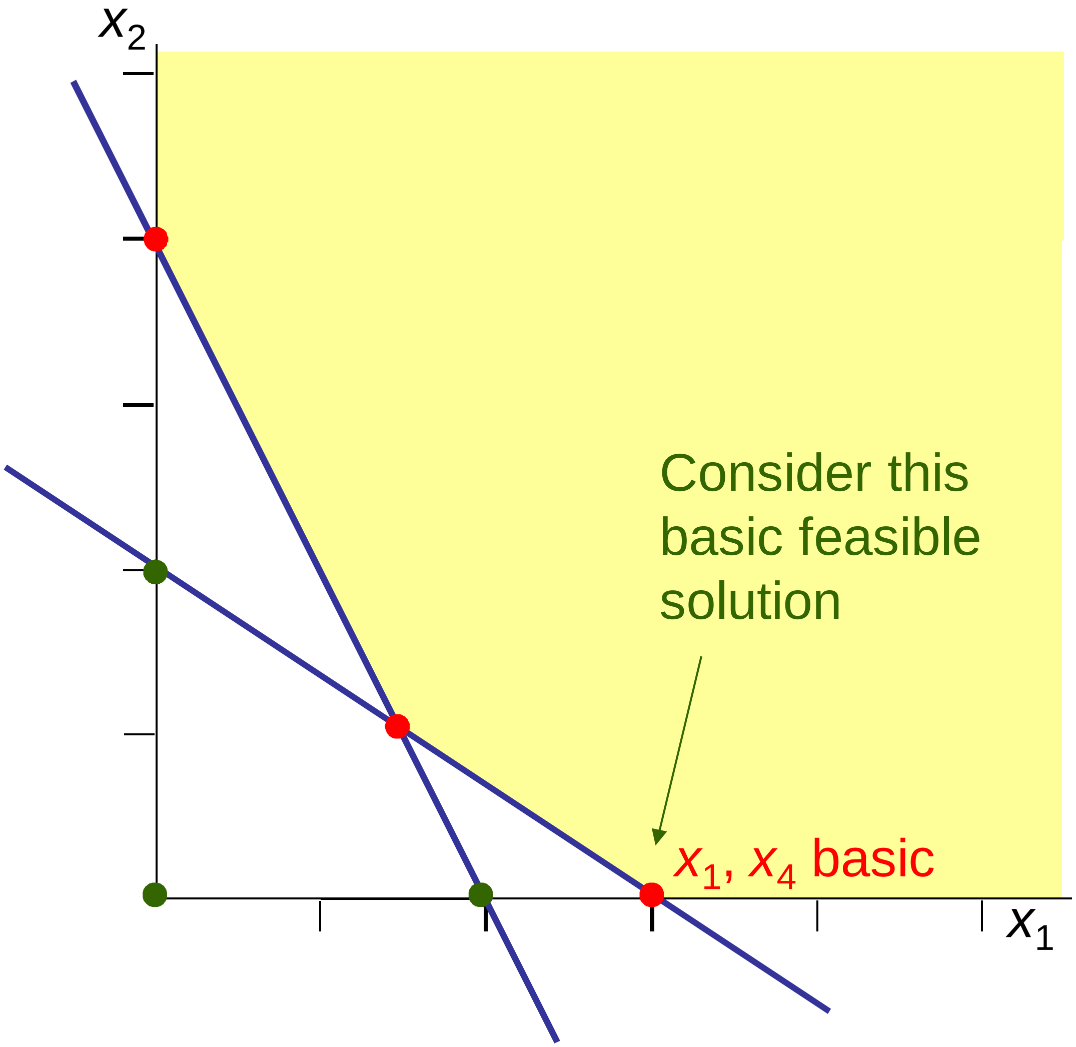
$$c_B B^{-1} b + (\boxed{c_N - c_B B^{-1} N}) x_N$$

Vector of reduced costs

Since $x_N \geq 0$,
basic solution (x_B,0)
is optimal if
reduced costs are
nonnegative.

Example…

$$\min \; 4x_1 + 7x_2$$
$$2x_1 + 3x_2 - x_3 = 6$$
$$2x_1 + x_2 - x_4 = 4$$
$$x_1, x_2, x_3, x_4 \geq 0$$



Consider this basic feasible solution

Example…

Write…

$$\min\ \ 4x_1+7x_2$$
$$2x_1+3x_2-x_3=6$$
$$2x_1+x_2-x_4=4$$
$$x_1,x_2,x_3,x_4\ge 0$$

as…

$$\min\ \overset{c_Bx_B}{\boxed{[4\ \ 0]\begin{bmatrix}x_1\\x_4\end{bmatrix}}}+\overset{c_Nx_N}{\boxed{[7\ \ 0]\begin{bmatrix}x_2\\x_3\end{bmatrix}}}$$

$$Bx_B\ \boxed{\begin{bmatrix}2&0\\2&-1\end{bmatrix}\begin{bmatrix}x_1\\x_4\end{bmatrix}}+\underset{Nx_N}{\boxed{\begin{bmatrix}3&-1\\1&0\end{bmatrix}\begin{bmatrix}x_1\\x_4\end{bmatrix}}}=\underset{b}{\boxed{\begin{bmatrix}6\\4\end{bmatrix}}}$$

$$\begin{bmatrix}x_1\\x_4\end{bmatrix},\begin{bmatrix}x_1\\x_4\end{bmatrix}\ge\begin{bmatrix}0\\0\end{bmatrix}$$

Example…

$$\min \overset{c_B x_B}{\boxed{\begin{bmatrix}4 & 0\end{bmatrix}\begin{bmatrix}x_1\\x_4\end{bmatrix}}} + \overset{c_N x_N}{\boxed{\begin{bmatrix}7 & 0\end{bmatrix}\begin{bmatrix}x_2\\x_3\end{bmatrix}}}$$

$$Bx_B\ \boxed{\begin{bmatrix}2 & 0\\2 & -1\end{bmatrix}\begin{bmatrix}x_1\\x_4\end{bmatrix}} + \underset{Nx_N}{\boxed{\begin{bmatrix}3 & -1\\1 & 0\end{bmatrix}\begin{bmatrix}x_1\\x_4\end{bmatrix}}} = \underset{b}{\boxed{\begin{bmatrix}6\\4\end{bmatrix}}}$$

$$\begin{bmatrix}x_1\\x_4\end{bmatrix}, \begin{bmatrix}x_1\\x_4\end{bmatrix} \geq \begin{bmatrix}0\\0\end{bmatrix}$$

Example…

$$\min \overbrace{\boxed{[4\ \ 0]\begin{bmatrix}x_1\\x_4\end{bmatrix}}}^{c_Bx_B} + \overbrace{\boxed{[7\ \ 0]\begin{bmatrix}x_2\\x_3\end{bmatrix}}}^{c_Nx_N}$$

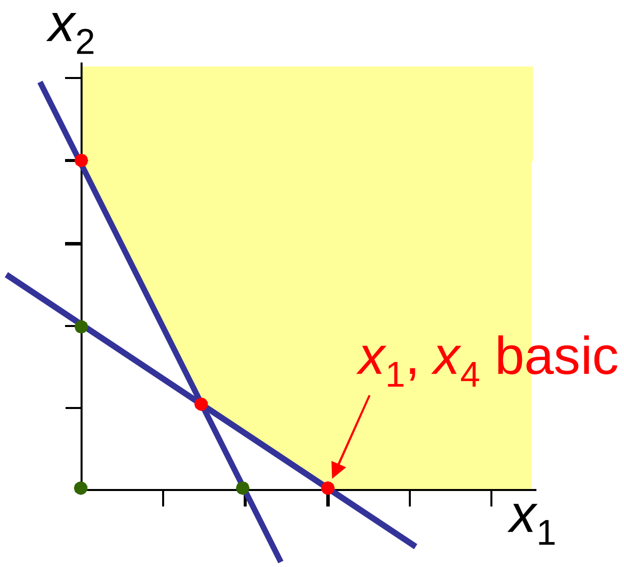
$$Bx_B\ \boxed{\begin{bmatrix}2&0\\2&-1\end{bmatrix}\begin{bmatrix}x_1\\x_4\end{bmatrix}} + \underbrace{\boxed{\begin{bmatrix}3&-1\\1&0\end{bmatrix}\begin{bmatrix}x_1\\x_4\end{bmatrix}}}_{Nx_N} = \underbrace{\boxed{\begin{bmatrix}6\\4\end{bmatrix}}}_{b}$$

$$\begin{bmatrix}x_1\\x_4\end{bmatrix},\begin{bmatrix}x_1\\x_4\end{bmatrix} \geq \begin{bmatrix}0\\0\end{bmatrix}$$

Basic solution is

$$x_B = B^{-1}b - B^{-1}Nx_N = B^{-1}b$$

$$= \begin{bmatrix}x_1\\x_4\end{bmatrix} = \begin{bmatrix}1/2&0\\1&-1\end{bmatrix}\begin{bmatrix}6\\4\end{bmatrix} = \begin{bmatrix}3\\2\end{bmatrix}$$



Example…

$$\min \overset{c_Bx_B}{\boxed{\begin{bmatrix}4 & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_4\end{bmatrix}}} + \overset{c_Nx_N}{\boxed{\begin{bmatrix}7 & 0\end{bmatrix}\begin{bmatrix}x_2\\ x_3\end{bmatrix}}}$$

$$Bx_B \boxed{\begin{bmatrix}2 & 0\\ 2 & -1\end{bmatrix}\begin{bmatrix}x_1\\ x_4\end{bmatrix}} + \underset{Nx_N}{\boxed{\begin{bmatrix}3 & -1\\ 1 & 0\end{bmatrix}\begin{bmatrix}x_1\\ x_4\end{bmatrix}}} = \boxed{\begin{bmatrix}6\\ 4\end{bmatrix}}$$

$$\begin{bmatrix}x_1\\ x_4\end{bmatrix}, \begin{bmatrix}x_1\\ x_4\end{bmatrix} \geq \begin{bmatrix}0\\ 0\end{bmatrix}$$

Basic solution is

$$x_B = B^{-1}b - B^{-1}Nx_N = B^{-1}b$$

$$= \begin{bmatrix}x_1\\ x_4\end{bmatrix} = \begin{bmatrix}1/2 & 0\\ 1 & -1\end{bmatrix}\begin{bmatrix}6\\ 4\end{bmatrix} = \begin{bmatrix}3\\ 2\end{bmatrix}$$

Reduced costs are

$$c_N - c_BB^{-1}N$$

$$= \begin{bmatrix}7 & 0\end{bmatrix} - \begin{bmatrix}4 & 0\end{bmatrix}\begin{bmatrix}1/2 & 0\\ 1 & -1\end{bmatrix}\begin{bmatrix}3 & -1\\ 1 & 0\end{bmatrix}$$

$$= \begin{bmatrix}1 & 2\end{bmatrix} \geq \begin{bmatrix}0 & 0\end{bmatrix}$$

Solution is optimal

Linear Programming Duality

An LP can be viewed as an inference problem…

$$\begin{array}{l} \min\ cx \\ Ax \ge b \\ x \ge 0 \end{array} \quad = \quad \boxed{\begin{array}{l} \max\ v \\ Ax \ge b \boxed{\overset{x\ge 0}{\Rightarrow}} cx \ge v \end{array}}$$

implies

Dual problem: Find the tightest lower bound on the objective function that is implied by the constraints.

An LP can be viewed as an inference problem…

$$\begin{aligned} &\min\ cx & &= & &\max\ v \\ &Ax \ge b & & & &Ax \ge b \overset{x\ge 0}{\Rightarrow} cx \ge v \\ &x \ge 0 & & & & \end{aligned}$$

That is, some **surrogate** (nonnegative linear combination) of $Ax \ge b$ dominates $cx \ge v$

From Farkas Lemma: If $Ax \ge b$, $x \ge 0$ is feasible,

$$Ax \ge b \overset{x\ge 0}{\Rightarrow} cx \ge v \quad \text{iff} \quad \lambda Ax \ge \lambda b\ \boxed{\text{dominates}}\ cx \ge v$$

$$\text{for some } \lambda \ge 0$$

$\lambda A \le c$ and $\lambda b \ge v$

An LP can be viewed as an inference problem…

$$\begin{array}{l} \min \; cx \\ Ax \geq b \\ x \geq 0 \end{array} \quad = \quad \begin{array}{l} \max \; v \\ Ax \geq b \overset{x\geq 0}{\Rightarrow} cx \geq v \end{array} \quad = \quad \begin{array}{l} \max \; \lambda b \\ \lambda A \leq c \\ \lambda \geq 0 \end{array}$$

This is the **classical LP dual**

From Farkas Lemma: If $Ax \geq b$, $x \geq 0$ is feasible,

$$Ax \geq b \overset{x\geq 0}{\Rightarrow} cx \geq v \quad \text{iff} \quad \lambda Ax \geq \lambda b \; \boxed{\text{dominates}} \; cx \geq v$$

for some $\lambda \geq 0$

$\lambda A \leq c$ and $\lambda b \geq v$

This equality is called **strong duality.**

$$\begin{array}{lcl} \min\ cx & = & \max\ \lambda b \\ Ax \geq b & & \lambda A \leq c \\ x \geq 0 & & \lambda \geq 0 \end{array}$$

This is the **classical LP dual**

If $Ax \geq b$, $x \geq 0$ is feasible

Note that the dual of the dual is the **primal** (i.e., the original LP).

Example

Primal		*Dual*	
$\min\ 4x_1 + 7x_2$	$=$	$\max\ 6\lambda_1 + 4\lambda_2$	$= 12$
$2x_1 + 3x_2 \geq 6$	(λ_1)	$2\lambda_1 + 2\lambda_2 \leq 4$	(x_1)
$2x_1 + x_2 \geq 4$	(λ_1)	$3\lambda_1 + \lambda_2 \leq 7$	(x_2)
$x_1, x_2 \geq 0$		$\lambda_1, \lambda_2 \geq 0$	

A dual solution is $(\lambda_1,\lambda_2) = (2,0)$

$$\begin{array}{ll} 2x_1 + 3x_2 \geq 6 & \cdot (\lambda_1 = 2) \\ 2x_1 + x_2 \geq 4 & \cdot (\lambda_2 = 0) \\ \hline \end{array} \quad \text{Dual multipliers}$$

$$4x_1 + 6x_2 \geq 12 \longleftarrow \text{Surrogate}$$

$$\Big\downarrow \text{dominates}$$

$$4x_1 + 7x_2 \geq 12 \longleftarrow \text{Tightest bound on cost}$$

Weak Duality

If x* is feasible in the primal problem

$$\begin{aligned} &\min\ cx \\ &Ax \ge b \\ &x \ge 0 \end{aligned}$$

and λ* is feasible in the dual problem

$$\begin{aligned} &\max\ \lambda b \\ &\lambda A \le c \\ &\lambda \ge 0 \end{aligned}$$

then $cx^* \ge \lambda^* b.$

This is because

$$cx^* \underset{\uparrow}{\ge} \lambda^* A x^* \underset{\uparrow}{\ge} \lambda^* b$$

λ^* is dual feasible and $x^* \ge 0$

x^* is primal feasible and $\lambda^* \ge 0$

Dual multipliers as marginal costs

Suppose we perturb the RHS of an LP (i.e., change the requirement levels):

$$\begin{aligned}&\min\ cx\\ &Ax \geq b + \Delta b\\ &x \geq 0\end{aligned}$$

The dual of the perturbed LP has the same constraints at the original LP:

$$\begin{aligned}&\max\ \lambda(b + \Delta b)\\ &\lambda A \leq c\\ &\lambda \geq 0\end{aligned}$$

So an optimal solution λ^* of the original dual is feasible in the perturbed dual.

Dual multipliers as marginal costs

Suppose we perturb the RHS of an LP (i.e., change the requirement levels):

$$\min\ cx$$
$$Ax \geq b + \Delta b$$
$$x \geq 0$$

By weak duality, the optimal value of the perturbed LP is at least $\lambda^*(b + \Delta\text{b}) = \boxed{\lambda^* b} + \lambda^* \Delta b.$

Optimal value of original LP, by strong duality.

So λ_i^* is a lower bound on the marginal cost of increasing the i-th requirement by one unit ($\Delta b_i = 1$).

If $\lambda_i^* > 0$, the i-th constraint must be tight **(complementary slackness).**

Dual of an LP in equality form

Primal

$$\begin{aligned}
&\min c_B x_B + c_N x_N \\
&B x_B + N x_N = b \qquad (\lambda) \\
&x_B, x_N \geq 0
\end{aligned}$$

Dual

$$\begin{aligned}
&\max \lambda b \\
&\lambda B \leq c_B \qquad (x_B) \\
&\lambda N \leq c_N \qquad (x_B) \\
&\lambda \text{ unrestricted}
\end{aligned}$$

Dual of an LP in equality form

Primal

$$\min c_B x_B + c_N x_N$$

$$B x_B + N x_N = b \qquad (\lambda)$$

$$x_B, x_N \geq 0$$

Dual

$$\max \lambda b$$

$$\lambda B \leq c_B \qquad (x_B)$$

$$\lambda N \leq c_N \qquad (x_B)$$

$$\lambda \text{ unrestricted}$$

Recall that reduced cost vector is $c_N - \underset{\lambda}{\boxed{c_B B^{-1}}} N = c_N - \lambda N$

this solves the dual
if (x_B,0) solves the primal

Dual of an LP in equality form

Primal

$$\min c_B x_B + c_N x_N$$
$$B x_B + N x_N = b \qquad (\lambda)$$
$$x_B, x_N \geq 0$$

Dual

$$\max \lambda b$$
$$\lambda B \leq c_B \qquad (x_B)$$
$$\lambda N \leq c_N \qquad (x_B)$$
$$\lambda \text{ unrestricted}$$

Recall that reduced cost vector is $c_N - \underbrace{\boxed{c_B B^{-1}}}_{\lambda} N = c_N - \lambda N$

this solves the dual
if (x_B,0) solves the primal

Check:

$$\lambda B = c_B B^{-1} B = c_B$$
$$\lambda N = c_B B^{-1} N \leq c_N$$

Because reduced cost is nonnegative
at optimal solution (x_B,0).

Dual of an LP in equality form

Primal

$$\begin{aligned}&\min c_B x_B + c_N x_N\\&B x_B + N x_N = b \qquad (\lambda)\\&x_B, x_N \geq 0\end{aligned}$$

Dual

$$\begin{aligned}&\max \lambda b\\&\lambda B \leq c_B \qquad (x_B)\\&\lambda N \leq c_N \qquad (x_B)\\&\lambda \text{ unrestricted}\end{aligned}$$

Recall that reduced cost vector is $c_N - \underset{\lambda}{\boxed{c_B B^{-1}}} N = c_N - \lambda N$

this solves the dual
if (x_B,0) solves the primal

In the example,

$$\lambda = c_B B^{-1} = [4 \quad 0]\begin{bmatrix} 1/2 & 0 \\ 1 & -1 \end{bmatrix} = [2 \quad 0]$$

Dual of an LP in equality form

Primal

$$\min c_B x_B + c_N x_N$$
$$B x_B + N x_N = b \qquad (\lambda)$$
$$x_B, x_N \geq 0$$

Dual

$$\max \lambda b$$
$$\lambda B \leq c_B \qquad (x_B)$$
$$\lambda N \leq c_N \qquad (x_B)$$
$$\lambda \text{ unrestricted}$$

Recall that reduced cost vector is $c_N - \underbrace{\boxed{c_B B^{-1}}}_{\lambda} N = c_N - \lambda N$

Note that the reduced cost of an individual variable x_j is $r_j = c_j - \lambda \boxed{A_j}$

Column *j* of *A*

LP-based Domain Filtering

Let $\begin{array}{l} \min\ cx \\ Ax \geq b \\ x \geq 0 \end{array}$ be an LP relaxation of a CP problem.

- One way to filter the domain of x_j is to minimize and maximize x_j subject to $Ax \geq b$, $x \geq 0$.
  - This is time consuming.
- A faster method is to use **dual multipliers** to derive valid inequalities.
  - A special case of this method uses **reduced costs** to bound or fix variables.
  - **Reduced-cost variable fixing** is a widely used technique in OR.

Suppose:

$$\begin{aligned} &\min\ cx \\ &Ax \geq b \\ &x \geq 0 \end{aligned}$$

has optimal solution x^*, optimal value v^*, and optimal dual solution λ^*.

…and $\lambda_i^* > 0$, which means the i-th constraint is tight (complementary slackness);

…and the LP is a relaxation of a CP problem;

…and we have a feasible solution of the CP problem with value U, so that U is an upper bound on the optimal value.

Supposing $\begin{aligned} &\min\ cx \\ &Ax \geq b \\ &x \geq 0 \end{aligned}$ has optimal solution x^*, optimal value v^*, and optimal dual solution λ^*:

If x were to change to a value other than x^*, the LHS of i-th constraint $A^i x \geq b_i$ would change by some amount Δb_i.

Since the constraint is tight, this would increase the optimal value as much as changing the constraint to $A^i x \geq b_i + \Delta b_i$.

So it would increase the optimal value at least $\lambda_i^* \Delta b_i$.

Supposing $\begin{aligned}&\min\ cx\\ &Ax \geq b\\ &x \geq 0\end{aligned}$ has optimal solution x^*, optimal value v^*, and optimal dual solution λ^*:

We have found: a change in x that changes $A^i x$ by Δb_i increases the optimal value of LP at least $\lambda_i^* \Delta b_i$.

Since optimal value of the LP ≤ optimal value of the CP ≤ U, we have $\lambda_i^* \Delta b_i \leq U - v^*$, or

$$\Delta b_i \leq \frac{U - v^*}{\lambda_i^*}$$

Supposing $\begin{array}{l} \min\ cx \\ Ax \geq b \\ x \geq 0 \end{array}$ has optimal solution x^*, optimal value v^*, and optimal dual solution λ^*:

We have found: a change in x that changes $A^i x$ by Δb_i increases the optimal value of LP at least $\lambda_i^* \Delta b_i$.

Since optimal value of the LP ≤ optimal value of the CP ≤ U, we have $\lambda_i^* \Delta b_i \leq U - v^*$, or

$$\Delta b_i \leq \frac{U - v^*}{\lambda_i^*}$$

Since $\Delta b_i = A^i x - A^i x^* = A^i x - b_i$, this implies the inequality

$$A^i x \leq b_i + \frac{U - v^*}{\lambda_i^*}$$

…which can be propagated.

Example

$$\min\ 4x_1 + 7x_2$$

$$2x_1 + 3x_2 \ge 6 \qquad (\lambda_1 = 2)$$

$$2x_1 + x_2 \ge 4 \qquad (\lambda_1 = 0)$$

$$x_1, x_2 \ge 0$$

Suppose we have a feasible solution of the original CP with value *U* = 13.

Since the first constraint is tight, we can propagate the inequality

$$A^1 x \le b_1 + \frac{U - v^*}{\lambda_1^*}$$

or $$2x_1 + 3x_2 \le 6 + \frac{13 - 12}{2} = 6.5$$

Reduced-cost domain filtering

Suppose $x_j^* = 0$, which means the constraint $x_j \ge 0$ is tight.

The inequality $A^i x \le b_i + \dfrac{U - v^*}{\lambda_i^*}$ becomes $x_j \le \dfrac{U - v^*}{\boxed{r_j}}$

The dual multiplier for $x_j \ge 0$ is the reduced cost r_j of x_j, because increasing x_j (currently 0) by 1 increases optimal cost by r_j.

Similar reasoning can bound a variable below when it is at its upper bound.

Example

$$\min\ 4x_1 + 7x_2$$
$$2x_1 + 3x_2 \geq 6 \qquad (\lambda_1 = 2)$$
$$2x_1 + x_2 \geq 4 \qquad (\lambda_1 = 0)$$
$$x_1, x_2 \geq 0$$

Suppose we have a feasible solution of the original CP with value U = 13.

Since x_2* = 0, we have $x_2 \leq \dfrac{U - v^*}{r_2}$

or $x_2 \leq \dfrac{13 - 12}{2} = 0.5$

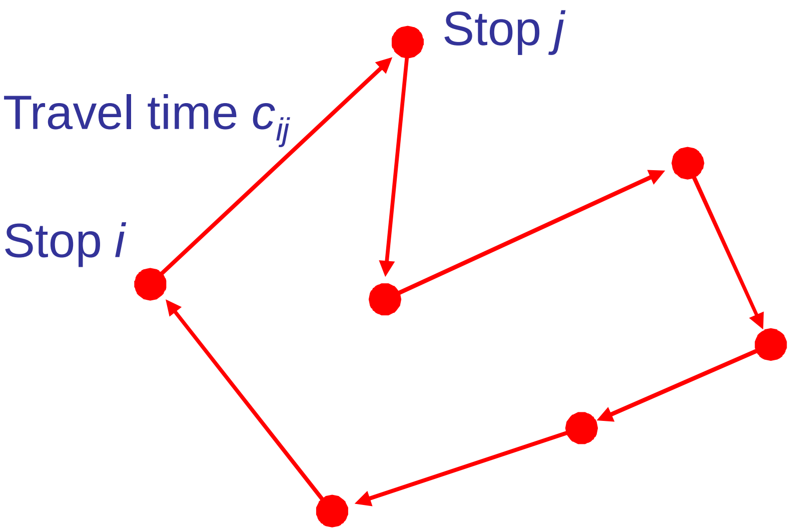
If x_2 is required to be integer, we can fix it to zero. This is **reduced-cost variable fixing.**

Example: Single-Vehicle Routing

A vehicle must make several stops and return home, perhaps subject to time windows.

The objective is to find the order of stops that minimizes travel time.

This is also known as the **traveling salesman problem with time windows**.



Assignment Relaxation

$$\min \sum_{ij} c_{ij} \boxed{x_{ij}}$$

= 1 if stop *i* immediately precedes stop *j*

$$\sum_{j} x_{ij} = \sum_{j} x_{ji} = 1, \text{ all } i$$

Stop *i* is preceded and followed by exactly one stop.

$$x_{ij} \in \{0,1\}, \text{ all } i, j$$

Assignment Relaxation

$$\min \sum_{ij} c_{ij} x_{ij}$$

= 1 if stop *i* immediately precedes stop *j*

$$\sum_j x_{ij} = \sum_j x_{ji} = 1, \text{ all } i$$

Stop *i* is preceded and
followed by exactly one stop.

$$0 \le x_{ij} \le 1, \text{ all } i, j$$

Because this problem is **totally unimodular**, it can be solved as an LP.

The relaxation provides a very weak lower bound on the optimal value.

But **reduced-cost variable fixing** can be very useful in a CP context.

Lagrangean Relaxation

Lagrangean Duality
Properties of the Lagrangean Dual
Example: Fast Linear Programming
Domain Filtering
Example: Continuous Global Optimization

Motivation

- **Lagrangean relaxation** can provide better bounds than LP relaxation.

- The **Lagrangean dual** generalizes LP duality.

- It provides **domain filtering** analogous to that based on LP duality.
  - This is a technique in **continuous global optimization**.

- Lagrangean relaxation gets rid of troublesome constraints by **dualizing** them.
  - That is, moving them into the objective function.
  - The Lagrangean relaxation may **decouple**.

Lagrangean Duality

Consider an inequality-constrained problem

$$\min \; f(x)$$

$$g(x) \geq 0$$

Hard constraints

$$x \in S$$

Easy constraints

The object is to get rid of (**dualize**) the hard constraints by moving them into the objective function.

Lagrangean Duality

Consider an inequality-constrained problem

$$\begin{aligned} &\min\ f(x) \\ &g(x) \geq 0 \\ &x \in S \end{aligned}$$

It is related to an inference problem

$$\max\ v$$

$$g(x) \geq b \boxed{\overset{x\in S}{\Rightarrow}} f(x) \geq v$$

implies

Lagrangean Dual problem: Find the tightest lower bound on the objective function that is implied by the constraints.

Primal

$$\min\ f(x)$$

$$g(x)\geq 0$$

$$x\in S$$

Dual

$$\max\ v$$

$$g(x)\geq b \overset{x\in S}{\Rightarrow} f(x)\geq v$$

Surrogate

Let us say that

$g(x)\geq 0 \overset{x\in S}{\Rightarrow} f(x)\geq v$ iff $\boxed{\lambda g(x)\geq 0}$ $\boxed{\text{dominates}}$ $f(x)-v\geq 0$

for some $\lambda\geq 0$

$\lambda g(x)\leq f(x) - v$ for all $x\in S$

That is, $v\leq f(x) - \lambda g(x)$ for all $x\in S$

Primal

$$\min \; f(x)$$

$$g(x) \ge 0$$

$$x \in S$$

Dual

$$\max \; v$$

$$g(x) \ge b \overset{s\in S}{\Rightarrow} f(x) \ge v$$

Let us say that

Surrogate

$$g(x) \ge 0 \overset{x\in S}{\Rightarrow} f(x) \ge v \quad \text{iff} \quad \boxed{\lambda g(x) \ge 0} \; \boxed{\text{dominates}} \; f(x) - v \ge 0$$

for some $\lambda \ge 0$

$\lambda g(x) \le f(x) - v$ for all $x \in S$

That is, $v \le f(x) - \lambda g(x)$ for all $x \in S$

Or $\quad v \le \min\limits_{x\in S} \{ f(x) - \lambda g(x) \}$

Primal

$$\min \; f(x)$$
$$g(x) \geq 0$$
$$x \in S$$

Dual

$$\max \; v$$
$$g(x) \geq b \overset{s \in S}{\Rightarrow} f(x) \geq v$$

Surrogate

Let us say that

$$g(x) \geq 0 \overset{x \in S}{\Rightarrow} f(x) \geq v \quad \text{iff} \quad \boxed{\lambda g(x) \geq 0} \; \boxed{\text{dominates}} \; f(x) - v \geq 0$$

for some $\lambda \geq 0$

$\lambda g(x) \leq f(x) - v$ for all $x \in S$

That is, $v \leq f(x) - \lambda g(x)$ for all $x \in S$

Or $\quad v \leq \min\limits_{x \in S} \{ f(x) - \lambda g(x) \}$

So the dual becomes

$$\max \; v$$
$$v \leq \min_{x \in S} \{ f(x) - \lambda g(x) \} \text{ for some } \lambda \geq 0$$

Now we have…

Primal

$$\min\ f(x)$$
$$\boxed{g(x)\ge 0}$$
$$x\in S$$

These constraints are **dualized**

Dual

$$\max\ v$$
$$v\le\min_{x\in S}\{f(x)-\lambda g(x)\}\ \text{ for some } \lambda\ge 0$$

or

$$\max_{\lambda\ge 0}\ \theta(\lambda)$$

where

$$\theta(\lambda)=\min_{x\in S}\{f(x)-\lambda g(x)\}$$

Lagrangean relaxation

Vector of **Lagrange multipliers**

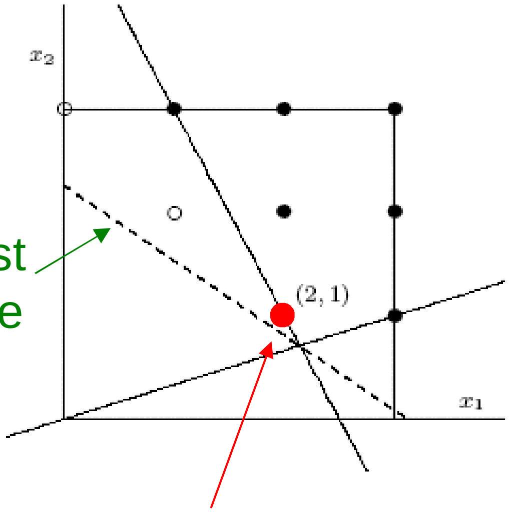
The Lagrangean dual can be viewed as the problem of finding the Lagrangean relaxation that gives the tightest bound.

Example

$$
\begin{aligned}
&\min\ \ 3x_1 + 4x_2 \\
&-x_1 + 3x_2 \ge 0 \\
&2x_1 + x_2 - 5 \ge 0 \\
&x_1, x_2 \in \{0,1,2,3\}
\end{aligned}
$$

The Lagrangean relaxation is

$$
\begin{aligned}
\theta(\lambda_1,\lambda_2) &= \min_{x_j\in\{0,\ldots,3\}} \{3x_1 + 4x_2 - \lambda_1(-x_1+3x_2) - \lambda_2(2x_1+x_2-5)\} \\
&= \min_{x_j\in\{0,\ldots,3\}} \{(3+\lambda_1-2\lambda_2)x_1 + (4-3\lambda_1-\lambda_2)x_2 + 5\lambda_2\}
\end{aligned}
$$



Strongest surrogate

Optimal solution (2,1)

The Lagrangean relaxation is easy to solve for any given λ_1, λ_2:

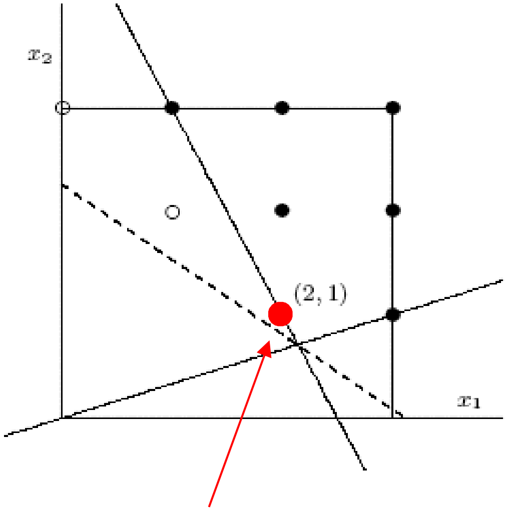
$$
x_1 = \begin{cases} 0 & \text{if } 3+\lambda_1-2\lambda_2 \ge 0 \\ 3 & \text{otherwise} \end{cases}
$$

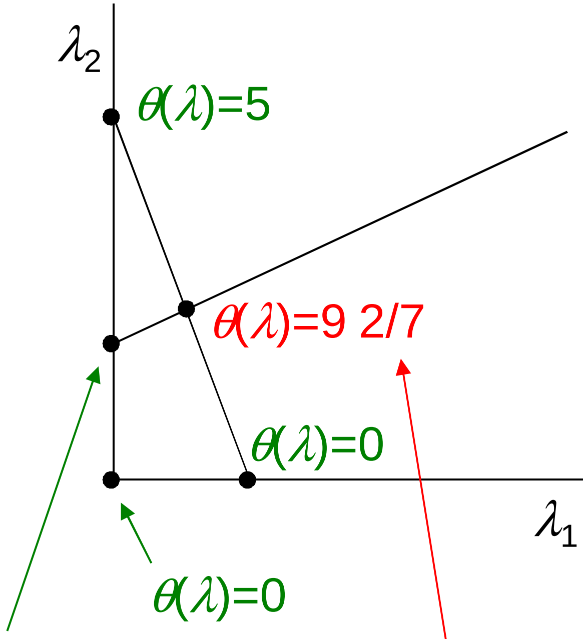
$$
x_2 = \begin{cases} 0 & \text{if } 4-3\lambda_1-\lambda_2 \ge 0 \\ 3 & \text{otherwise} \end{cases}
$$

Example

$\theta(\lambda_1,\lambda_2)$ is piecewise linear and concave.

$$\begin{aligned}&\min\ 3x_1+4x_2\\&-x_1+3x_2\ge 0\\&2x_1+x_2-5\ge 0\\&x_1,x_2\in\{0,1,2,3\}\end{aligned}$$





$\theta(\lambda)$=5

$\theta(\lambda)$=9 2/7

$\theta(\lambda)$=0

$\theta(\lambda)$=0

$\theta(\lambda)$=7.5

Solution of Lagrangean dual:

(λ_1,λ_2) = (5/7, 13/7), $\theta(\lambda)$ = 9 2/7

Optimal solution (2,1)
Value = 10

Note **duality gap** between 10 and 9 2/7 (no strong duality).

Example

$$\begin{aligned}&\min\ 3x_1 + 4x_2\\ &-x_1 + 3x_2 \geq 0\\ &2x_1 + x_2 - 5 \geq 0\\ &x_1, x_2 \in \{0,1,2,3\}\end{aligned}$$

Note: in this example, the Lagrangean dual provides the same bound (9 2/7) as the continuous relaxation of the IP.

This is because the Lagrangean relaxation can be solved as an LP:

$$\theta(\lambda_1,\lambda_2) = \min_{x_j\in\{0,\ldots,3\}} \{(3+\lambda_1 - 2\lambda_2)x_1 + (4 - 3\lambda_1 - \lambda_2)x_2 + 5\lambda_2\}$$

$$= \min_{0\leq x_j \leq 3} \{(3+\lambda_1 - 2\lambda_2)x_1 + (4 - 3\lambda_1 - \lambda_2)x_2 + 5\lambda_2\}$$

Lagrangean duality is useful when the Lagrangean relaxation is tighter than an LP but nonetheless easy to solve.

Properties of the Lagrangean dual

Weak duality: For any feasible x^* and any $\lambda^* \geq 0$, $f(x^*) \geq \theta(\lambda^*)$.

In particular,

$$\begin{aligned} &\min\ f(x) \quad \geq \quad \max_{\lambda \geq 0} \theta(\lambda) \\ &g(x) \geq 0 \\ &x \in S \end{aligned}$$

Concavity: $\theta(\lambda)$ is concave. It can therefore be maximized by local search methods.

Complementary slackness: If x^* and λ^* are optimal, and there is no duality gap, then $\lambda^* g(x^*) = 0$.

Solving the Lagrangean dual

Let λ^k be the kth iterate, and let $\lambda^{k+1} = \lambda^k + \alpha_k \boxed{\xi^k}$

Subgradient of $\theta(\lambda)$ at $\lambda = \lambda^k$

If x^k solves the Lagrangean relaxation for $\lambda = \lambda^k$, then $\xi^k = g(x^k)$.

This is because $\theta(\lambda) = f(x^k) + \lambda g(x^k)$ at $\lambda = \lambda^k$.

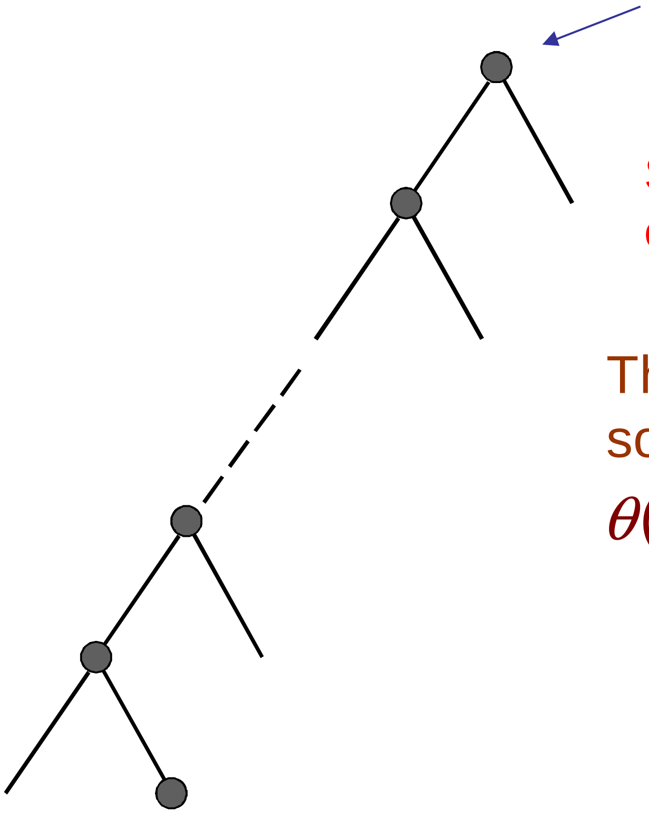
The stepsize α_k must be adjusted so that the sequence converges but not before reaching a maximum.

Example: Fast Linear Programming

- In CP contexts, it is best to process each node of the search tree very rapidly.

- Lagrangean relaxation may allow very fast calculation of a lower bound on the optimal value of the LP relaxation at each node.

- The idea is to solve the Lagrangean dual at the root node (which is an LP) and use the same Lagrange multipliers to get an LP bound at other nodes.



At root node, solve

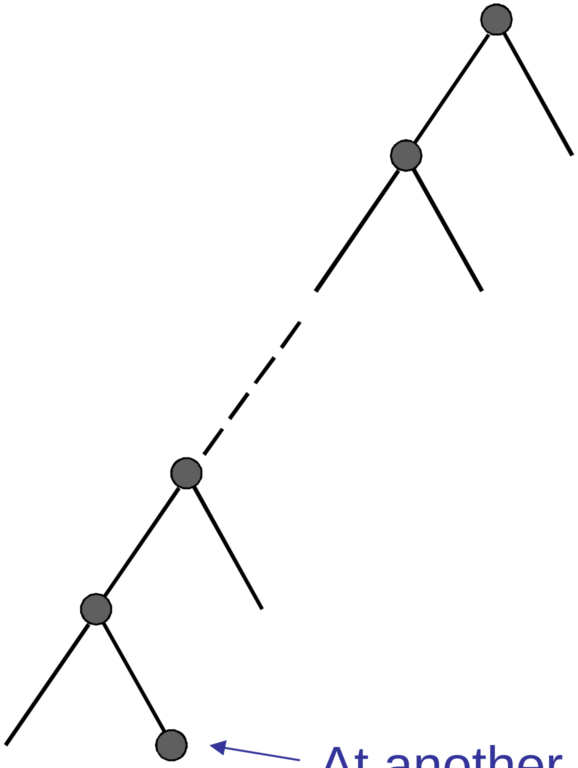
$$\begin{aligned} \min\ & cx \\ & Ax \ge b \quad (\lambda) \\ & Dx \ge d \\ & x \ge 0 \end{aligned}$$

Dualize

Special structure, e.g. variable bounds

The (partial) LP dual solution λ^* solves the Lagrangean dual in which

$$\theta(\lambda) = \min_{\substack{Dx \ge d \\ x \ge 0}} \{cx - \lambda(Ax - b)\}$$



At root node, solve

$$\begin{array}{ll} \min\ cx & \\ Ax \geq b & (\lambda) \\ Dx \geq d & \\ x \geq 0 & \end{array}$$

Dualize

Special structure, e.g. variable bounds

The (partial) LP dual solution λ^* solves the Lagrangean dual in which

$$\theta(\lambda) = \min_{\substack{Dx \geq d \\ x \geq 0}} \{cx - \lambda(Ax - b)\}$$

At another node, the LP is

$$\begin{array}{ll} \min\ cx & \\ Ax \geq b & (\lambda) \\ Dx \geq d & \\ Hx \geq h & \\ x \geq 0 & \end{array}$$

Branching constraints, etc.

Here $\theta(\lambda^*)$ is still a lower bound on the optimal value of the LP and can be quickly calculated by solving a specially structured LP.

Domain Filtering

Suppose:

$$\begin{aligned} &\min\ f(x) \\ &g(x) \geq 0 \\ &x \in S \end{aligned}$$

has optimal solution x^*, optimal value v^*, and optimal Lagrangean dual solution λ^*.

…and $\lambda_i^* > 0$, which means the i-th constraint is tight (complementary slackness);

…and the problem is a relaxation of a CP problem;

…and we have a feasible solution of the CP problem with value U, so that U is an upper bound on the optimal value.

Supposing $\begin{array}{l}\min\ f(x) \\ g(x) \geq 0 \\ x \in S\end{array}$ has optimal solution x^*, optimal value v^*, and optimal Lagrangean dual solution λ^*:

If x were to change to a value other than x^*, the LHS of i-th constraint $g_i(x) \geq 0$ would change by some amount Δ_i.

Since the constraint is tight, this would increase the optimal value as much as changing the constraint to $g_i(x) - \Delta_i \geq 0$.

So it would increase the optimal value at least $\lambda_i^* \Delta_i$.

(It is easily shown that Lagrange multipliers are marginal costs. Dual multipliers for LP are a special case of Lagrange multipliers.)

Supposing
$$\begin{aligned}&\min\ f(x)\\ &g(x)\geq 0\\ &x\in S\end{aligned}$$
has optimal solution x^*, optimal value v^*, and optimal Lagrangean dual solution λ^*:

We have found: a change in x that changes $g_i(x)$ by Δ_i increases the optimal value at least $\lambda_i^*\Delta_i$.

Since optimal value of this problem ≤ optimal value of the CP ≤ U, we have $\lambda_i^*\Delta_i \leq U - v^*$, or

$$\Delta_i \leq \frac{U - v^*}{\lambda_i^*}$$

Supposing $\begin{array}{l} \min\ f(x) \\ g(x) \geq 0 \\ x \in S \end{array}$ has optimal solution x^*, optimal value v^*, and optimal Lagrangean dual solution λ^*:

We have found: a change in x that changes $g_i(x)$ by Δ_i increases the optimal value at least $\lambda_i^*\Delta_i$.

Since optimal value of this problem ≤ optimal value of the CP ≤ U, we have $\lambda_i^*\Delta_i \leq U - v^*$, or

$$\Delta_i \leq \frac{U - v^*}{\lambda_i^*}$$

Since $\Delta_i = g_i(x) - g_i(x^*) = g_i(x)$, this implies the inequality

$$g_i(x) \leq \frac{U - v^*}{\lambda_i^*}$$

…which can be propagated.

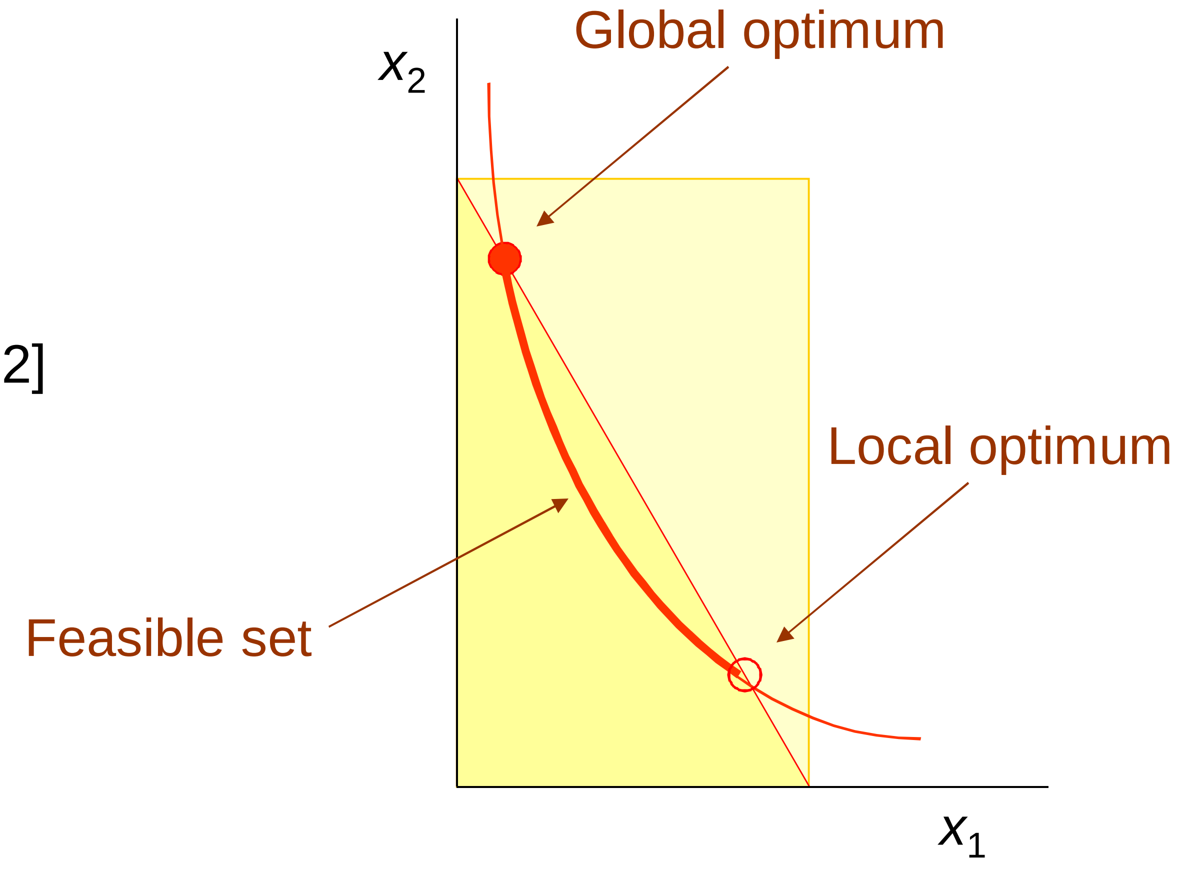
Example: Continuous Global Optimization

- Some of the best continuous global solvers (e.g., BARON) combine OR-style relaxation with CP-style interval arithmetic and domain filtering.

- These methods can be combined with domain filtering based on Lagrange multipliers.

Continuous Global Optimization

$$\max \quad x_1 + x_2$$
$$4x_1x_2 = 1$$
$$2x_1 + x_2 \le 2$$
$$x_1 \in [0,1], \quad x_2 \in [0,2]$$

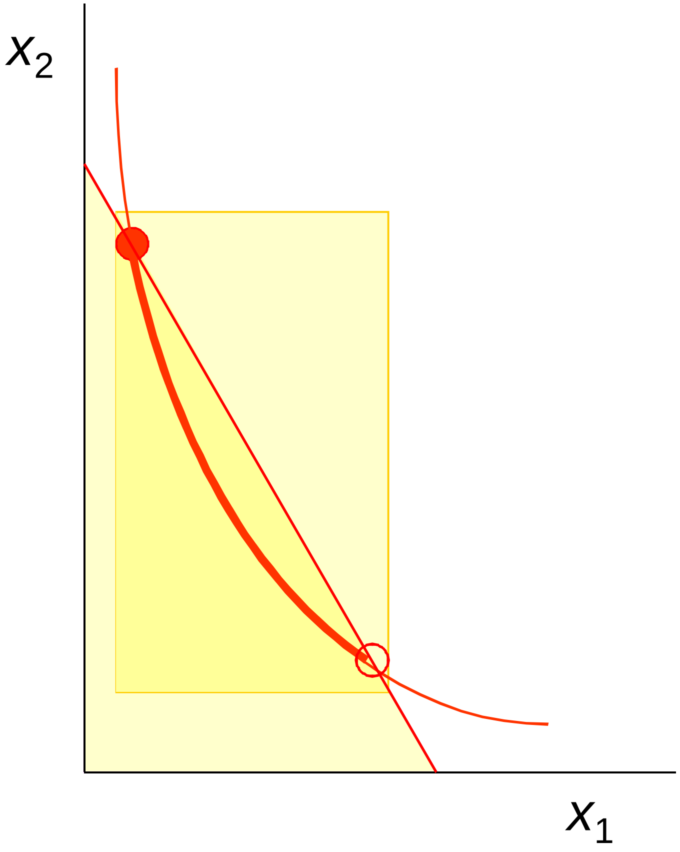


To solve it:

- **Search**: split interval domains of x_1, x_2.
  - Each **node** of search tree is a problem restriction.
- **Propagation:** Interval propagation, domain filtering.
  - Use **Lagrange multipliers** to infer valid inequality for propagation.
  - **Reduced-cost variable** fixing is a special case.
- **Relaxation:** Use function **factorization** to obtain linear continuous relaxation.

Interval propagation

Propagate intervals
[0,1], [0,2]
through constraints
to obtain
[1/8,7/8], [1/4,7/4]



Relaxation (function factorization)

Factor complex functions into elementary functions that have known linear relaxations.

Write $4x_1x_2 = 1$ as $4y = 1$ where $y = x_1x_2$.

This factors $4x_1x_2$ into linear function $4y$ and bilinear function x_1x_2.

Linear function $4y$ is its own linear relaxation.

Relaxation (function factorization)

Factor complex functions into elementary functions that have known linear relaxations.

Write $4x_1x_2 = 1$ as $4y = 1$ where $y = x_1x_2$.

This factors $4x_1x_2$ into linear function $4y$ and bilinear function x_1x_2.

Linear function $4y$ is its own linear relaxation.

Bilinear function $y = x_1x_2$ has relaxation:

$$\underline{x}_2x_1 + \underline{x}_1x_2 - \underline{x}_1\underline{x}_2 \le y \le \underline{x}_2x_1 + \overline{x}_1x_2 - \overline{x}_1\underline{x}_2$$

$$\overline{x}_2x_1 + \overline{x}_1x_2 - \overline{x}_1\overline{x}_2 \le y \le \overline{x}_2x_1 + \underline{x}_1x_2 - \underline{x}_1\overline{x}_2$$

where domain of x_j is $[\underline{x}_j, \overline{x}_j]$

Relaxation (function factorization)

The linear relaxation becomes:

$$\min \;\; x_1 + x_2$$

$$4y = 1$$

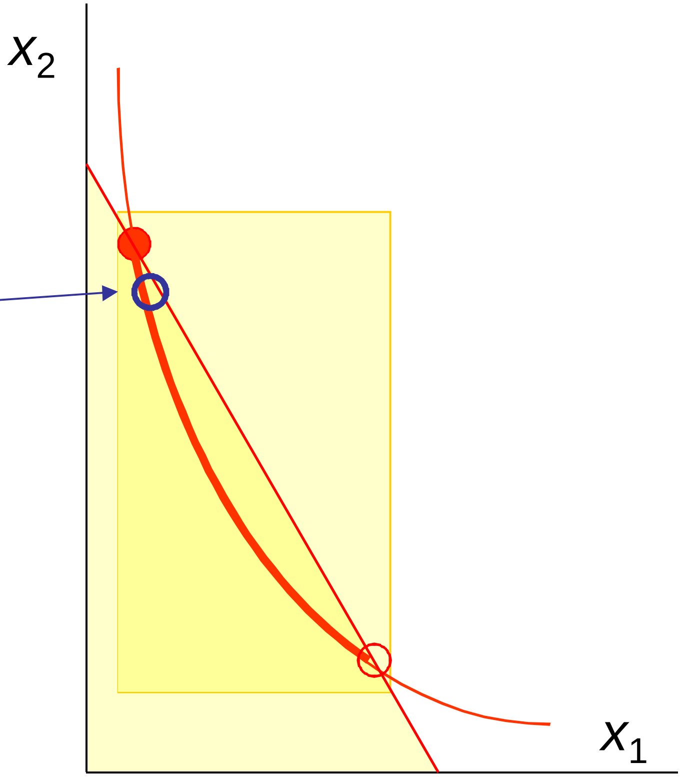
$$2x_1 + x_2 \le 2$$

$$\underline{x}_2 x_1 + \underline{x}_1 x_2 - \underline{x}_1 \underline{x}_2 \le y \le \underline{x}_2 x_1 + \bar{x}_1 x_2 - \bar{x}_1 \underline{x}_2$$

$$\bar{x}_2 x_1 + \bar{x}_1 x_2 - \bar{x}_1 \bar{x}_2 \le y \le \bar{x}_2 x_1 + \underline{x}_1 x_2 - \underline{x}_1 \bar{x}_2$$

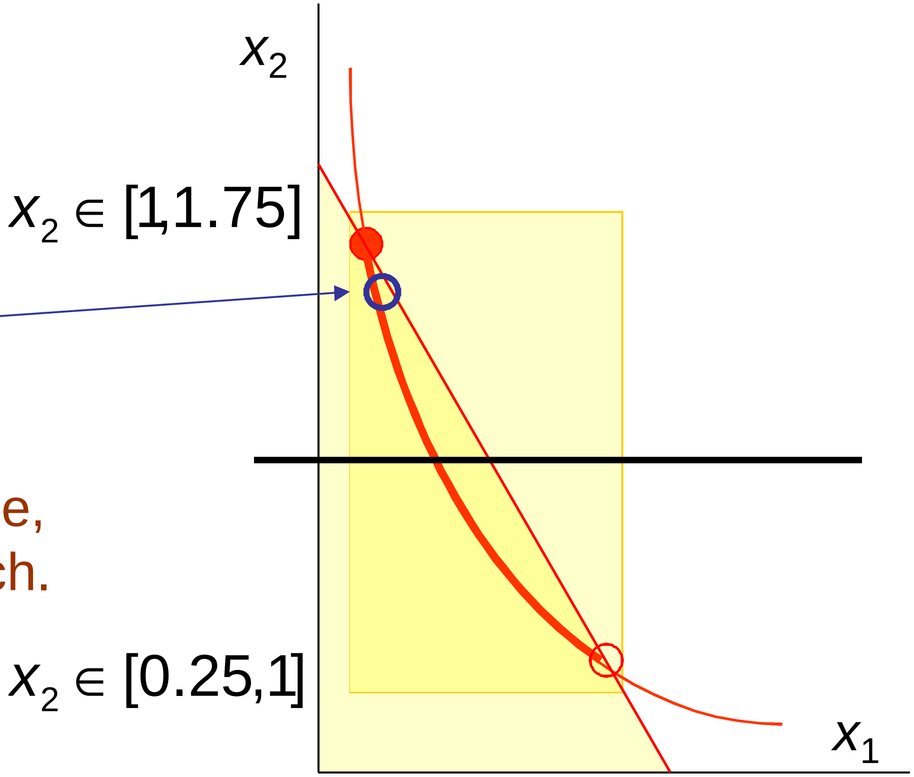
$$\underline{x}_j \le x_j \le \bar{x}_j, \;\; j = 1,2$$

Relaxation (function factorization)



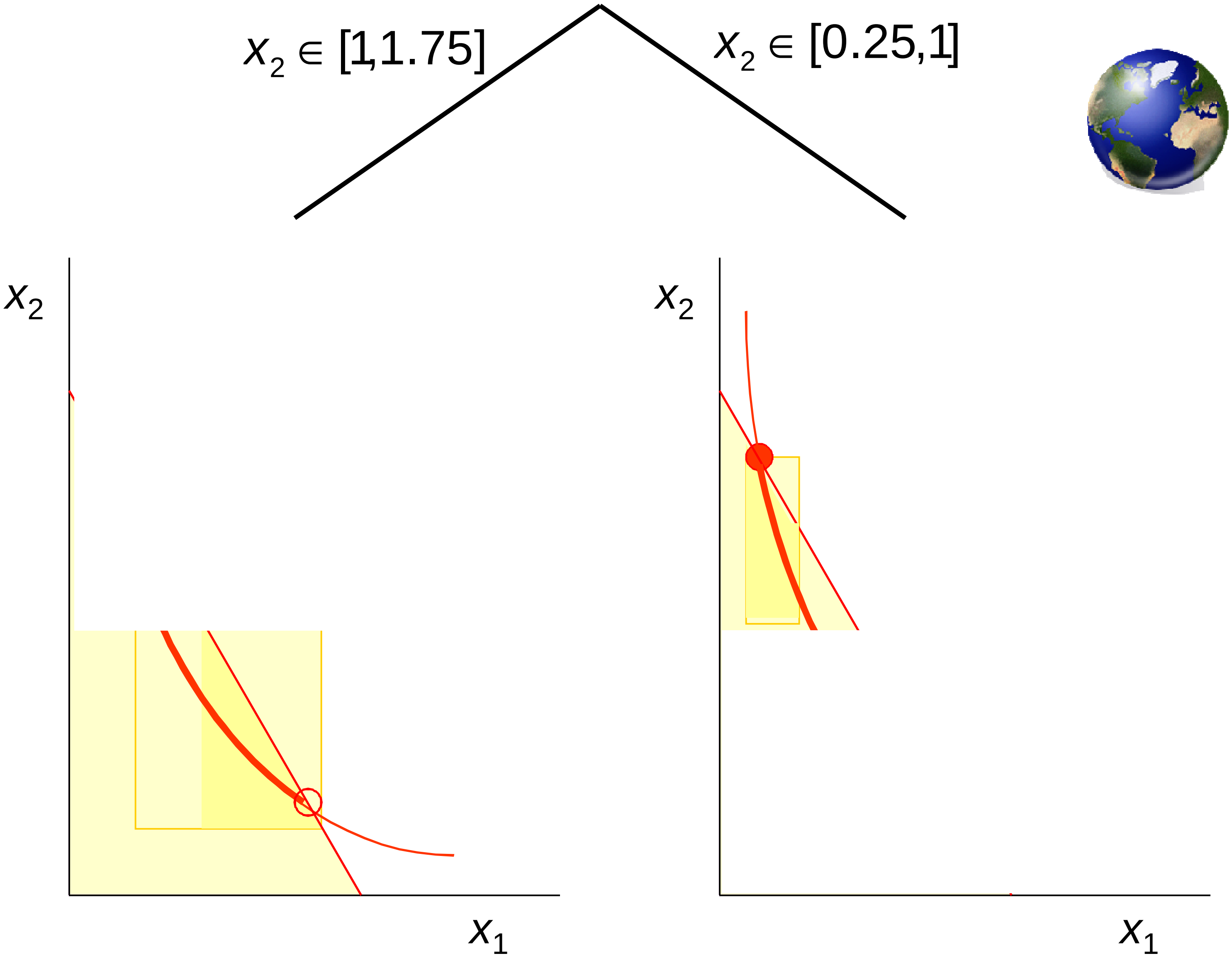
Solve linear relaxation.

Relaxation (function factorization)



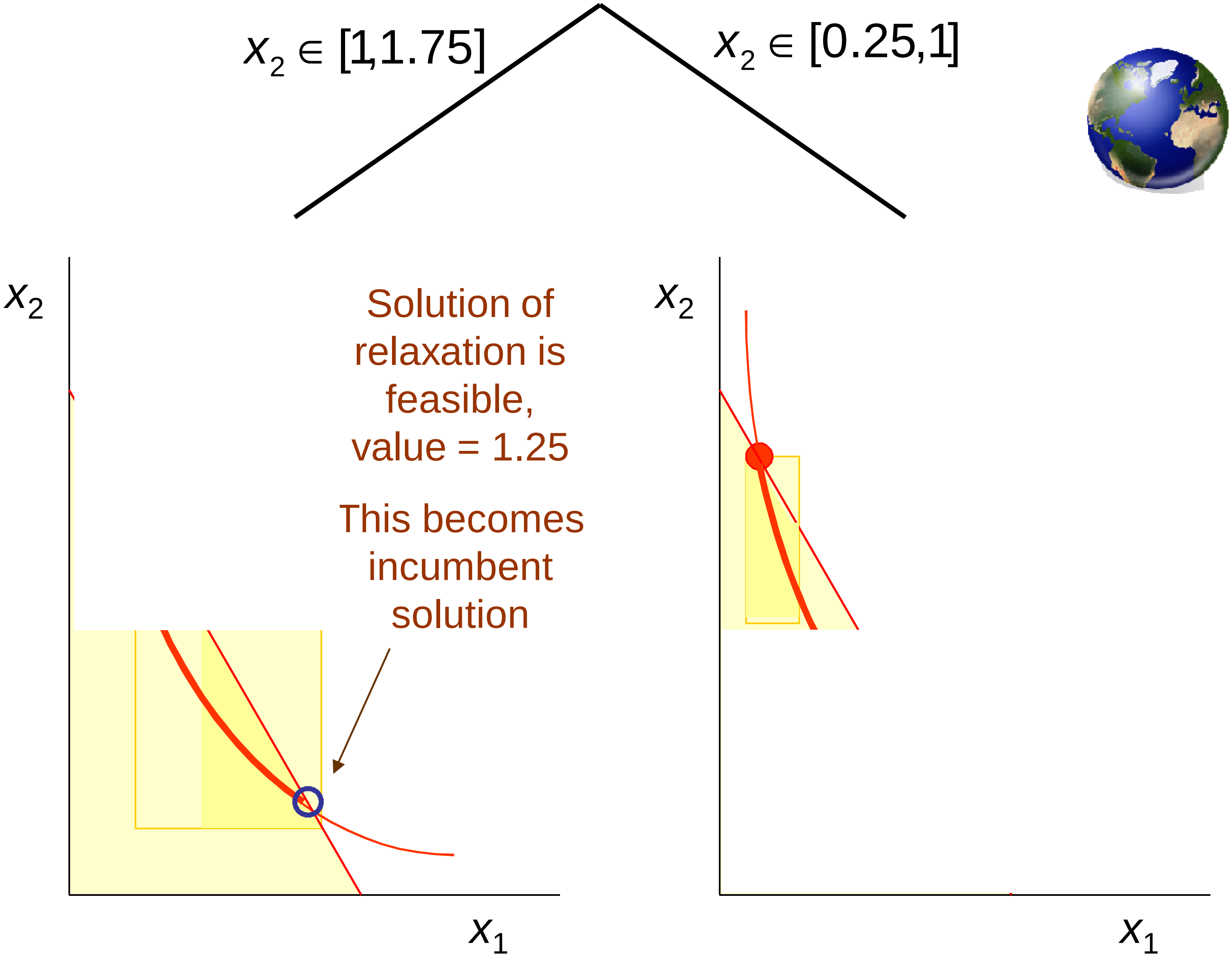
Solve linear relaxation.

Since solution is infeasible,
split an interval and branch.



$x_2 \in [1,1.75]$

$x_2 \in [0.25,1]$

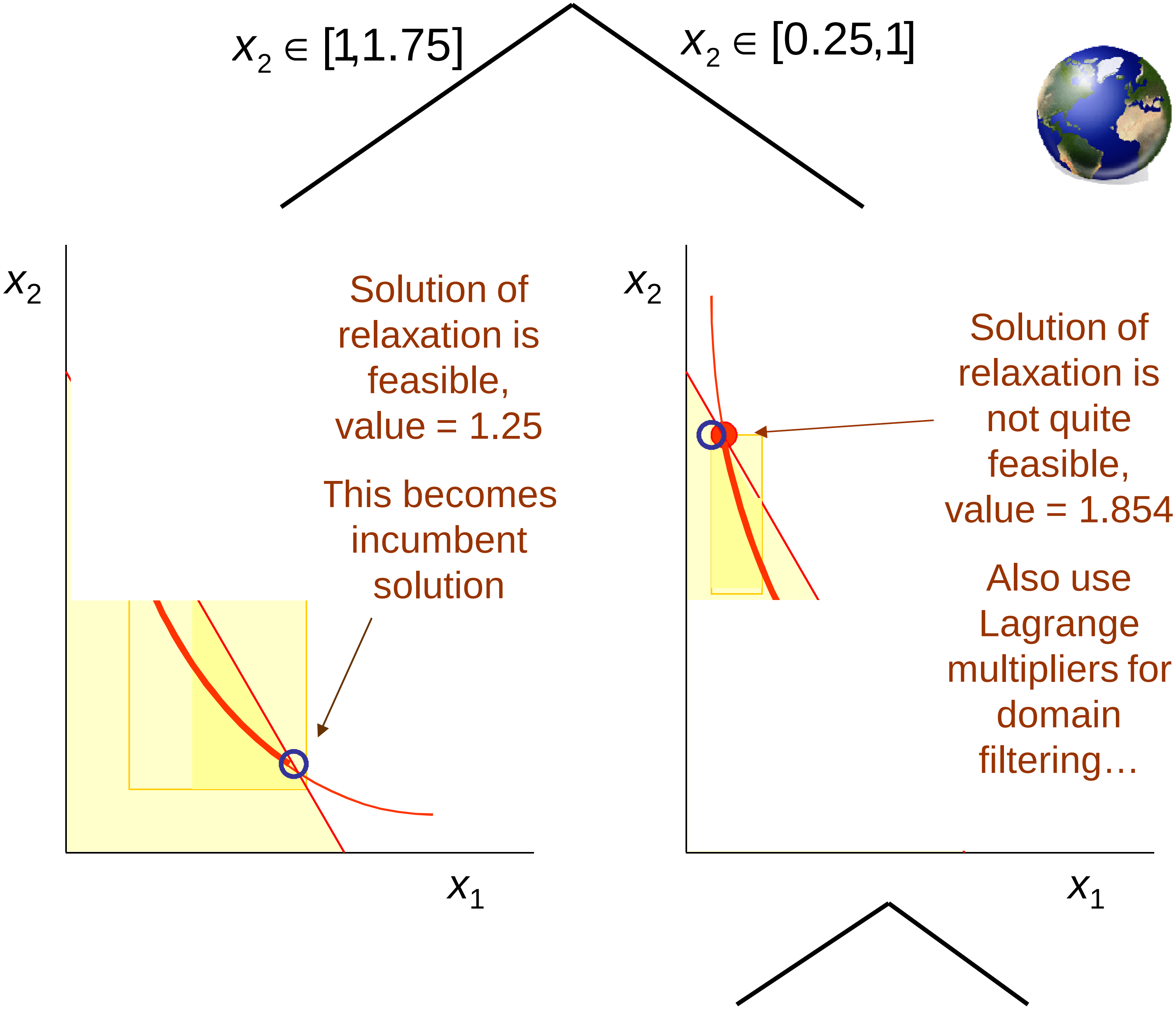


Solution of relaxation is feasible, value = 1.25

This becomes incumbent solution

$x_2 \in [1,1.75]$

$x_2 \in [0.25,1]$



Solution of relaxation is feasible, value = 1.25

This becomes incumbent solution

Solution of relaxation is not quite feasible, value = 1.854

Also use Lagrange multipliers for domain filtering…

Relaxation (function factorization)

$$\min \; x_1 + x_2$$
$$4y = 1$$
$$2x_1 + x_2 \le 2$$
$$\underline{x}_2 x_1 + \underline{x}_1 x_2 - \underline{x}_1 \underline{x}_2 \le y \le \underline{x}_2 x_1 + \bar{x}_1 x_2 - \bar{x}_1 \underline{x}_2$$
$$\bar{x}_2 x_1 + \bar{x}_1 x_2 - \bar{x}_1 \bar{x}_2 \le y \le \bar{x}_2 x_1 + \underline{x}_1 x_2 - \underline{x}_1 \bar{x}_2$$
$$\underline{x}_j \le x_j \le \bar{x}_j, \;\; j = 1,2$$

Associated Lagrange multiplier in solution of relaxation is $\lambda_2 = 1.1$

Relaxation (function factorization)

$$\min \; x_1 + x_2$$

$$4y = 1$$

$$2x_1 + x_2 \le 2$$

Associated Lagrange multiplier in solution of relaxation is $\lambda_2 = 1.1$

$$\underline{x}_2 x_1 + \underline{x}_1 x_2 - \underline{x}_1 \underline{x}_2 \le y \le \underline{x}_2 x_1 + \bar{x}_1 x_2 - \bar{x}_1 \underline{x}_2$$

$$\bar{x}_2 x_1 + \bar{x}_1 x_2 - \bar{x}_1 \bar{x}_2 \le y \le \bar{x}_2 x_1 + \underline{x}_1 x_2 - \underline{x}_1 \bar{x}_2$$

$$\underline{x}_j \le x_j \le \bar{x}_j, \quad j = 1,2$$

This yields a valid inequality for propagation:

$$2x_1 + x_2 \ge 2 - \frac{\boxed{1.854} - \boxed{1.25}}{\boxed{1.1}} = 1.451$$

Value of relaxation

Lagrange multiplier

Value of incumbent solution

CP-based Branch and Price

Basic Idea

Example: Airline Crew Scheduling

Motivation

- **Branch and price** allows solution of integer programming problems with a huge number of variables.
- The problem is solved by a branch-and-bound method. The difference lies in how the LP relaxation is solved.
- Variables are added to the LP relaxation only as needed.
- Variables are **priced** to find which ones should be added.
- **CP** is useful for solving the pricing problem, particularly when constraints are complex.
- **CP-based branch and price** has been successfully applied to airline crew scheduling, transit scheduling, and other transportation-related problems.

Basic Idea

Suppose the LP relaxation of an integer programming problem has a huge number of variables:

$$\begin{aligned} &\min\ cx \\ &Ax = b \\ &x \geq 0 \end{aligned}$$

We will solve a **restricted master problem**, which has a small subset of the variables:

$$\begin{aligned} &\min\ \sum_{j \in J} c_j x_j \\ &\sum_{j \in J} \boxed{A_j} x_j = b \qquad (\lambda) \\ &x_j \geq 0 \end{aligned}$$

Column *j* of *A*

Adding x_k to the problem would improve the solution if x_k has a negative reduced cost:

$$r_k = c_k - \lambda A_k < 0$$

Basic Idea

Adding x_k to the problem would improve the solution if x_k has a negative reduced cost: $r_k = c_k - \lambda A_k < 0$

Computing the reduced cost of x_k is known as **pricing** x_k.

Cost of column *y*

So we solve the pricing problem:

$$\begin{aligned} &\min \ \boxed{c_y} - \lambda y \\ &y \text{ is a column of } A \end{aligned}$$

If the solution y^* satisfies $c_{y^*} - \lambda y^* < 0$, then we can add column y to the restricted master problem.

Basic Idea

The pricing problem

$$\min\ c_y - \lambda y$$
$$y \text{ is a column of } A$$

need not be solved to optimality, so long as we find a column with negative reduced cost.

However, when we can no longer find an improving column, we solved the pricing problem to optimality to make sure we have the optimal solution of the LP.

If we can state constraints that the columns of *A* must satisfy, CP may be a good way to solve the pricing problem.

Airline Crew Scheduling

Assign crew members to flights to minimize cost while covering the flights and observing complex work rules.

Flight data

j	s_j	f_j
1	0	3
2	1	3
3	5	8
4	6	9
5	10	12
6	12	14

Start time

Finish time

A **roster** is the sequence of flights assigned to a single crew member.

The gap between two consecutive flights in a roster must be from 2 to 3 hours.

Total flight time for a roster must be between 6 and 10 hours.

The possible rosters are:

(1,3,5), (1,4,6), (2,3,5), (2,4,6)

Airline Crew Scheduling

There are 2 crew members, and the possible rosters are:

1 2 3 4

(1,3,5), (1,4,6), (2,3,5), (2,4,6)

The LP relaxation of the problem is:

$$\min\ z$$

$$\begin{bmatrix} 10 & 12 & 7 & 13 & 9 & 11 & 6 & 12 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{21} \\ x_{22} \\ x_{23} \\ x_{24} \end{bmatrix} \begin{matrix} = \\ = \\ = \\ \geq \\ \geq \\ \geq \\ \geq \\ \geq \\ \geq \end{matrix} \begin{bmatrix} z \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

$$x_{ik} \geq 0,\ \text{all } i, k$$

Cost of assigning crew member 1 to roster 2

= 1 if we assign crew member 1 to roster 2, = 0 otherwise.

Each crew member is assigned to exactly 1 roster.

Each flight is assigned at least 1 crew member.

Airline Crew Scheduling

There are 2 crew members, and the possible rosters are:

1 2 3 4

(1,3,5), (1,4,6), (2,3,5), (2,4,6)

The LP relaxation of the problem is:

Cost of assigning crew member 1 to roster 2

$$\min z$$

$$\begin{bmatrix} 10 & \boxed{12} & 7 & 13 & 9 & 11 & 6 & 12 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ \boxed{1 \quad 1} & & 0 & 0 & \boxed{1 \quad 1} & & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{11} \\ \boxed{x_{12}} \\ x_{13} \\ x_{14} \\ x_{21} \\ x_{22} \\ x_{23} \\ x_{24} \end{bmatrix} \begin{matrix} = \\ = \\ = \\ \geq \\ \geq \\ \geq \\ \geq \\ \geq \\ \geq \end{matrix} \begin{bmatrix} z \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

$$x_{ik} \geq 0, \text{ all } i, k$$

= 1 if we assign crew member 1 to roster 2, = 0 otherwise.

Each crew member is assigned to exactly 1 roster.

Each flight is assigned at least 1 crew member.

Rosters that cover flight 1.

Airline Crew Scheduling

There are 2 crew members, and the possible rosters are:

1	2	3	4
(1,3,5),	(1,4,6),	(2,3,5),	(2,4,6)

The LP relaxation of the problem is:

Cost c_{12} of assigning crew member 1 to roster 2

$$\min z$$

$$\begin{bmatrix} 10 & 12 & 7 & 13 & 9 & 11 & 6 & 12 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{11} \\ x_{12} \\ x_{13} \\ x_{14} \\ x_{21} \\ x_{22} \\ x_{23} \\ x_{24} \end{bmatrix} \begin{matrix} = \\ = \\ = \\ \geq \\ \geq \\ \geq \\ \geq \\ \geq \\ \geq \end{matrix} \begin{bmatrix} z \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix}$$

$$x_{ik} \geq 0, \ \text{all } i, k$$

= 1 if we assign crew member 1 to roster 2, = 0 otherwise.

Each crew member is assigned to exactly 1 roster.

Each flight is assigned at least 1 crew member.

In a real problem, there can be **millions** of rosters.

Airline Crew Scheduling

We start by solving the problem with a subset of the columns:

$$\min\ z$$

$$\begin{bmatrix}10 & 13 & 9 & 12\\ 1 & 1 & 0 & 0\\ 0 & 0 & 1 & 1\\ 1 & 0 & 1 & 0\\ 0 & 1 & 0 & 1\\ 1 & 0 & 1 & 0\\ 0 & 1 & 0 & 1\\ 1 & 0 & 1 & 0\\ 0 & 1 & 0 & 1\end{bmatrix}\begin{bmatrix}x_{11}\\ x_{14}\\ x_{21}\\ x_{24}\end{bmatrix}\begin{matrix}=\\ =\\ =\\ \geq\\ \geq\\ \geq\\ \geq\\ \geq\\ \geq\end{matrix}\begin{bmatrix}z\\ 1\\ 1\\ 1\\ 1\\ 1\\ 1\\ 1\\ 1\end{bmatrix}$$

$$x_{ik} \geq 0,\ \text{all } i, k$$

Optimal dual solution

$$\boxed{\begin{array}{c}(10)\\ (9)\\ (0)\\ (0)\\ (0)\\ (0)\\ (0)\\ (3)\end{array}}\ \begin{array}{l}u_1\\ u_2\\ v_1\\ v_2\\ v_3\\ v_4\\ v_5\\ v_6\end{array}$$

Airline Crew Scheduling

We start by solving the problem with a subset of the columns:

Dual variables

$$\min\ z$$

$$\begin{bmatrix}10 & 13 & 9 & 12\\ 1 & 1 & 0 & 0\\ 0 & 0 & 1 & 1\\ 1 & 0 & 1 & 0\\ 0 & 1 & 0 & 1\\ 1 & 0 & 1 & 0\\ 0 & 1 & 0 & 1\\ 1 & 0 & 1 & 0\\ 0 & 1 & 0 & 1\end{bmatrix}\begin{bmatrix}x_{11}\\ x_{14}\\ x_{21}\\ x_{24}\end{bmatrix}\begin{matrix}=\\ =\\ =\\ \geq\\ \geq\\ \geq\\ \geq\\ \geq\\ \geq\end{matrix}\begin{bmatrix}z\\ 1\\ 1\\ 1\\ 1\\ 1\\ 1\\ 1\\ 1\end{bmatrix}\qquad\begin{matrix}\\ (10)\\ (9)\\ (0)\\ (0)\\ (0)\\ (0)\\ (0)\\ (3)\end{matrix}\ \begin{matrix}\\ \boxed{\begin{matrix}u_1\\ u_2\\ v_1\\ v_2\\ v_3\\ v_4\\ v_5\\ v_6\end{matrix}}\end{matrix}$$

$$x_{ik} \geq 0,\ \text{all } i, k$$

Airline Crew Scheduling

We start by solving the problem with a subset of the columns:

Dual variables

$$
\begin{array}{l}
\min\ z \\
\begin{bmatrix}
10 & 13 & 9 & 12 \\
1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 \\
1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 \\
0 & 1 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x_{11} \\ x_{14} \\ x_{21} \\ x_{24}
\end{bmatrix}
\begin{array}{c}
= \\ = \\ = \\ \geq \\ \geq \\ \geq \\ \geq \\ \geq \\ \geq
\end{array}
\begin{bmatrix}
z \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1
\end{bmatrix}
\quad
\begin{array}{cc}
 & \\
(10) & u_1 \\
(9) & u_2 \\
(0) & v_1 \\
(0) & v_2 \\
(0) & v_3 \\
(0) & v_4 \\
(0) & v_5 \\
(3) & v_6
\end{array} \\
x_{ik} \geq 0,\ \text{all } i,k
\end{array}
$$

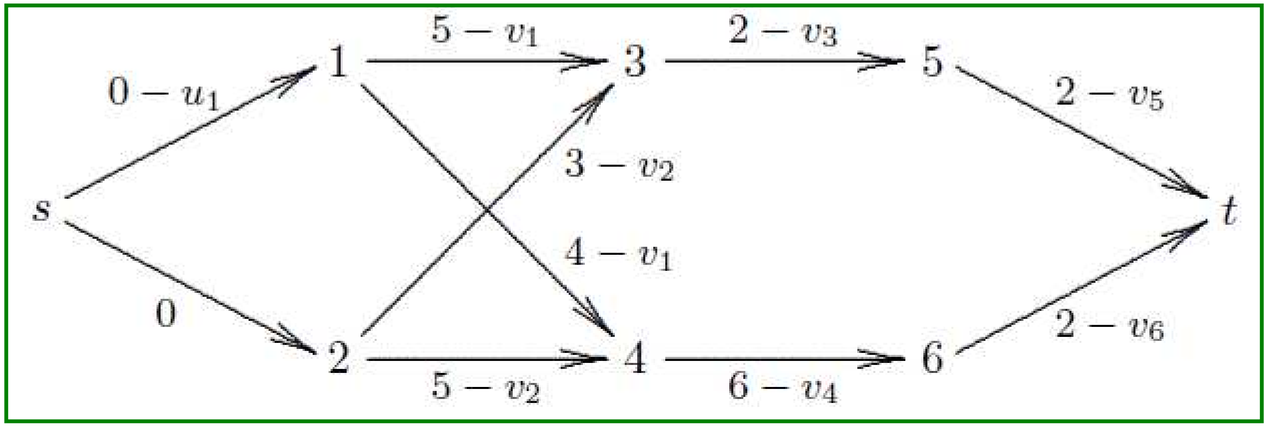
The reduced cost of an excluded roster *k* for crew member *i* is

$$c_{ik} - u_i - \sum_{j \text{ in roster k}} v_j$$

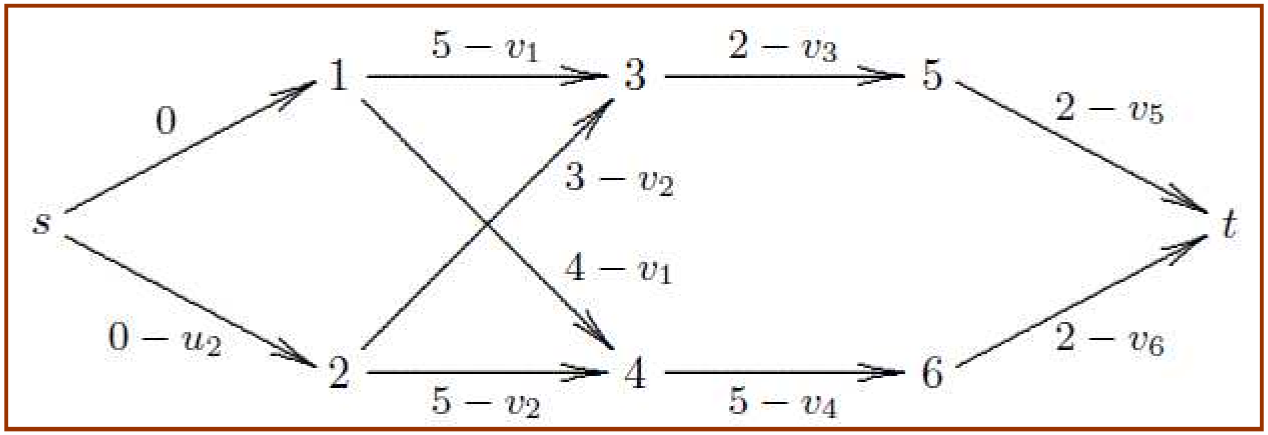
We will formulate the pricing problem as a shortest path problem.

Pricing problem

Crew member 1



Crew member 2

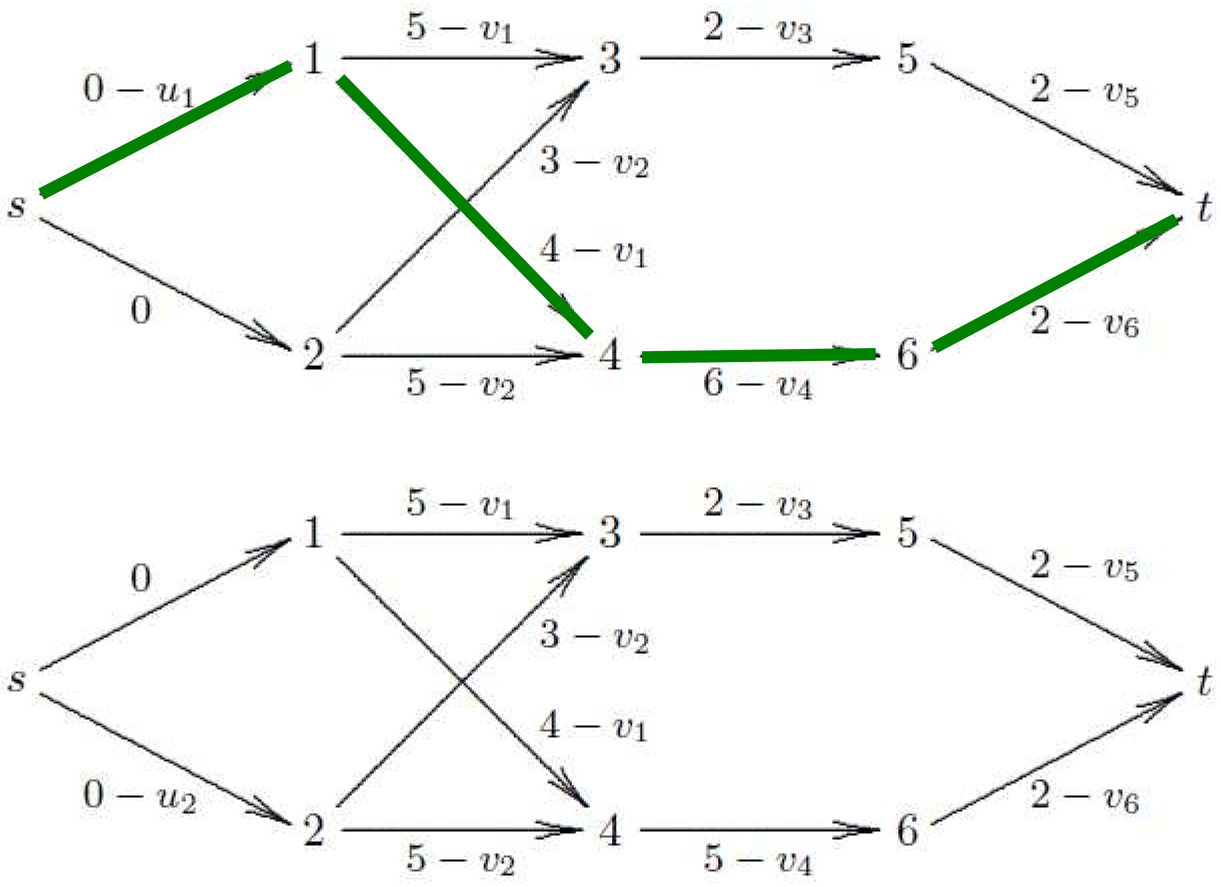


Pricing problem

Each s-t path corresponds to a roster, provided the flight time is within bounds.

Crew member 1

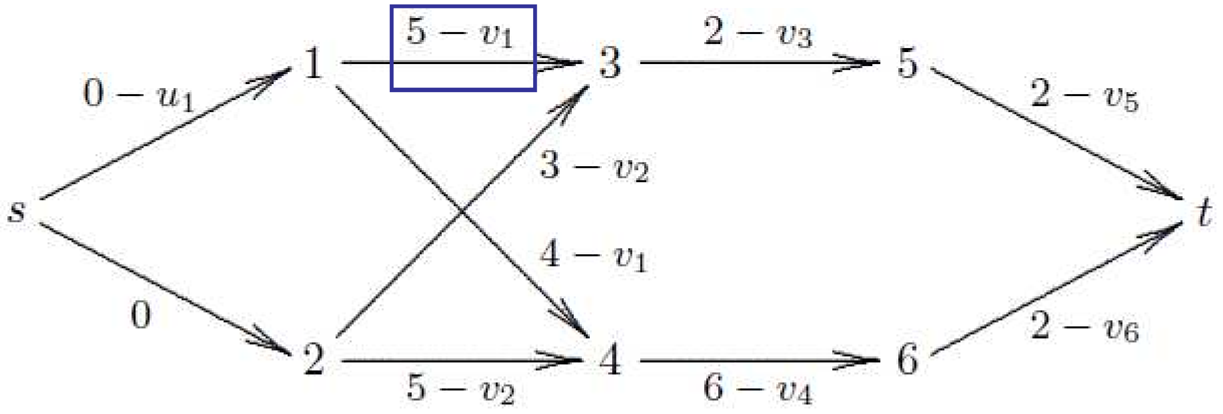
Crew member 2



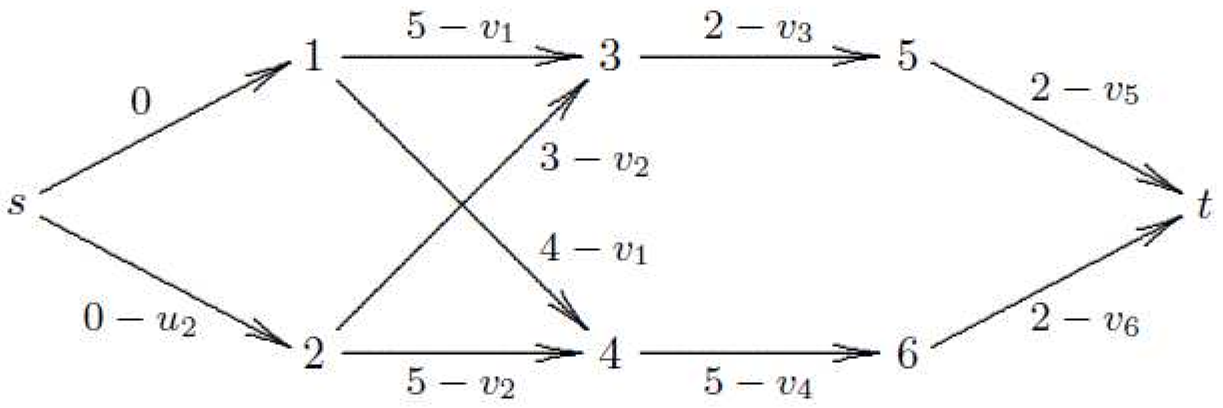
Pricing problem

Cost of flight 3 if it immediately follows flight 1, offset by dual multiplier for flight 1

Crew member 1



Crew member 2



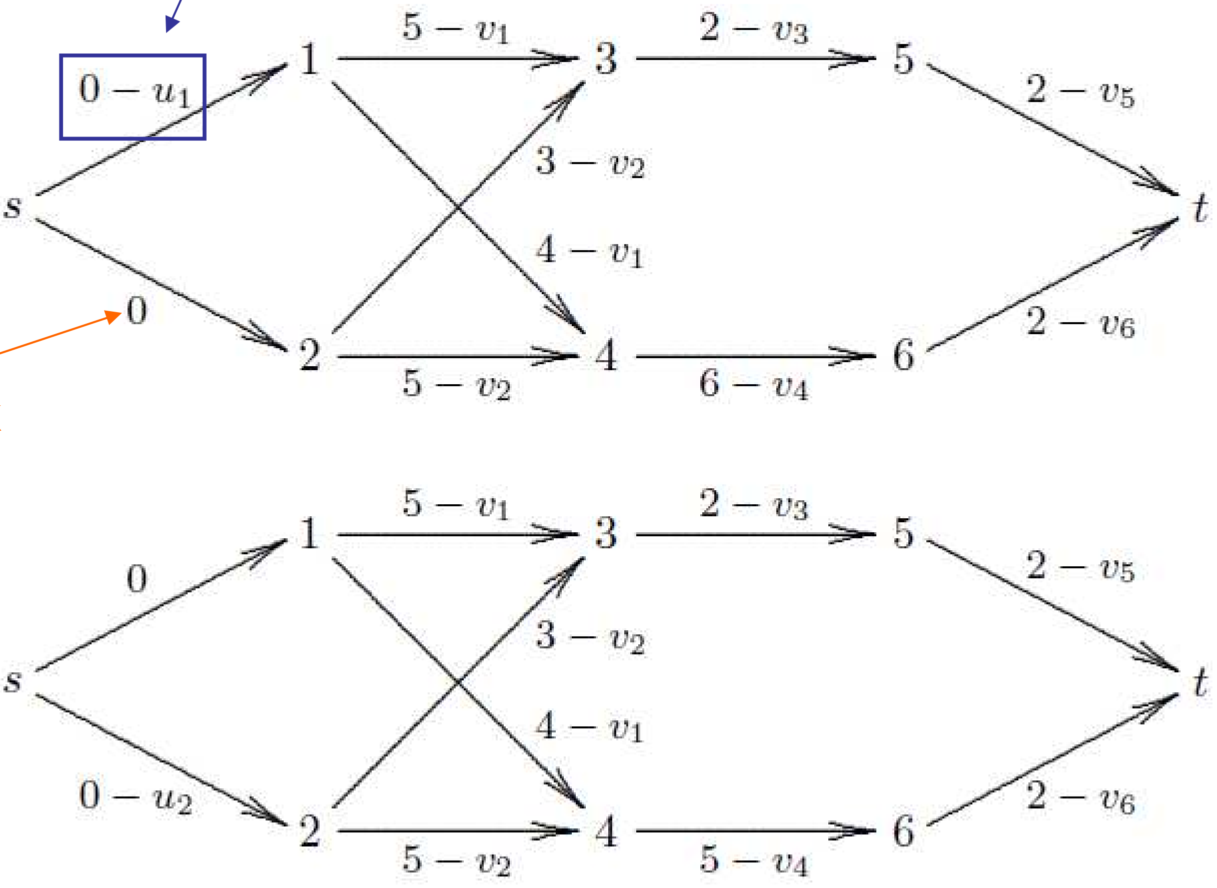
Pricing problem

Cost of transferring from home to flight 1, offset by dual multiplier for crew member 1

Crew member 1

Dual multiplier omitted to break symmetry

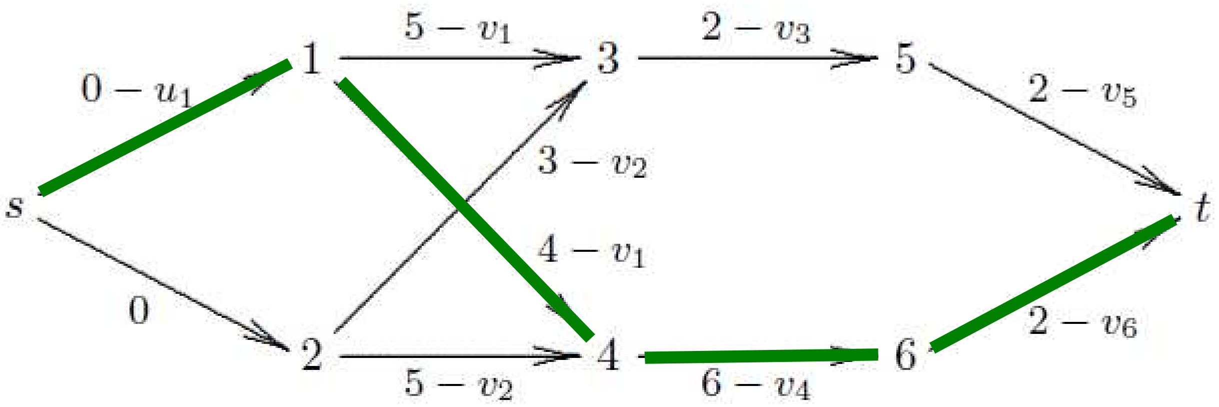
Crew member 2



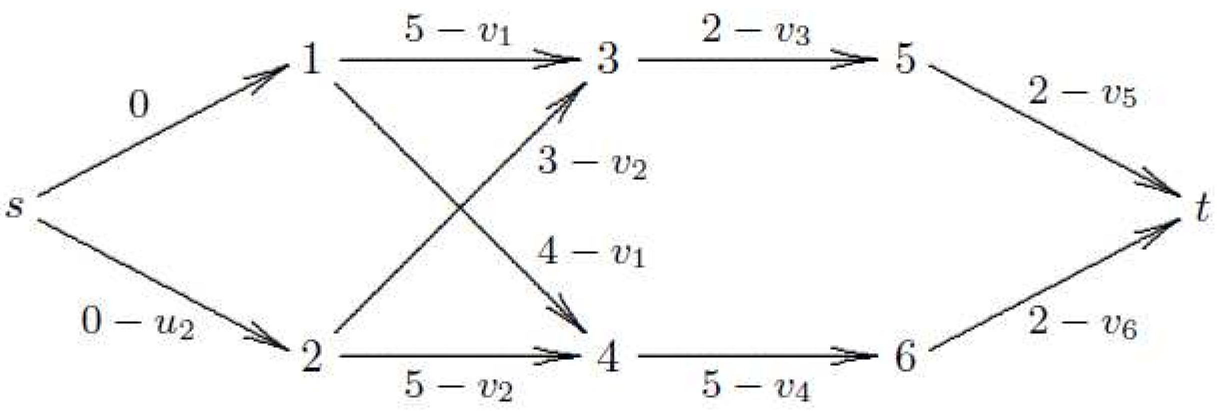
Pricing problem

Length of a path is reduced cost of the corresponding roster.

Crew member 1



Crew member 2

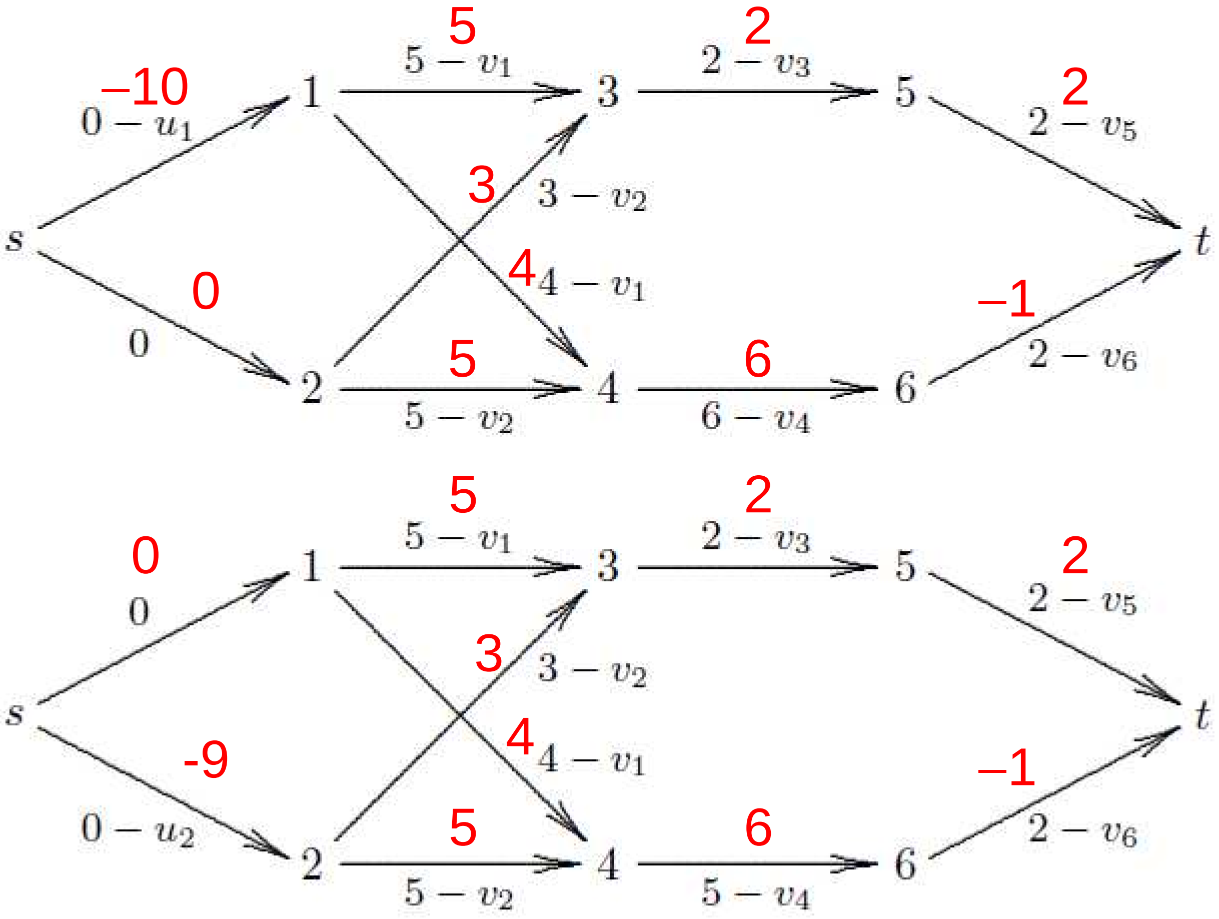


Pricing problem

Arc lengths using dual solution of LP relaxation

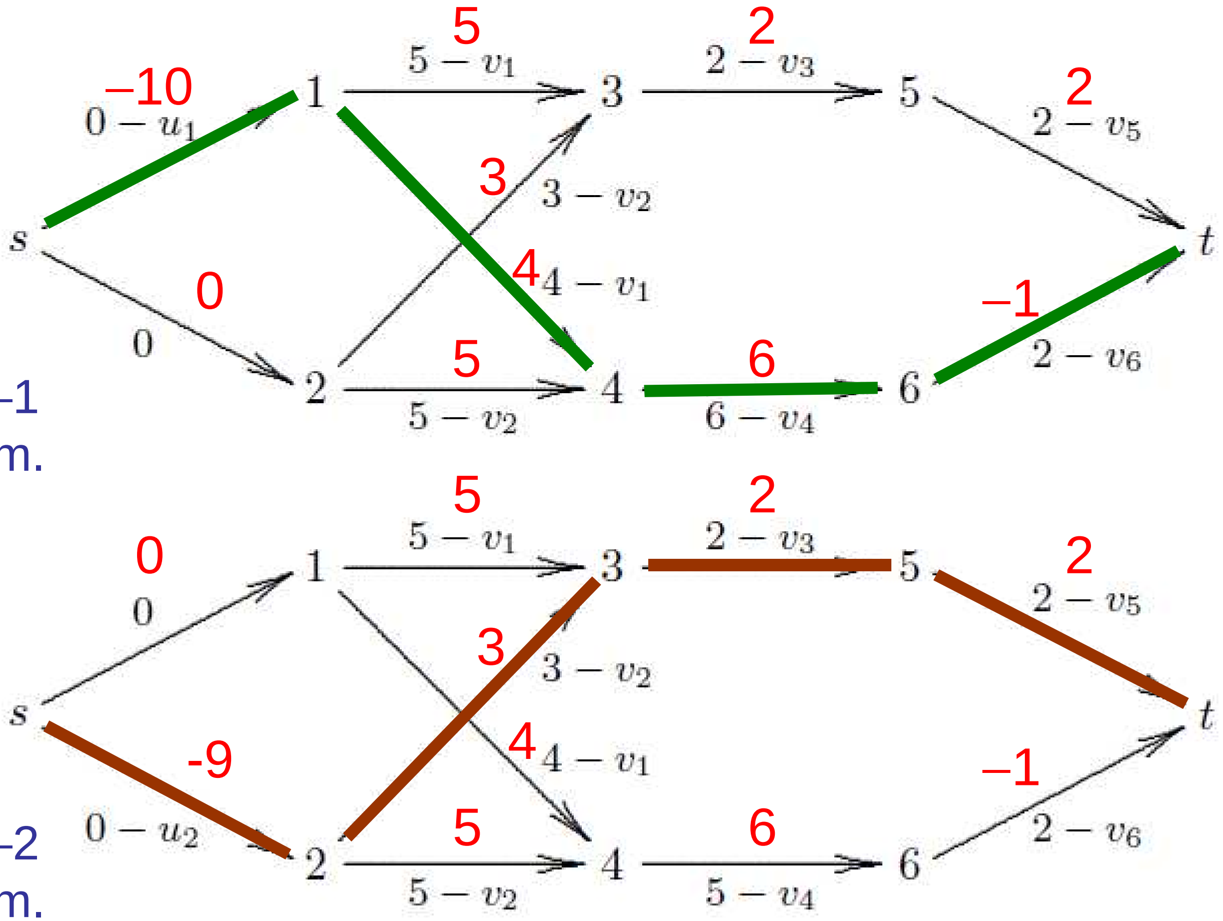
Crew member 1

Crew member 2



Pricing problem

Solution of shortest path problems



Crew member 1

Reduced cost = –1
Add x_{12} to problem.

Crew member 2

Reduced cost = –2
Add x_{23} to problem.

After x_{12} and x_{23} are added to the problem, no remaining variable has negative reduced cost.

Pricing problem

The shortest path problem cannot be solved by traditional shortest path algorithms, due to the bounds on total duration of flights.

It **can** be solved by CP:

Set of flights assigned to crew member *i*

Path length

Graph

Path global constraint →

$$\text{Path}(\boxed{X_i},\boxed{z_i},\boxed{G}),\ \text{all flights } i$$

Setsum global constraint →

$$T_{\min} \le \sum_{j\in X_i} \left(\boxed{f_j - s_j}\right) \le T_{\max}$$

$$X_i \subset \{\text{flights}\},\ z_i < 0,\ \text{all } i$$

Duration of flight *j*

CP-based Benders Decomposition

Benders Decomposition in the Abstract

Classical Benders Decomposition

Example: Machine Scheduling

Motivation

- **Benders decomposition** allows us to apply CP and OR to different parts of the problem.
- It searches over values of certain variables that, when fixed, result in a much simpler **subproblem**.
- The search learns from past experience by accumulating **Benders cuts** (a form of nogood).
- The technique can be **generalized** far beyond the original OR conception.
- Generalized Benders methods have resulted in the **greatest speedups** achieved by combining CP and OR.

Benders Decomposition in the Abstract

Benders decomposition can be applied to problems of the form

$$\begin{aligned}&\min\ f(x,y)\\&S(x,y)\\&x\in D_x,\ \ y\in D_y\end{aligned}$$

When x is fixed to some value, the resulting **subproblem** is much easier:

$$\begin{aligned}&\min\ f(\bar{x},y)\\&S(\bar{x},y)\\&y\in D_y\end{aligned}$$

…perhaps because it decouples into smaller problems.

For example, suppose *x* assigns jobs to machines, and *y* schedules the jobs on the machines.

When *x* is fixed, the problem decouples into a separate scheduling subproblem for each machine.

Benders Decomposition

We will search over assignments to x. This is the **master problem**.

In iteration k we assume $x = x^k$ and solve the subproblem

$$\begin{array}{l} \min\ f(x^k, y) \\ S(x^k, y) \\ y \in D_y \end{array}$$

and get optimal value v_k

We generate a **Benders cut** (a type of nogood) $\boxed{v} \geq B_{k+1}(x)$

Cost in the original problem

that satisfies $B_{k+1}(x^k) = v_k$.

The Benders cut says that if we set $x = x^k$ again, the resulting cost v will be at least v_k. To do better than v_k, we must try something else.

It also says that any other x will result in a cost of at least $B_{k+1}(x)$, perhaps due to some similarity between x and x^k.

Benders Decomposition

We will search over assignments to *x*. This is the **master problem**.

In iteration *k* we assume $x = x^k$ and solve the subproblem

$$\begin{aligned}&\min\ f(x^k, y)\\ &S(x^k, y)\\ &y \in D_y\end{aligned}$$

and get optimal value v_k

We generate a **Benders cut** (a type of nogood) $\boxed{v} \geq B_{k+1}(x)$

that satisfies $B_{k+1}(x) = v_k$.

Cost in the original problem

We add the Benders cut to the master problem, which becomes

$$\begin{aligned}&\min\ v\\ &v \geq B_i(x),\ i = 1,\ldots,k+1 \longleftarrow\\ &x \in D_x\end{aligned}$$

Benders cuts generated so far

Benders Decomposition

We now solve the master problem

$$\begin{aligned} &\min\ v \\ &v \geq B_i(x),\ i = 1,\ldots,k+1 \\ &x \in D_x \end{aligned}$$

to get the next trial value x^{k+1}.

The master problem is a relaxation of the original problem, and its optimal value is a **lower bound** on the optimal value of the original problem.

The subproblem is a restriction, and its optimal value is an **upper bound**.

The process continues until the bounds meet.

The Benders cuts partially define the **projection** of the feasible set onto *x*. We hope not too many cuts are needed to find the optimum.

Classical Benders Decomposition

The classical method applies to problems of the form

$$\begin{aligned} &\min\ f(x) + cy \\ &g(x) + Ay \ge b \\ &x \in D_x,\ y \ge 0 \end{aligned}$$

and the subproblem is an LP

$$\begin{aligned} &\min\ f(x^k) + cy \\ &Ay \ge b - g(x^k) \quad (\lambda) \\ &y \ge 0 \end{aligned}$$

whose dual is

$$\begin{aligned} &\max\ f(x^k) + \lambda\big(b - g(x^k)\big) \\ &\lambda A \le c \\ &\lambda \ge 0 \end{aligned}$$

Let λ^k solve the dual.

By strong duality, $B_{k+1}(x) = f(x) + \lambda^k(b - g(x))$ is the tightest lower bound on the optimal value v of the original problem when $x = x^k$.

Even for other values of x, λ^k **remains feasible in the dual**. So by weak duality, $B_{k+1}(x)$ remains a lower bound on v.

Classical Benders

So the master problem

$$
\begin{aligned}
&\min\ v \\
&v \geq B_i(x),\ i = 1,\ldots,k+1 \\
&x \in D_x
\end{aligned}
$$

becomes

$$
\begin{aligned}
&\min\ v \\
&v \geq f(x) + \lambda^i (b - g(x)),\ i = 1,\ldots,k+1 \\
&x \in D_x
\end{aligned}
$$

In most applications the master problem is

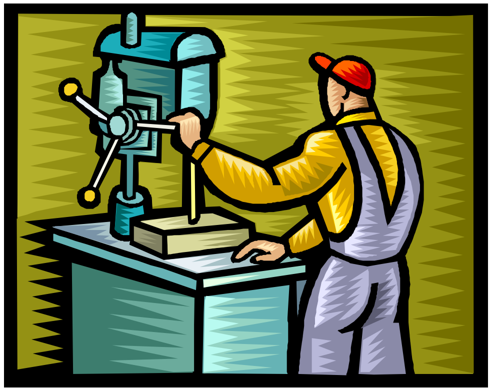
- an MILP
- a nonlinear programming problem (NLP), or
- a mixed integer/nonlinear programming problem (MINLP).

Example: Machine Scheduling

- Assign 5 jobs to 2 machines (A and B), and schedule the machines assigned to each machine within time windows.

- The objective is to minimize **makespan**.

Time lapse between start of first job and end of last job.

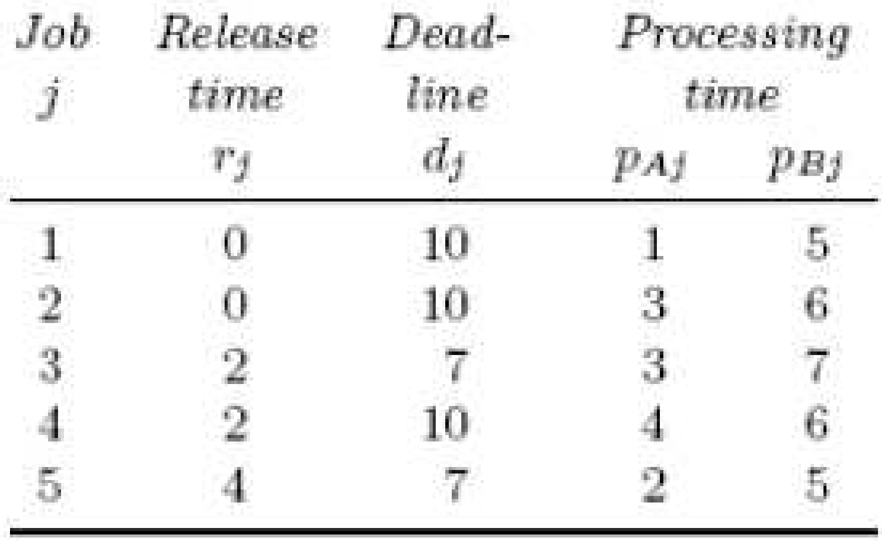


- Assign the jobs in the **master problem**, to be solved by **MILP**.

- Schedule the jobs in the **subproblem**, to be solved by **CP**.

Machine Scheduling

Job Data



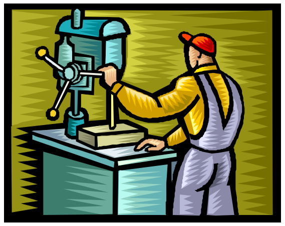
Job j	Release time r_j	Dead- line d_j	Processing time p_{Aj}	 p_{Bj}
1	0	10	1	5
2	0	10	3	6
3	2	7	3	7
4	2	10	4	6
5	4	7	2	5

Machine A

Machine B

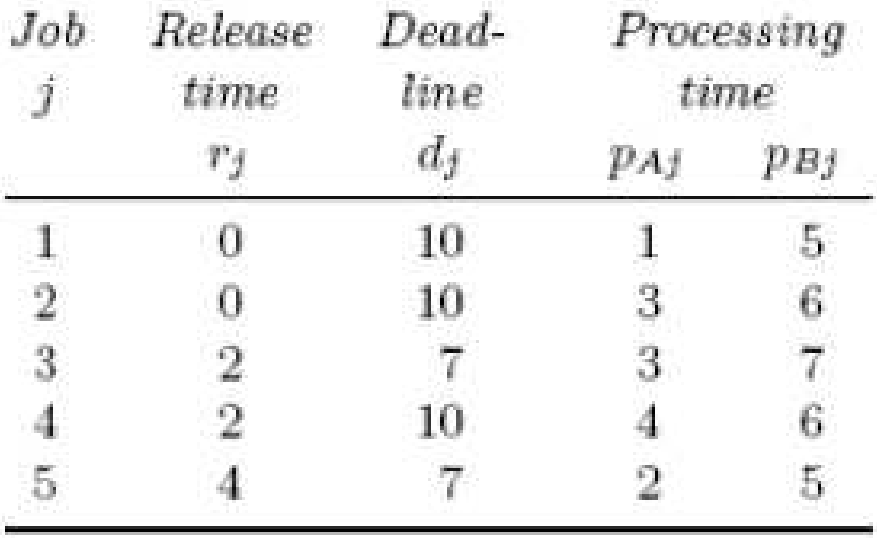
Once jobs are assigned, we can minimize overall makespan by minimizing makespan on each machine individually.

So the subproblem decouples.



Machine Scheduling

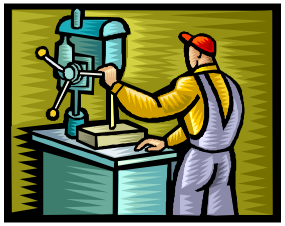
Job Data



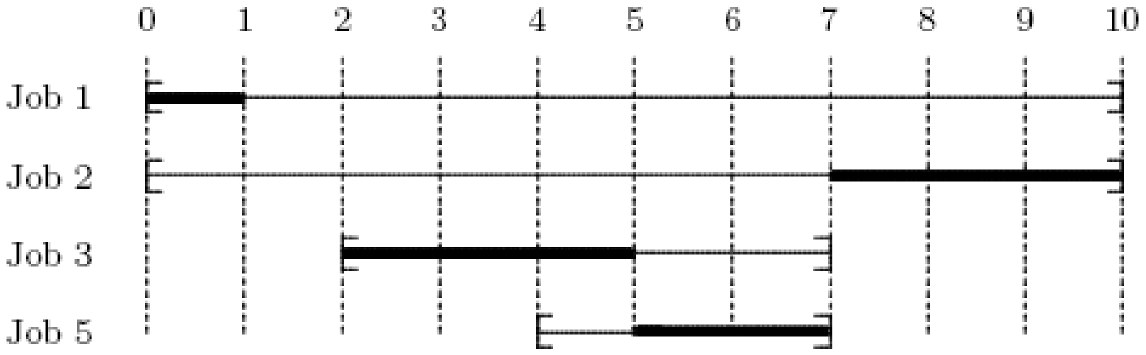
Job j	Release time r_j	Dead- line d_j	Processing time p_{Aj}	 p_{Bj}
1	0	10	1	5
2	0	10	3	6
3	2	7	3	7
4	2	10	4	6
5	4	7	2	5

Once jobs are assigned, we can minimize overall makespan by minimizing makespan on each machine individually.

So the subproblem decouples.



Minimum makespan schedule for jobs 1, 2, 3, 5 on machine A



Machine Scheduling

The problem is

Start time of job *j*

$$\min \; M$$

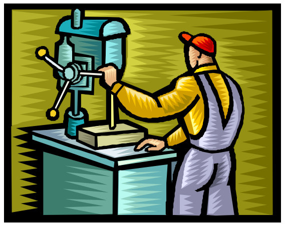
$$M \ge \boxed{s_j} + p_{x_j j}, \text{ all } j$$

Time windows

$$r_j \le s_j \le d_j - p_{x_j j}, \text{ all } j$$

Jobs cannot overlap

$$\text{disjunctive}\big((s_j \,|\, x_j = i),(p_{ij} \,|\, x_j = i)\big), \text{ all } i$$



Machine Scheduling

The problem is

Start time of job *j*

$$\min\ M$$

$$M \geq \boxed{s_j} + p_{x_j j},\ \text{all } j$$

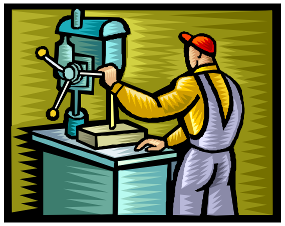
Time windows

$$r_j \leq s_j \leq d_j - p_{x_j j},\ \text{all } j$$

Jobs cannot overlap

$$\text{disjunctive}\big((s_j \mid x_j = i),(p_{ij} \mid x_j = i)\big),\ \text{all } i$$

For a fixed assignment $\bar{x}$ the subproblem on each machine *i* is



$$\min\ M$$

$$M \geq s_j + p_{\bar{x}_j j},\ \text{all } j \text{ with } \bar{x}_j = i$$

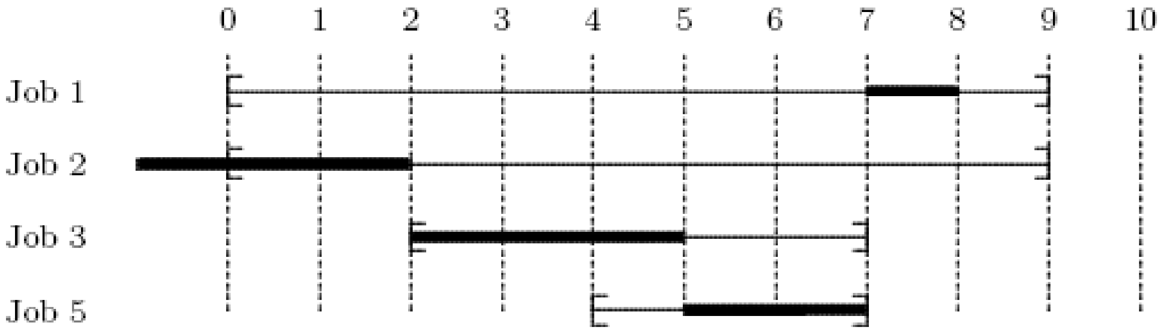
$$r_j \leq s_j \leq d_j - p_{\bar{x}_j j},\ \text{all } j \text{ with } \bar{x}_j = i$$

$$\text{disjunctive}\big((s_j \mid \bar{x}_j = i),(p_{ij} \mid \bar{x}_j = i)\big)$$

Benders cuts

Suppose we assign jobs 1,2,3,5 to machine A in iteration *k*.

We can prove that 10 is the optimal makespan by proving that the schedule is infeasible with makespan 9.



Edge finding derives infeasibility by reasoning only with jobs 2,3,5. So these jobs alone create a minimum makespan of 10.

So we have a Benders cut

$$v \geq B_{k+1}(x) = \begin{cases} 10 & \text{if } x_2 = x_3 = x_4 = A \\ 0 & \text{otherwise} \end{cases}$$

Benders cuts

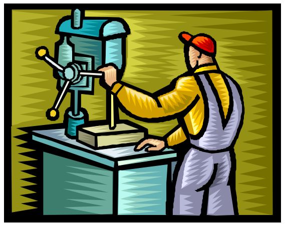
We want the master problem to be an MILP, which is good for assignment problems.

So we write the Benders cut

$$v \geq B_{k+1}(x) = \begin{cases} 10 & \text{if } x_2 = x_3 = x_4 = A \\ 0 & \text{otherwise} \end{cases}$$

Using 0-1 variables:
$$v \geq 10\left(x_{A2} + x_{A3} + \boxed{x_{A5}} - 2\right)$$
$$v \geq 0$$

= 1 if job 5 is assigned to machine A



Master problem

The master problem is an MILP:

$$\min\ v$$

$$\sum_{j=1}^{5} p_{Aj} x_{Aj} \le 10, \text{ etc.}$$

$$\sum_{j=1}^{5} p_{Bj} x_{Bj} \le 10, \text{ etc.}$$

Constraints derived from time windows

$$v \ge \sum_{j=1}^{5} p_{ij} x_{ij},\quad v \ge 2 + \sum_{j=3}^{5} p_{ij} x_{ij}, \text{ etc.},\quad i = A,B$$

Constraints derived from release times

$$\text{v} \ge 10(x_{A2} + x_{A3} + x_{A5} - 2)$$

Benders cut from machine A

$$v \ge 8x_{B4}$$

Benders cut from machine B

$$x_{ij} \in \{0,1\}$$

Stronger Benders cuts

If all release times are the same, we can strengthen the Benders cuts.

We are now using the cut

$$v \geq M_{ik} \left(\sum_{j \in J_{ik}} x_{ij} - |J_{ik}| + 1 \right)$$

Min makespan on machine *i* in iteration *k*

Set of jobs assigned to machine *i* in iteration *k*

A stronger cut provides a useful bound even if only some of the jobs in J_{ik} are assigned to machine *i*:

$$v \geq M_{ik} - \sum_{j \in J_{ik}} (1 - x_{ij}) p_{ij}$$

These results can be generalized to cumulative scheduling.

Cumulative scheduling in subproblem

Subproblem for each facility *i*, given an assignment *x* from master

$$
\begin{aligned}
&\min\ M \\
&M \geq t_j + p_{x_j j},\ \text{all } j \\
&r_j \leq t_j \leq d_j - p_{x_j j},\ \text{all } j \\
&\text{cumulative}\Big((t_j \,|\, x_j = i), (p_{ij} \,|\, x_j = i), (c_{ij} \,|\, x_j = i) \Big)
\end{aligned}
$$

Sample Benders cut (all release times the same):

Deadline for job *j*

$$
M \geq M_{ik} \left(\sum_{j \in J_{ik}} p_{ij}(1 - y_{ij}) + \max_{j \in J_{ik}} \{d_j\} - \min_{j \in J_{ik}} \{d_j\} \right)
$$

Min makespan on facility *i* in iteration *k*

=1 if job *j* assigned to facility *i* ($x_j = i$)

Set of jobs assigned to facility *i* in iteration *k*

Some Very Recent Work

Benders for scheduling

Cutting planes from CP model

BDDs as constraint store

BDDs for relaxation bounds

Recent work – Benders for Scheduling

Joint work with Elvin Coban.

Apply logic-based Benders to single-facility scheduling with long time horizons and many jobs.

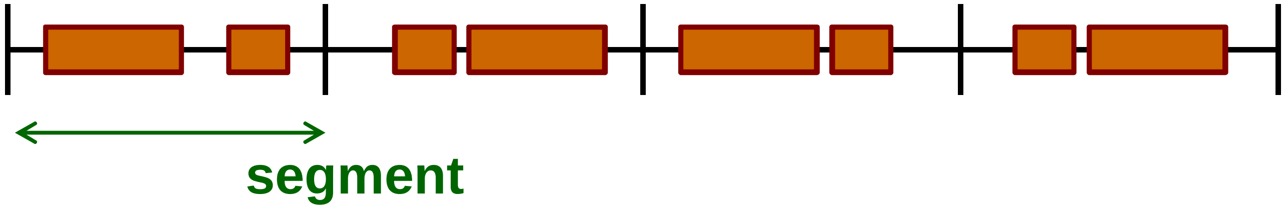
Decompose the problem by assigning jobs to segments of time horizon.

Segmented problem – Jobs cannot cross segment boundaries (e.g., weekends).

Unsegmented problem – Jobs can cross segment boundaries.

Segmented problem

- Benders approach is very similar to that for the planning and scheduling problem.
  - Assign jobs to time segments rather than processors.
  - Benders cuts are the same.



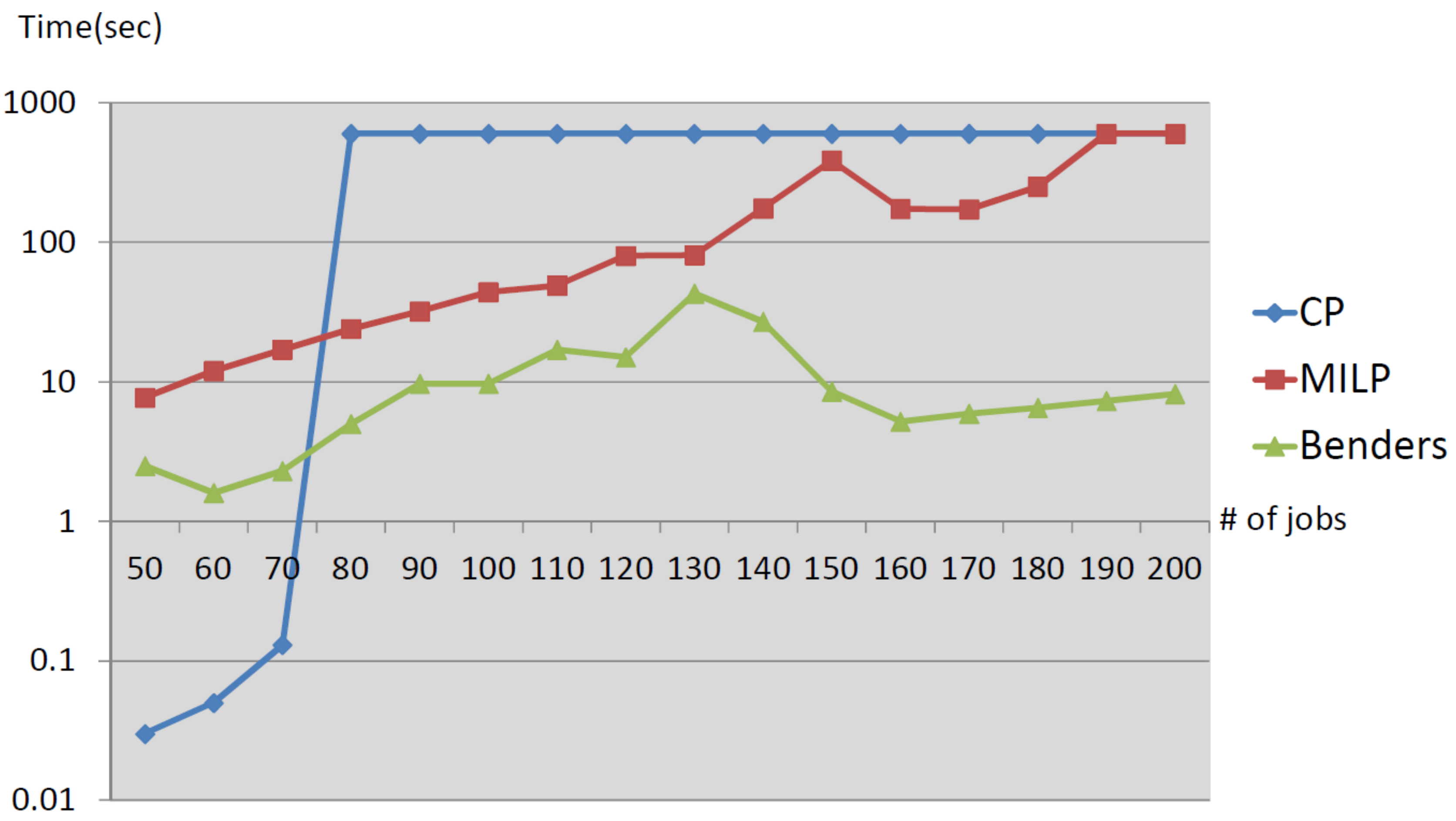
Jobs do not overlap segment boundaries

Segmented problem

- Experiments use most recent versions of CP and IP solvers.
    - IBM OPL Studio 6.1
    - CPLEX 12

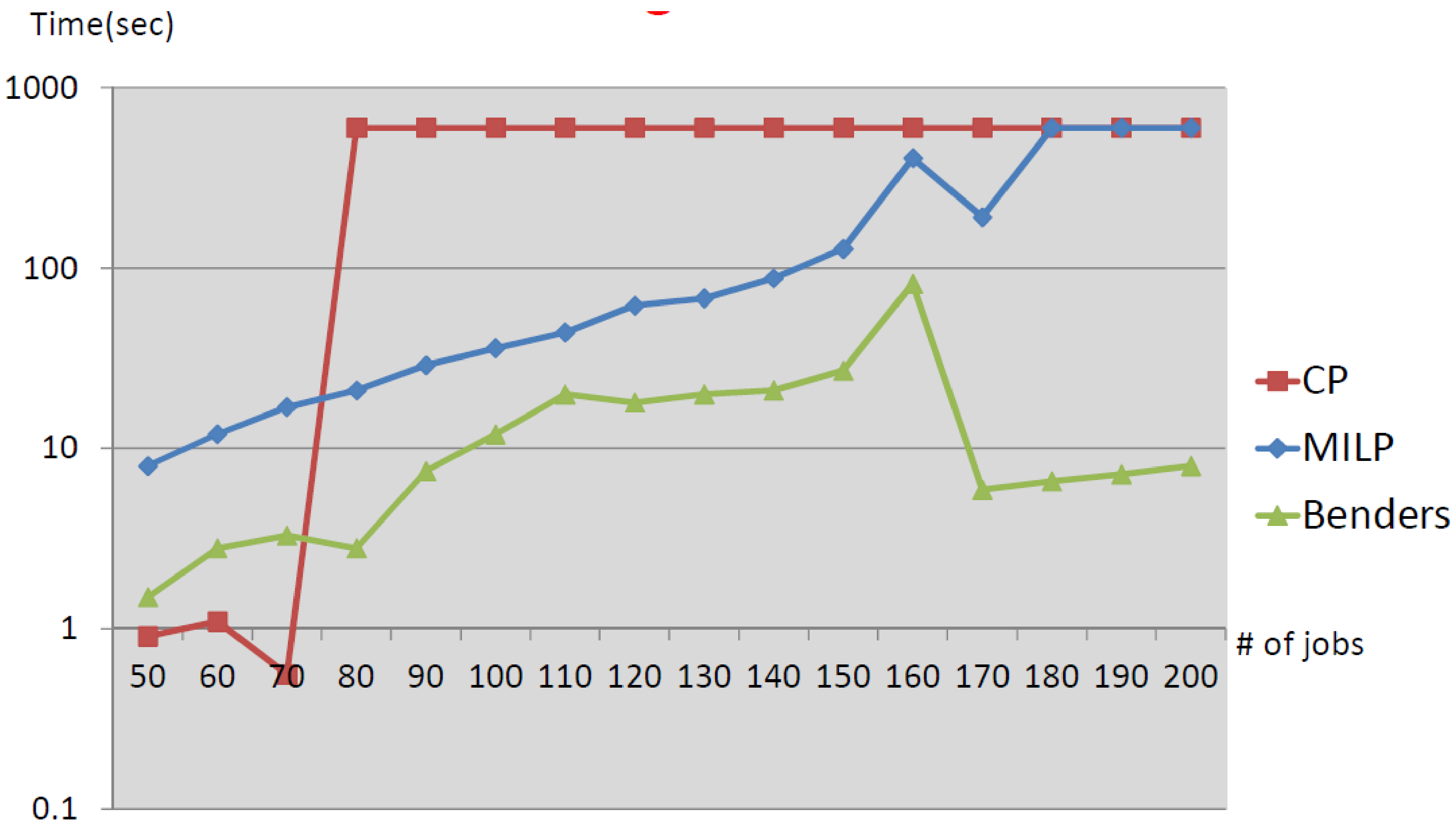
Segmented problem computational results

Feasibility – Wide time windows (individual instances)



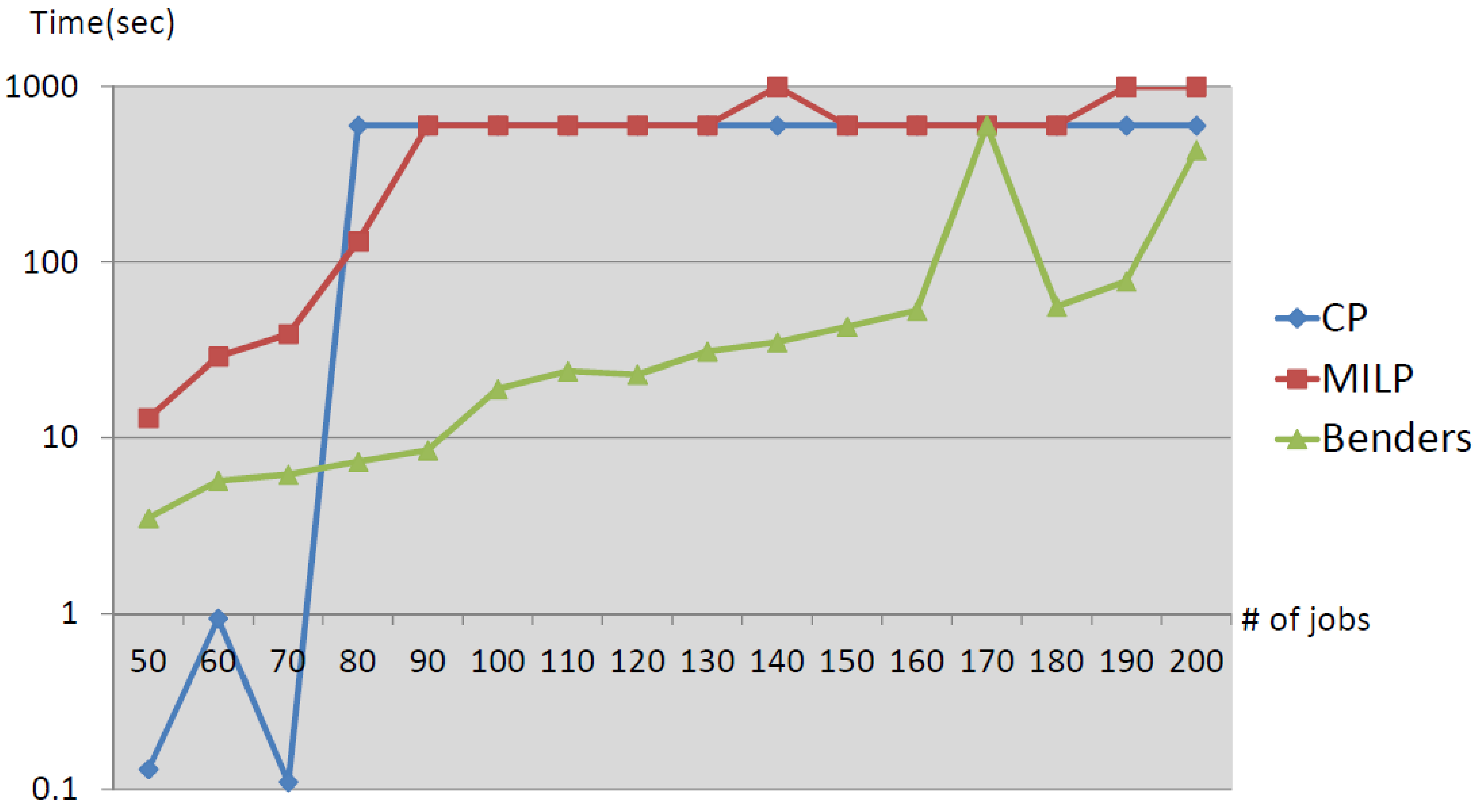
Segmented problem computational results

Feasibility – Tight time windows (individual instances)



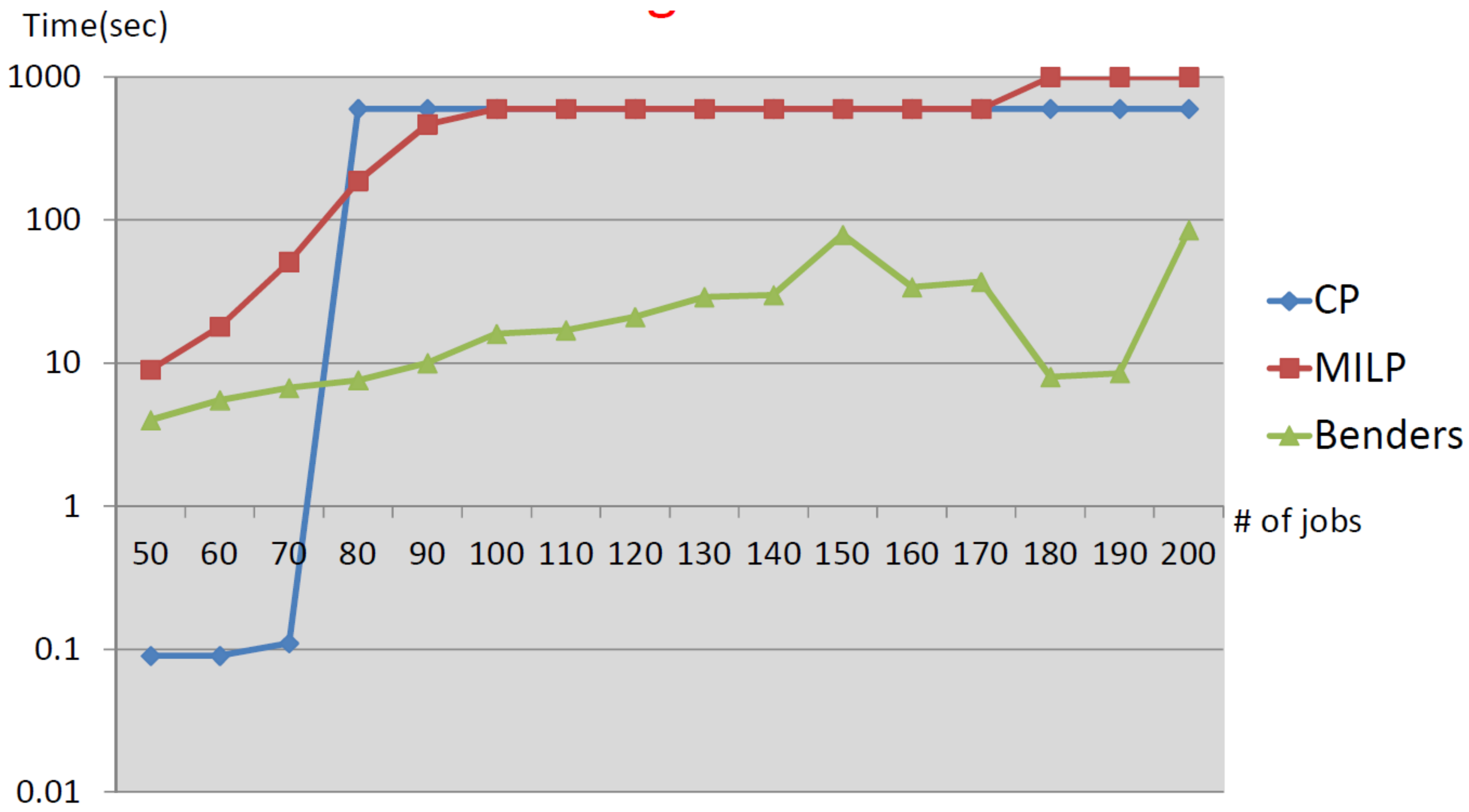
Segmented problem computational results

Min makespan – Wide time windows (individual instances)



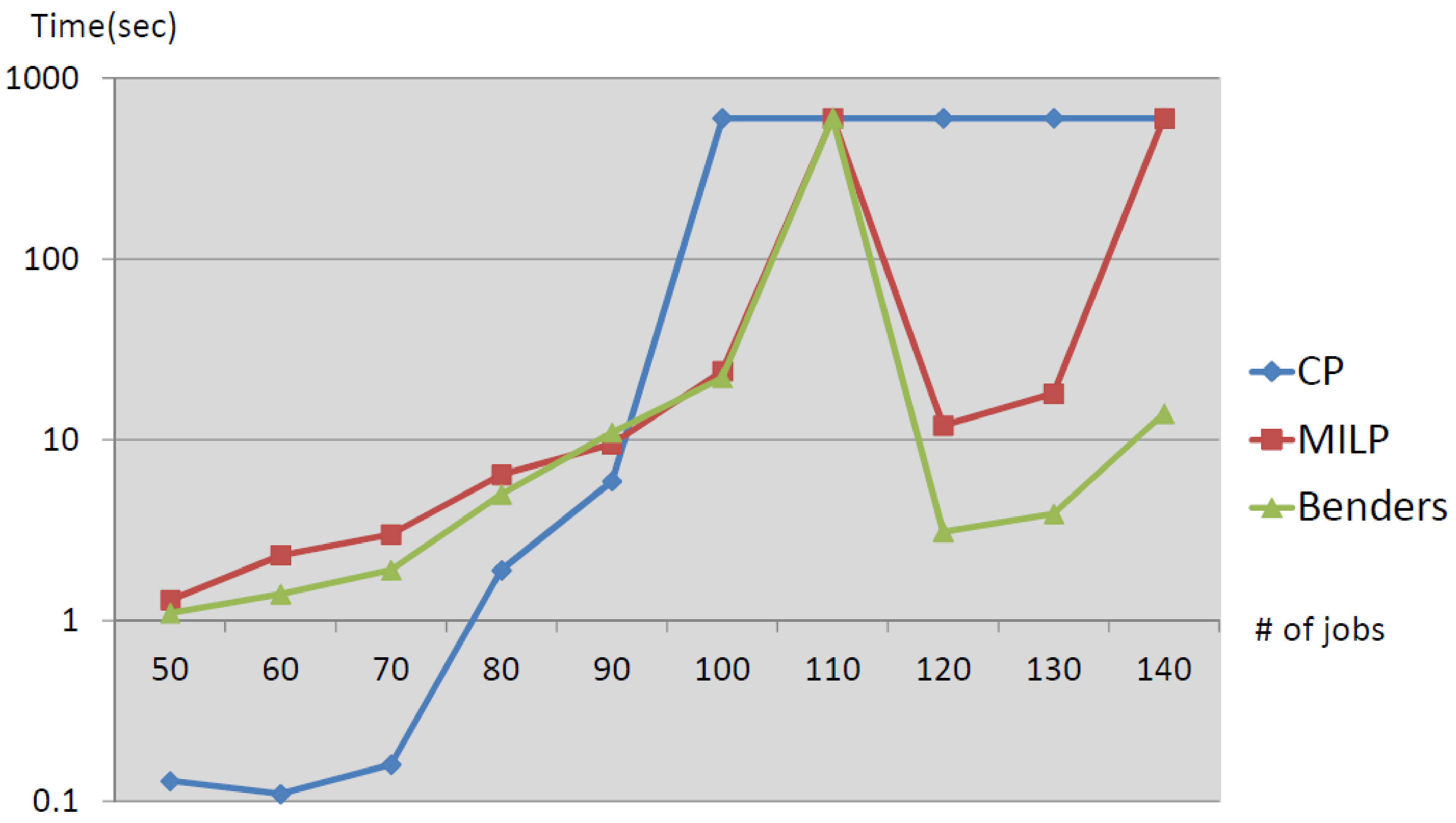
Segmented problem computational results

Min makespan – Tight time windows (individual instances)



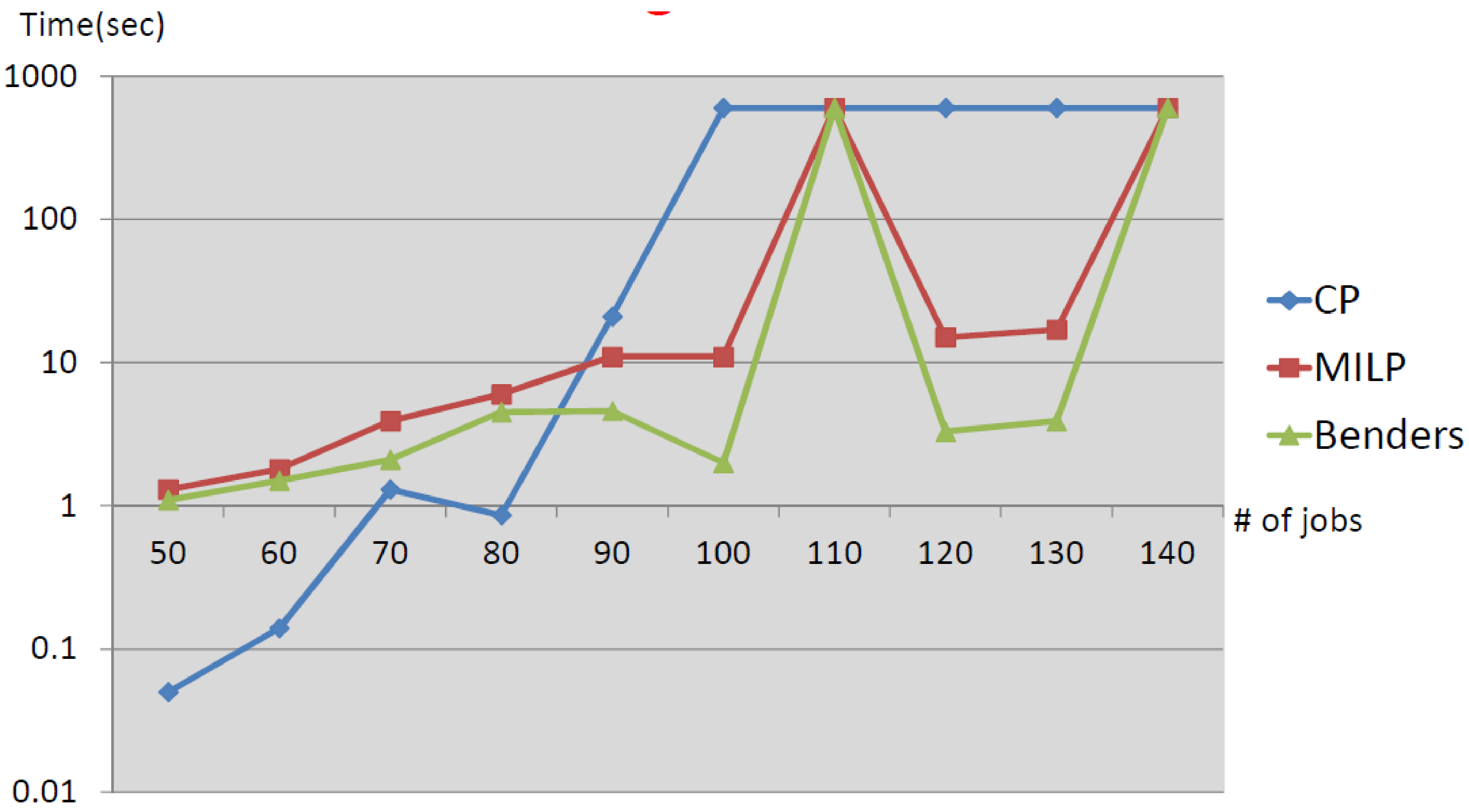
Segmented problem computational results

Min tardiness – Wide time windows (individual instances)



Segmented problem computational results

Min tardiness – Tight time windows (individual instances)



Segmented problem

Computational results – tight time windows

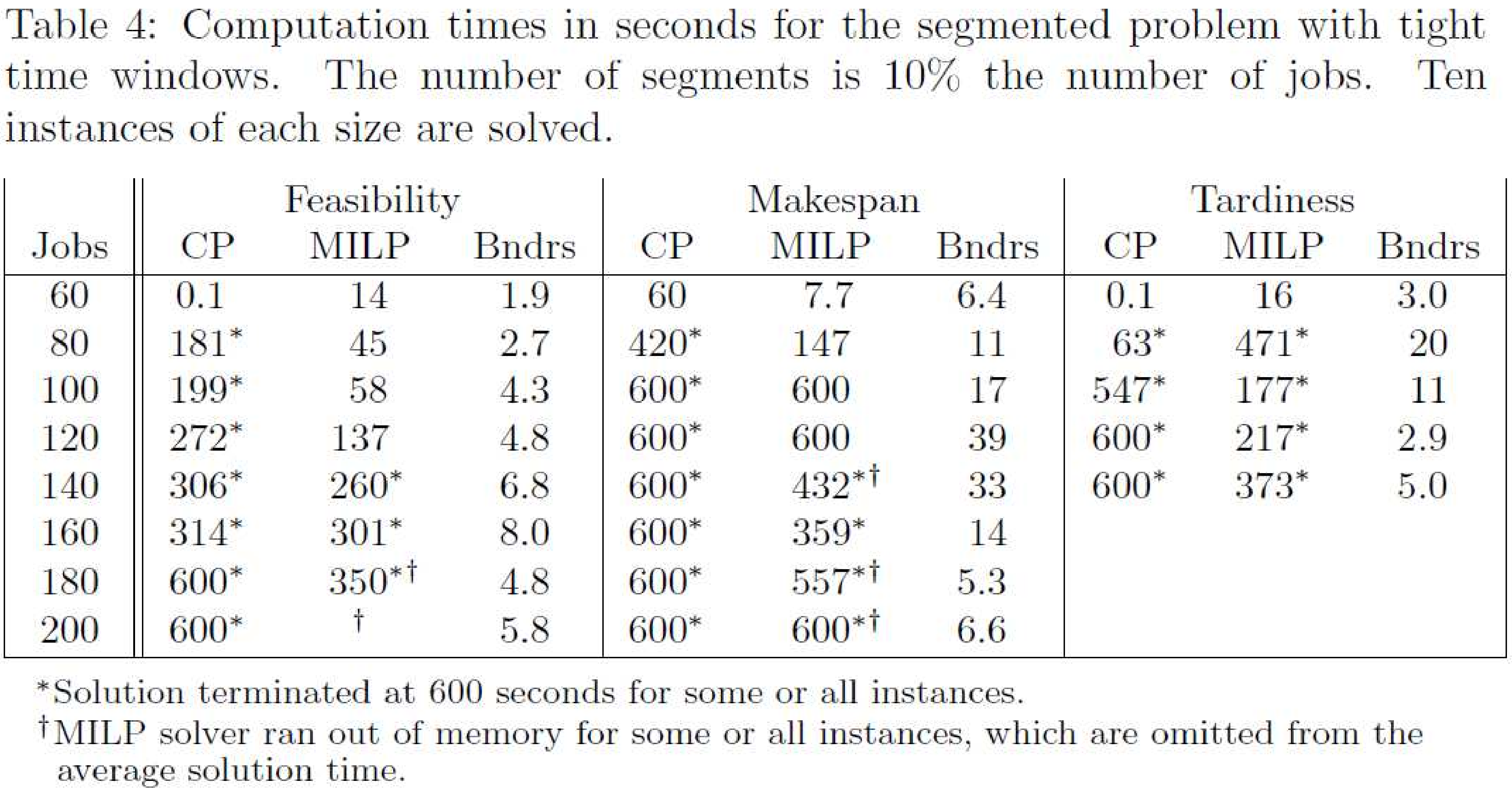


Table 4: Computation times in seconds for the segmented problem with tight time windows. The number of segments is 10% the number of jobs. Ten instances of each size are solved.

	Feasibility			Makespan			Tardiness		
Jobs	CP	MILP	Bndrs	CP	MILP	Bndrs	CP	MILP	Bndrs
60	0.1	14	1.9	60	7.7	6.4	0.1	16	3.0
80	181*	45	2.7	420*	147	11	63*	471*	20
100	199*	58	4.3	600*	600	17	547*	177*	11
120	272*	137	4.8	600*	600	39	600*	217*	2.9
140	306*	260*	6.8	600*	432*†	33	600*	373*	5.0
160	314*	301*	8.0	600*	359*	14			
180	600*	350*†	4.8	600*	557*†	5.3			
200	600*	†	5.8	600*	600*†	6.6			

*Solution terminated at 600 seconds for some or all instances.

†MILP solver ran out of memory for some or all instances, which are omitted from the average solution time.

Segmented problem

Computational results – wide time windows

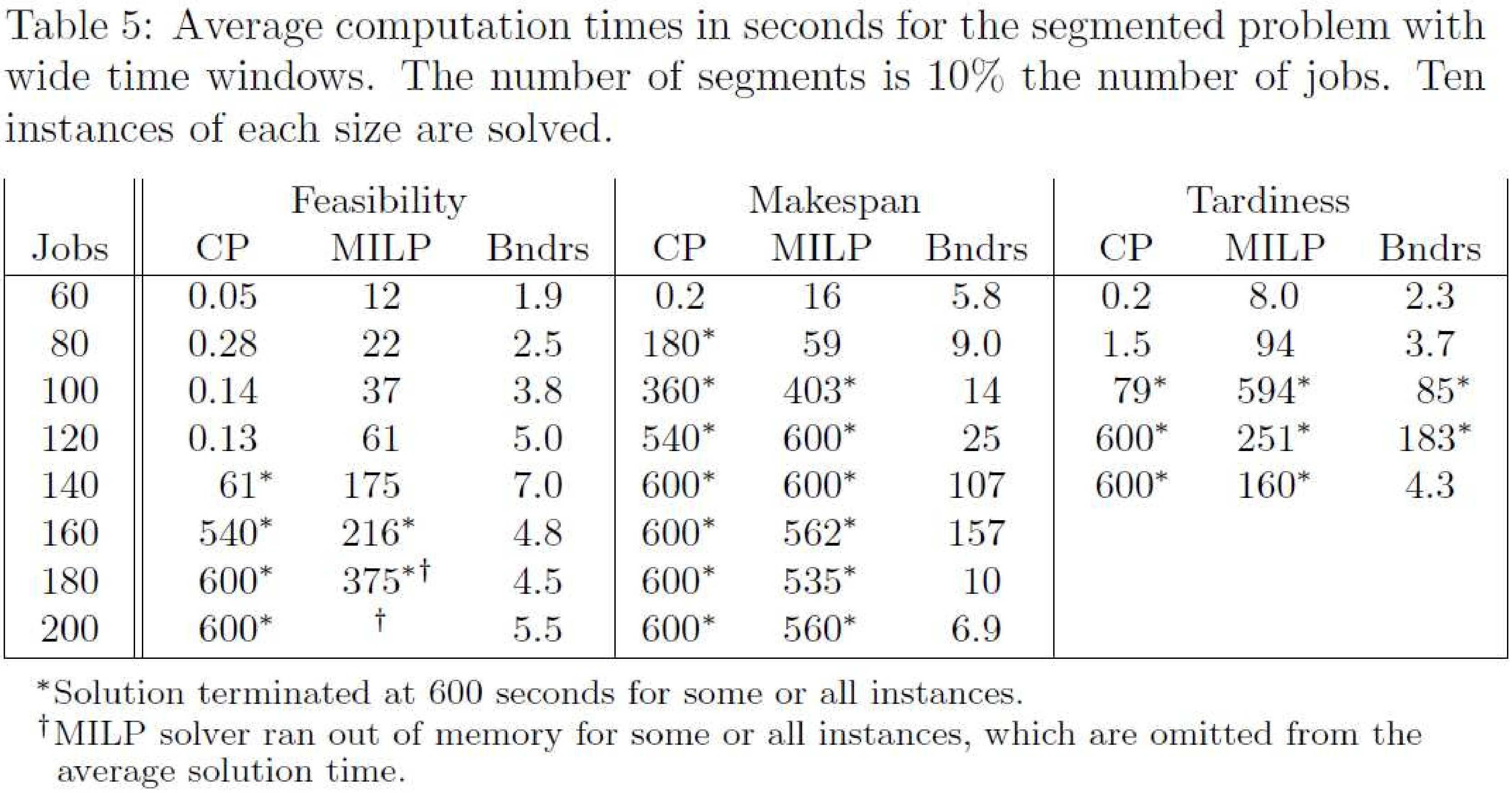


Table 5: Average computation times in seconds for the segmented problem with wide time windows. The number of segments is 10% the number of jobs. Ten instances of each size are solved.

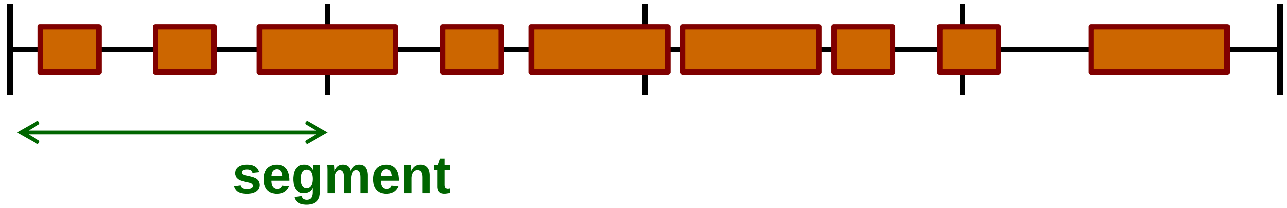
	Feasibility			Makespan			Tardiness		
Jobs	CP	MILP	Bndrs	CP	MILP	Bndrs	CP	MILP	Bndrs
60	0.05	12	1.9	0.2	16	5.8	0.2	8.0	2.3
80	0.28	22	2.5	180*	59	9.0	1.5	94	3.7
100	0.14	37	3.8	360*	403*	14	79*	594*	85*
120	0.13	61	5.0	540*	600*	25	600*	251*	183*
140	61*	175	7.0	600*	600*	107	600*	160*	4.3
160	540*	216*	4.8	600*	562*	157			
180	600*	375*†	4.5	600*	535*	10			
200	600*	†	5.5	600*	560*	6.9			

*Solution terminated at 600 seconds for some or all instances.

†MILP solver ran out of memory for some or all instances, which are omitted from the average solution time.

Unsegmented problem

- Master problem is more complicated.
  - Jobs can overlap two or more segments.
  - Master problem variables must keep track of this.
- Benders cuts more sophisticated.



Jobs can overlap
segment boundaries

Unsegmented problem

- **Master problem:**

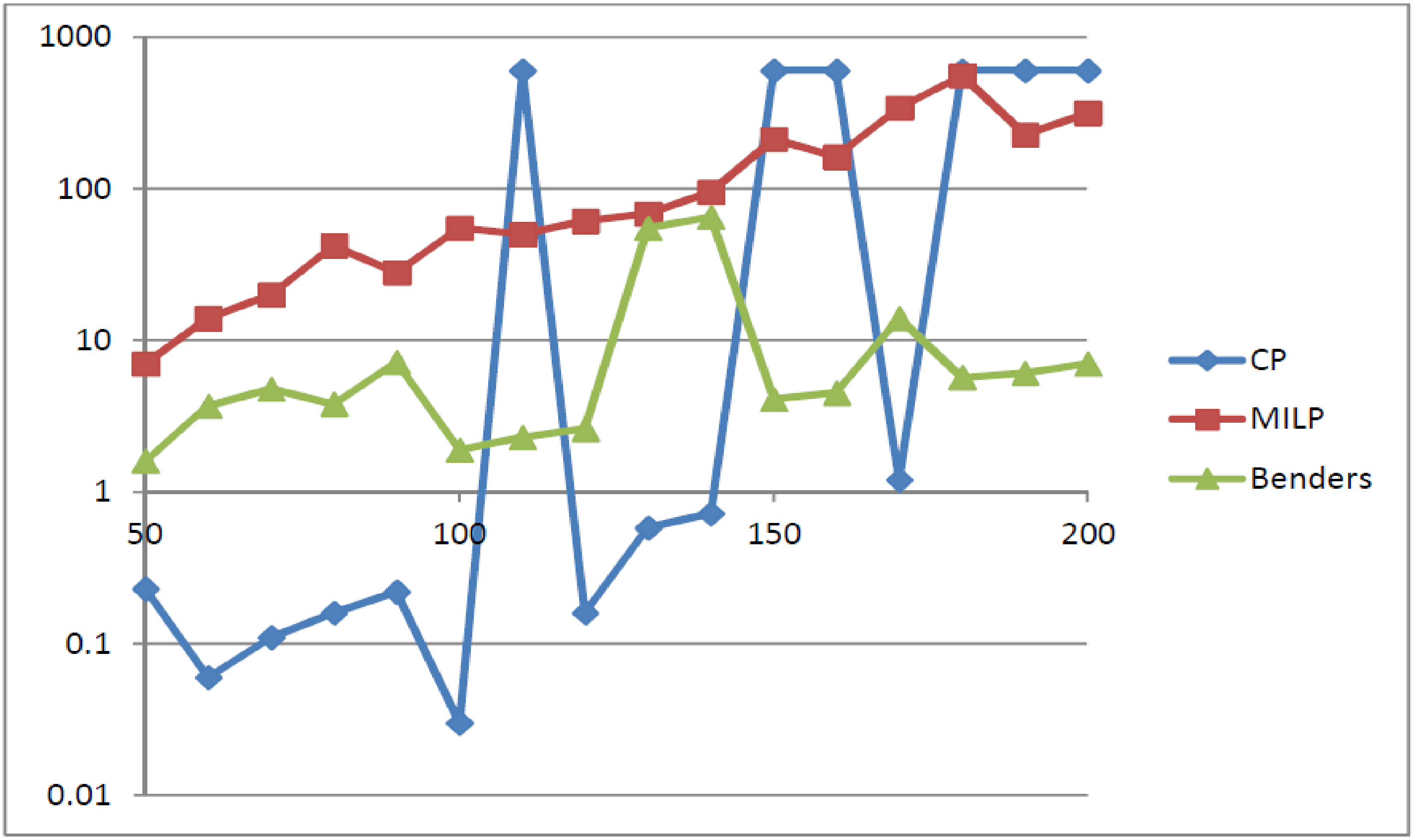
$$
\begin{aligned}
&\sum_{i\in I} y_{ij} \geq 1, \quad j \in J \\
&y_{ij} = y_{ij0} + y_{ij1} + y_{ij2} + y_{ij3}, \quad i \in I, j \in J \\
&\sum_{j\in J} y_{ij1} \leq 1, \quad \sum_{j\in J} y_{ij2} \leq 1, \quad \sum_{j\in J} y_{ij3} \leq 1, \quad i \in I \\
&y_{ij1} \leq y_{i-1,j,2} + y_{i-1,j,3}, \quad i \in I, i > 1, \ j \in J \\
&y_{ij2} \leq y_{i+1,j,1} + y_{i+1,j,3}, \quad i \in I, i < n, \ j \in J \\
&y_{ij3} \leq y_{i-1,j,3} + y_{i-1,j,2}, \quad i \in I, i > 1, \ j \in J \\
&y_{ij3} \leq y_{i+1,j,3} + y_{i+1,j,1}, \quad i \in I, i < n, \ j \in J \\
&\sum_{i\in I} y_{ij0} \leq 1, \quad \sum_{i\in I} y_{ij1} \leq 1, \quad \sum_{i\in I} y_{ij2} \leq 1, \quad j \in J \\
&y_{1j1} = y_{1j3} = y_{nj2} = y_{nj3} = 0, \quad j \in J \\
&\sum_{i\in I} y_{ij3} \leq \left\lfloor \frac{p_j}{a_{i+1} - a_i} \right\rfloor, \quad j \in J \\
&y_{ij}, y_{ij0}, y_{ij1}, y_{ij2}, y_{ij3} \in \{0,1\}, \quad i \in I, j \in J \\
&x_{ij1} \leq p_j y_{ij1}, \quad x_{ij2} \leq p_j y_{ij2} \\
&x_{ij} = p_j y_{ij0} + x_{ij1} + x_{ij2} + (a_{i+1} - a_i) y_{ij3} \\
&x_{ij1}, x_{ij2} \geq 0
\end{aligned}
$$

***y**$_{ijk}$* **variables keep track of whether job *j* starts, finishes, or runs entirely in segment *i*.**

***x**$_{ijk}$* **variables keep track of how long a partial job *j* runs in segment *i*.**

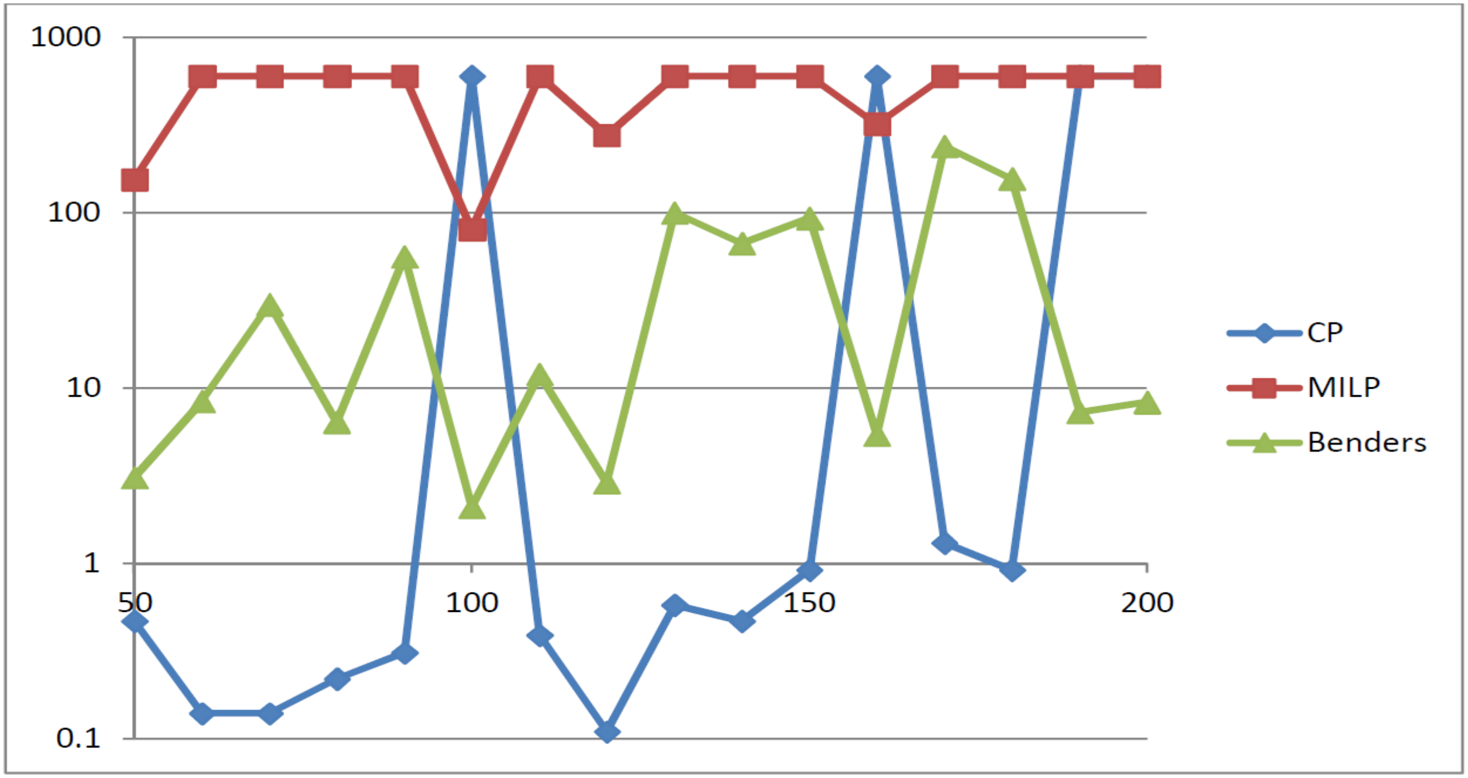
Unsegmented problem computational results

Feasibility -- individual instances



Unsegmented problem computational results

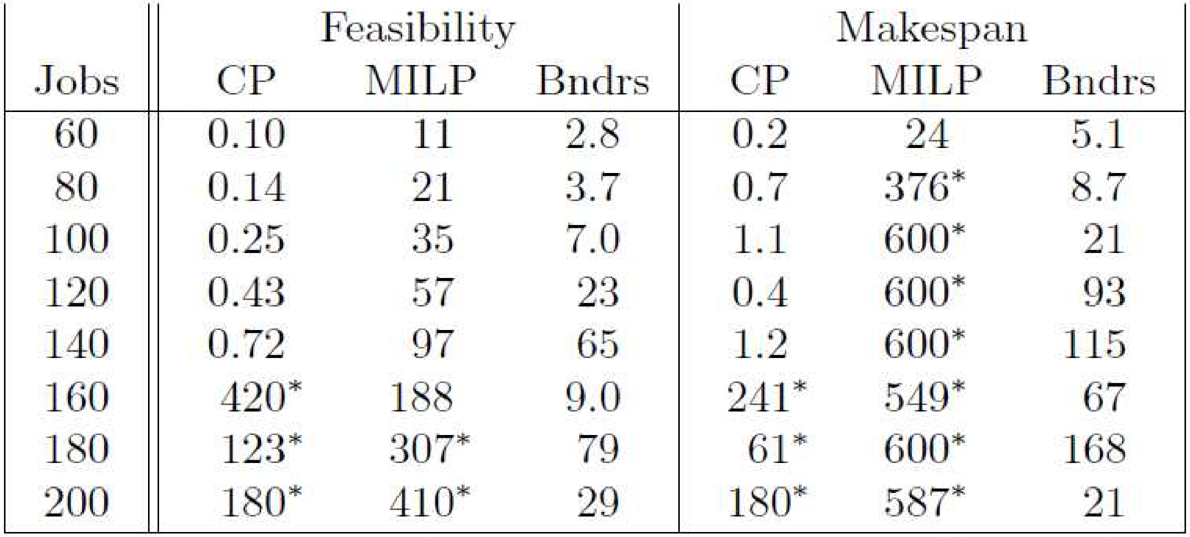
Min makespan – individual instances



Unsegmented problem

Computational results

Table 6: Average computation times in seconds for the unsegmented problem. The number of segments is 10% the number of jobs. Ten instances of each size are solved,

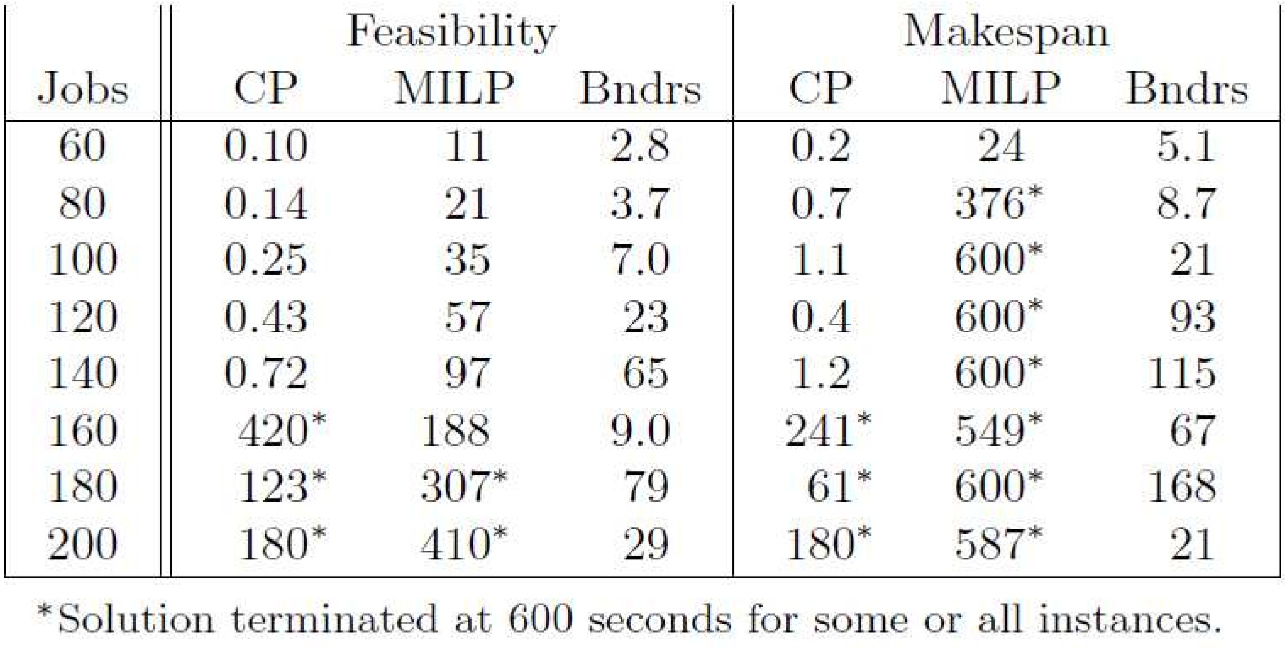


	Feasibility			Makespan		
Jobs	CP	MILP	Bndrs	CP	MILP	Bndrs
60	0.10	11	2.8	0.2	24	5.1
80	0.14	21	3.7	0.7	376*	8.7
100	0.25	35	7.0	1.1	600*	21
120	0.43	57	23	0.4	600*	93
140	0.72	97	65	1.2	600*	115
160	420*	188	9.0	241*	549*	67
180	123*	307*	79	61*	600*	168
200	180*	410*	29	180*	587*	21

*Solution terminated at 600 seconds for some or all instances.

Unsegmented problem

Computational results



	Feasibility			Makespan		
Jobs	CP	MILP	Bndrs	CP	MILP	Bndrs
60	0.10	11	2.8	0.2	24	5.1
80	0.14	21	3.7	0.7	376*	8.7
100	0.25	35	7.0	1.1	600*	21
120	0.43	57	23	0.4	600*	93
140	0.72	97	65	1.2	600*	115
160	420*	188	9.0	241*	549*	67
180	123*	307*	79	61*	600*	168
200	180*	410*	29	180*	587*	21

*Solution terminated at 600 seconds for some or all instances.

CP solves it quickly (< 1 sec) or blows up, in which case Benders solves it in 6 seconds (average).

Summary of results

- **Segmented problems:**
  - **Benders is much faster for min cost and min makespan problems.**
  - **Benders is somewhat faster for min tardiness problem.**

Summary of results

- Segmented problems:
  - Benders is much faster for min cost and min makespan problems.
  - Benders is somewhat faster for min tardiness problem.
- Unsegmented problems:
  - Benders and CP can work together.
  - Let CP run for 1 second.
  - If it fails to solve the problem, it will probably blow up. Switch to Benders for reasonably fast solution.

Recent work – Cutting Planes from CP Model

Joint work with David Bergman.

Polyhedral analysis of overlapping all-different constraints (equivalent to graph coloring).

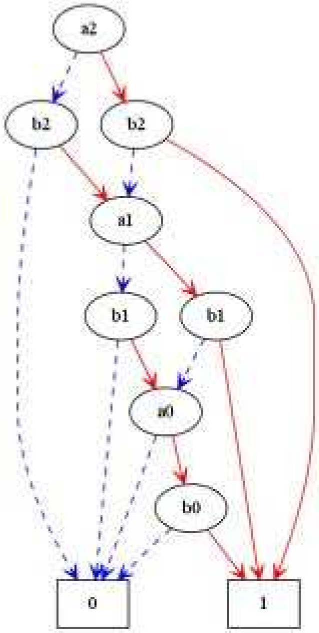
Used in many scheduling problems, sudoku puzzles, etc. etc.

Derive cutting planes from CP alldiff formulation and map them into 0-1 model.

Provides tighter bounds than all CPLEX cuts in a small fraction of the time (e.g., 1%).

Recent work – BDDs as Constraint Store

Joint work with Henrik Andersen, David Bergman, Andre Cire, Tarik Hadzic, Willem van Hoeve, Barry O'Sullivan, Peter Tiedemann



Replace variable domains in CP with relaxed **binary decision diagrams** (BDDs).

BDDs have long been used for circuit design, configuration, etc.

We use them to represent relaxation of feasible set.

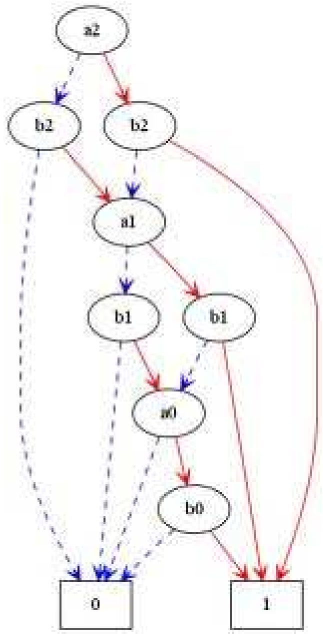
Replace domain filtering with BDD-based propagation.

Reduces search tree for multiple alldiffs from 1 million nodes to 1 node, time speedup factor of 100. Speedups on other problems.

Now being incorporated into **Google CP solver**.

Recent work – BDDs for Relaxation Bounding

Joint work with David Bergman, Andre Cire,
Willem van Hoeve



Replace LP relaxation with a relaxed **binary decision diagram** (BDD).

Shortest path in BDD provides a lower bound on optimal value.

For most instances of independent set problem, we get tighter bounds than full cutting plane technology in CPLEX.

Bound is normally obtained in very small fraction of the time.

Obrigado!

Vocês têm perguntas?