Implementation of Global Constraints

Pascal Van Hentenryck

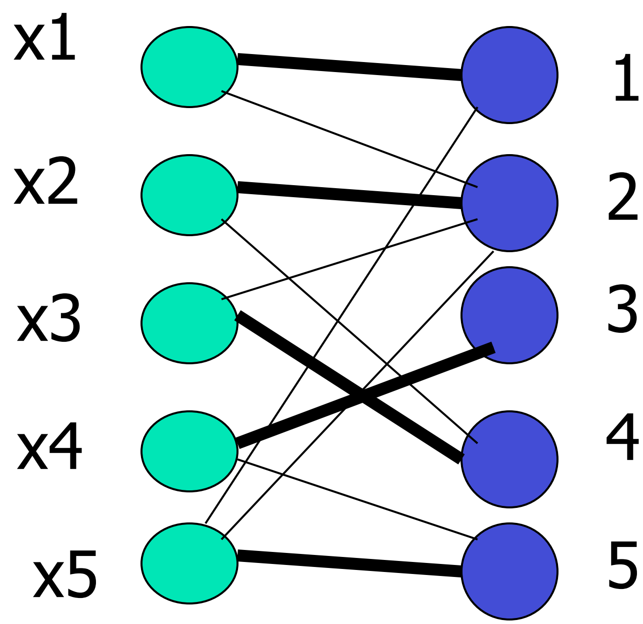
Brown University

Outline

- Alldifferent
- Binary Knapsack
- Disjunctive Constraint

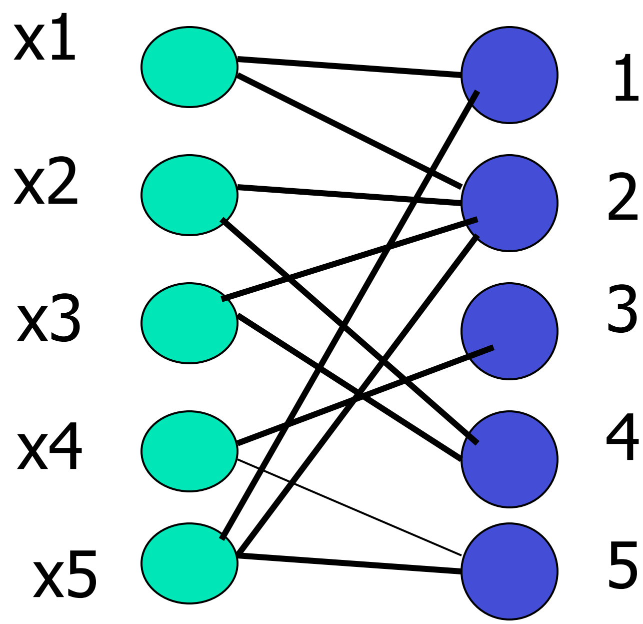
Alldifferent: Feasibility

- Feasibility? Given domains, create domain/variable bipartite graph



Alldifferent: Pruning

- Pruning? Which edges are in no matching?

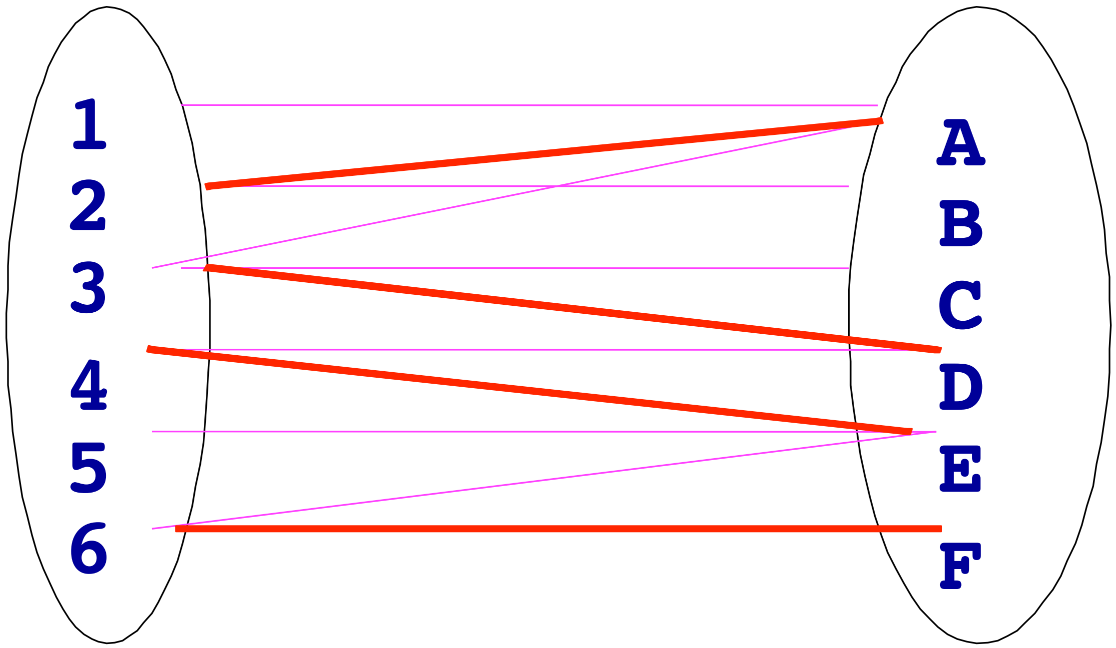


Matching

- A matching for a graph G=(V,E) is a set of edges in E such that no two edges in E share a vertex
- A maximum matching M for a graph G is a matching with the largest number of edges
- A bipartite graph is a graph where the vertices can be partitioned into two sets and all edges connect vertices from different sets.
- Find a maximum matching in a bipartite graph

Matching

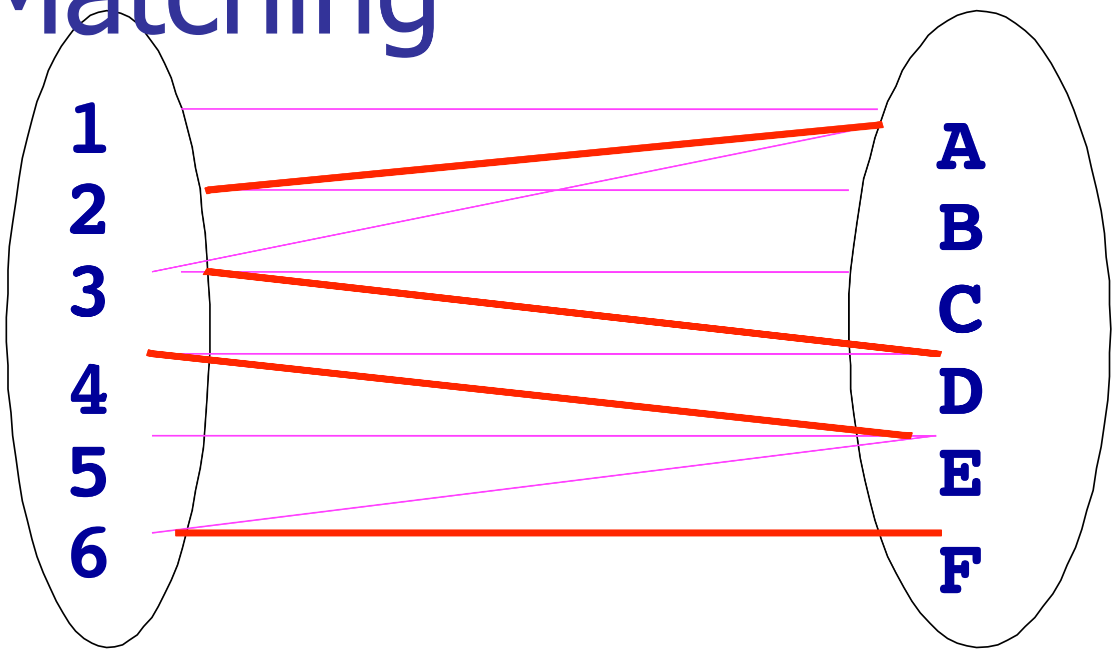
How to improve a maximal matching?



Replace 2A by 1A and 2B

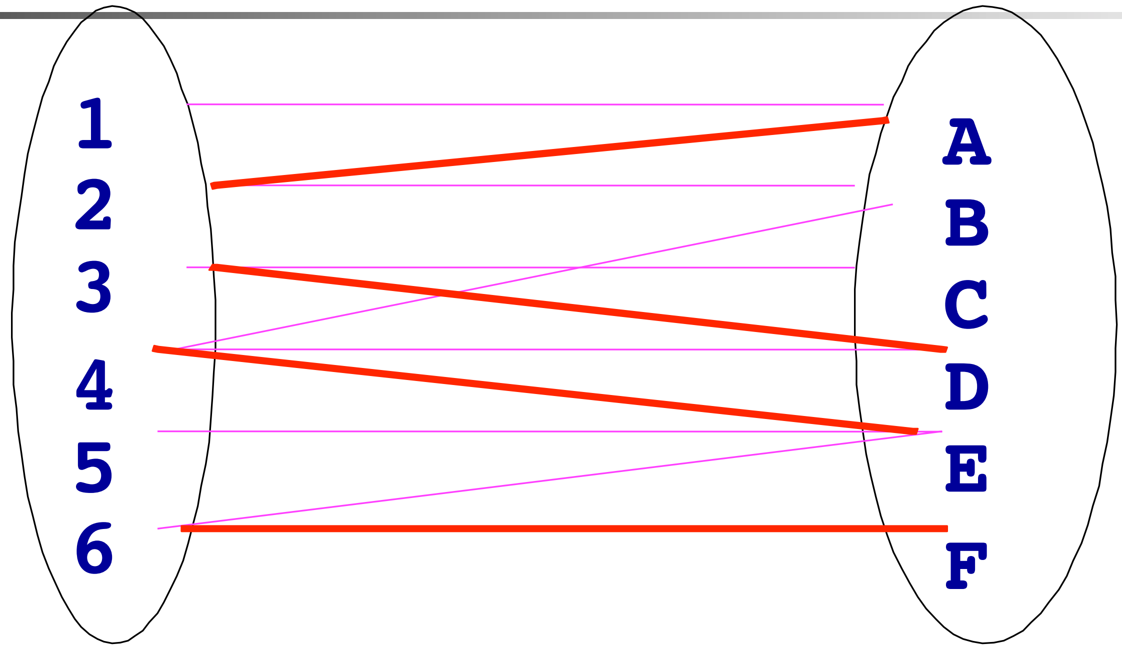
Replace 3D & 4E by 3C

Matching



- Select a free vertex x. Since the matching is maximal, all its neighbors are matched.
- Take a vertex v adjacent to x that was matched to y
- Match x to v and make y free
- If y is adjacent to a free vertex, we are done
- Otherwise, apply the same idea to y

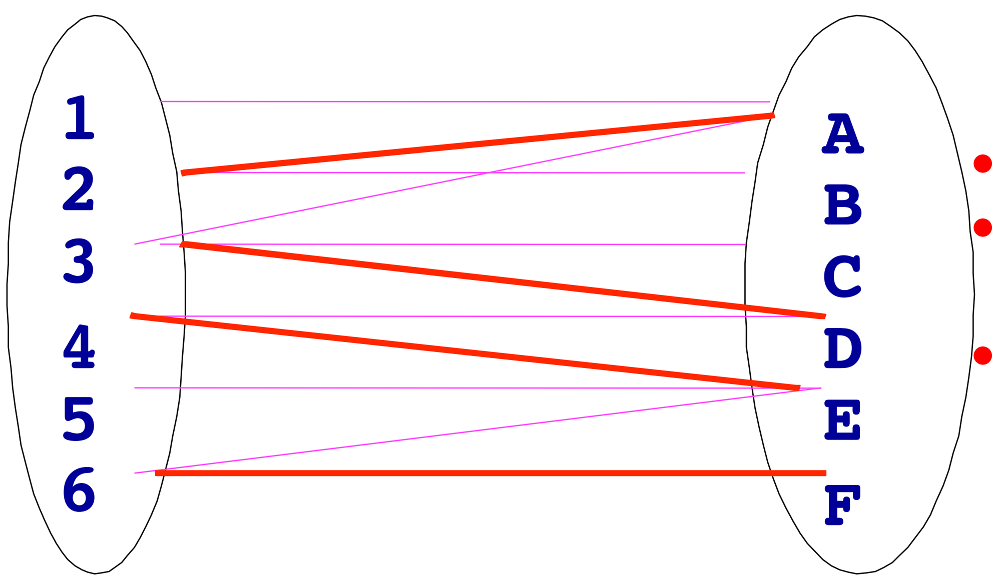
Matching



- Show that the procedure terminates
- Show that, if there is an improvement, an algorithm based on this idea would find it

Bipartite Matching

An alternating path **P** for a matching **M** is a path from a vertex **x** in **X** to a vertex **v** in **V** (both of which are free) such that the edges in the path are alternatively in **E\M** and **M**.

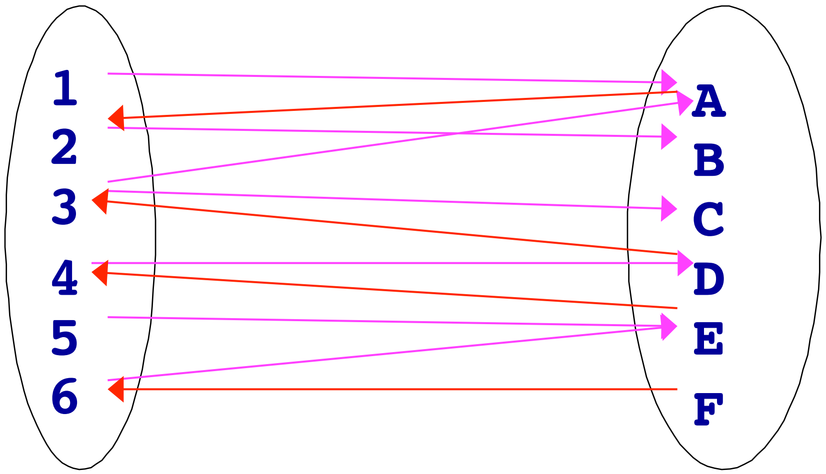


- the number of edges is odd
- there is one more edge in **E\M** than in **M**
- **(5E, E4, 4D, D3, 3C)** is an alternating path

Bipartite Matching

Given a matching M, orient the edges

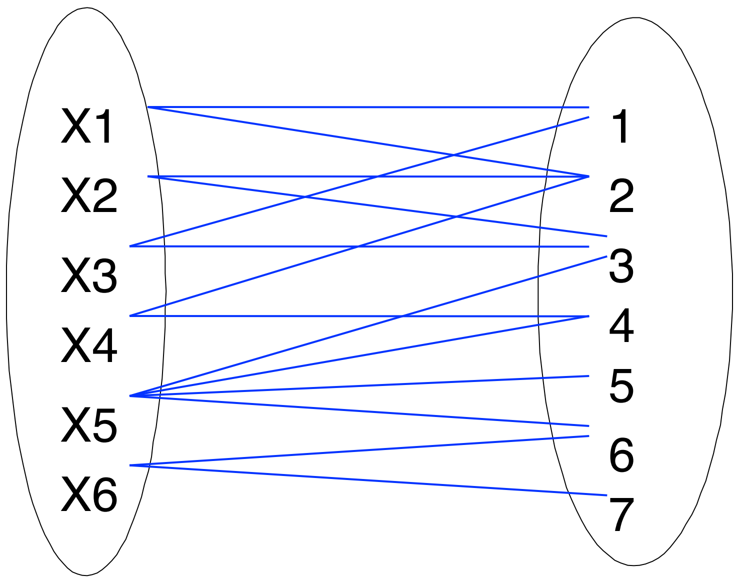
- Edges in M go from V to X
- Other edges go from X to V



- An alternating path is thus a directed path from a free vertex in X to a free vertex in V
- Use depth-first search or breadth-first search
- Complexity: O(|X| + |E|)

Alldifferent

- Feasibility
  - Finding a bipartite matching that covers all vertices in X



Alldifferent: Pruning

- Pruning
  - The matching tells us that there is at least one solution
  - The problem now is to remove values that do not belong to any solution
- Naïve approach
  - Force an edge (x,v) to be in the matching
  - Apply a bipartite matching to the resulting graph
  - If no matching can be found that covers all remaining vertices, then the assignment (x,v) does not belong to any solution

Alldifferent: Pruning

- Basic Property (Berge, 1970)
  - An edge belongs to some, but not all maximum matching, iff, for an arbitrary matching M, it belongs to either an even alternating path starting a free vertex or to an even alternating cycle

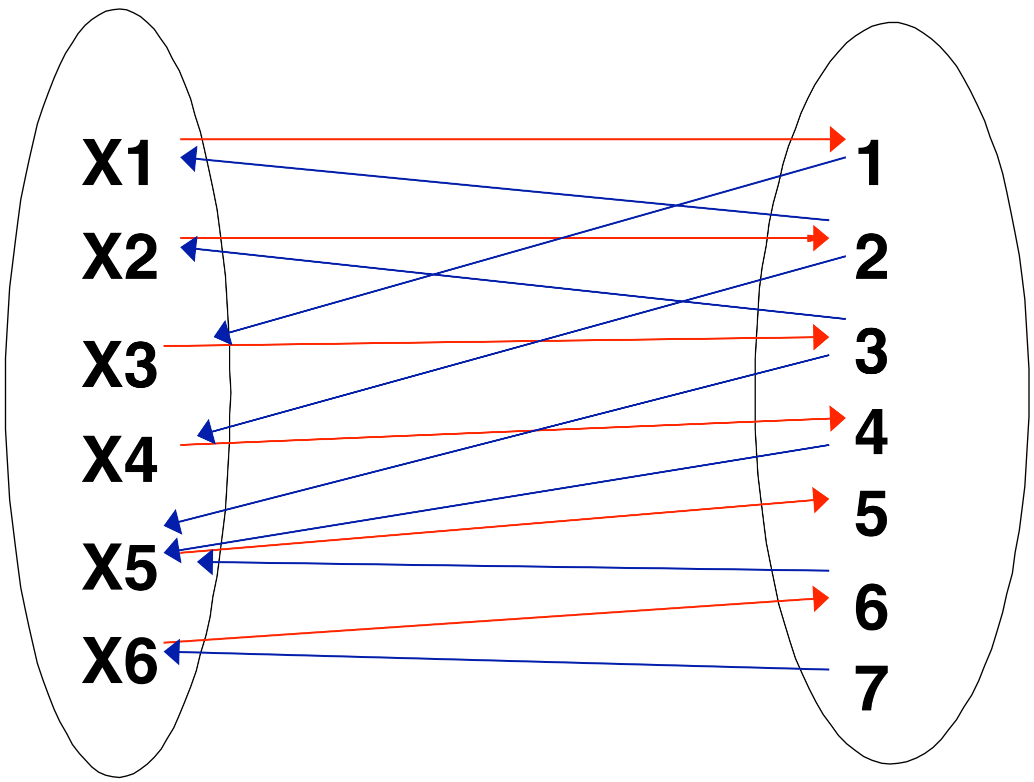
Alldifferent: Pruning

Pruning algorithm: basic idea

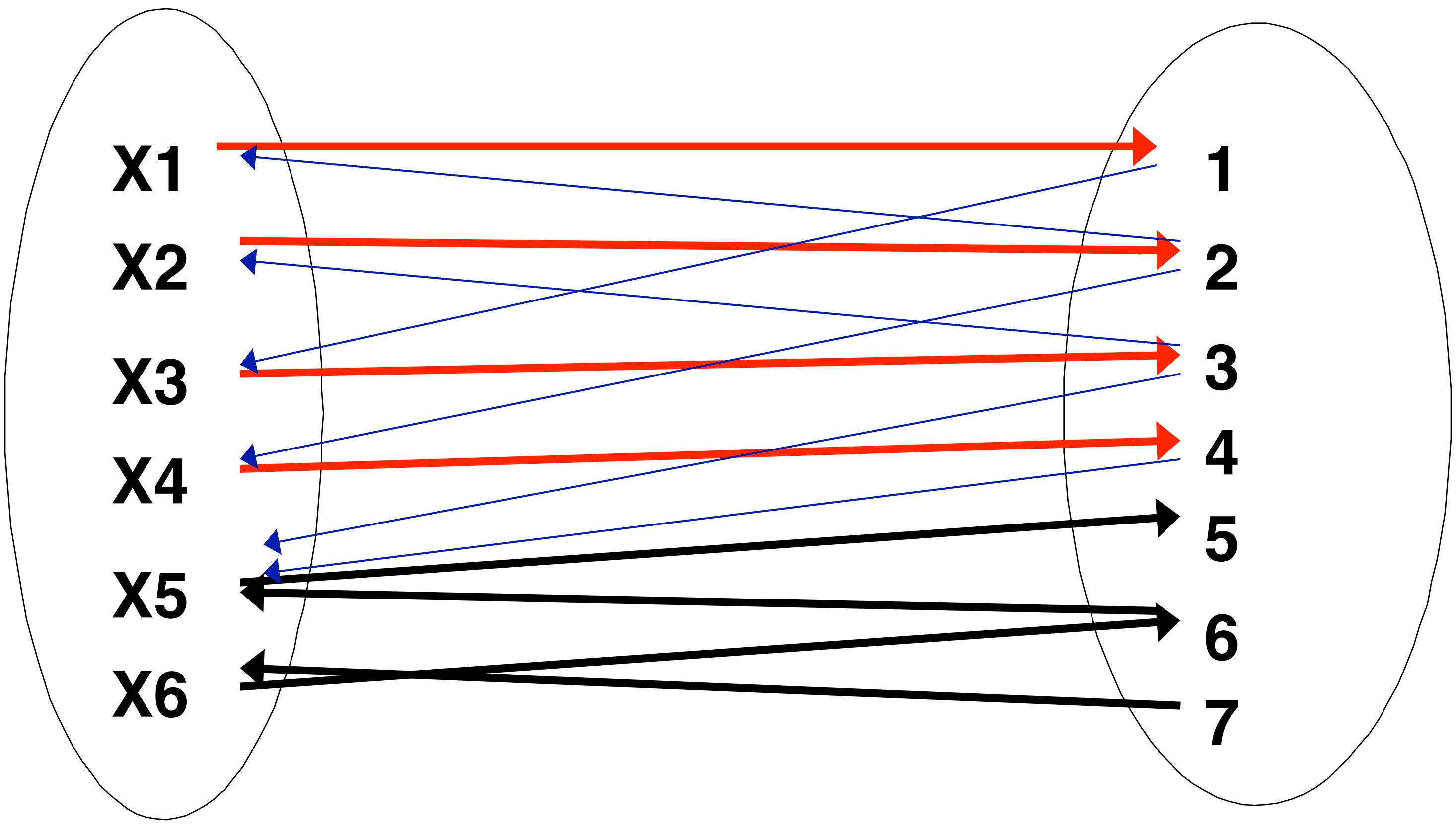
- Use the directed graph seen previously but change the direction of all edges
- Search for the edges belonging to an alternating path starting a free vertex: P
- Search for strongly connected components and collect all the edges belonging to them: C
- All the vertices in E \ (P union C union M) can be deleted

Complexity: $O(|X + V| \, |E|)$

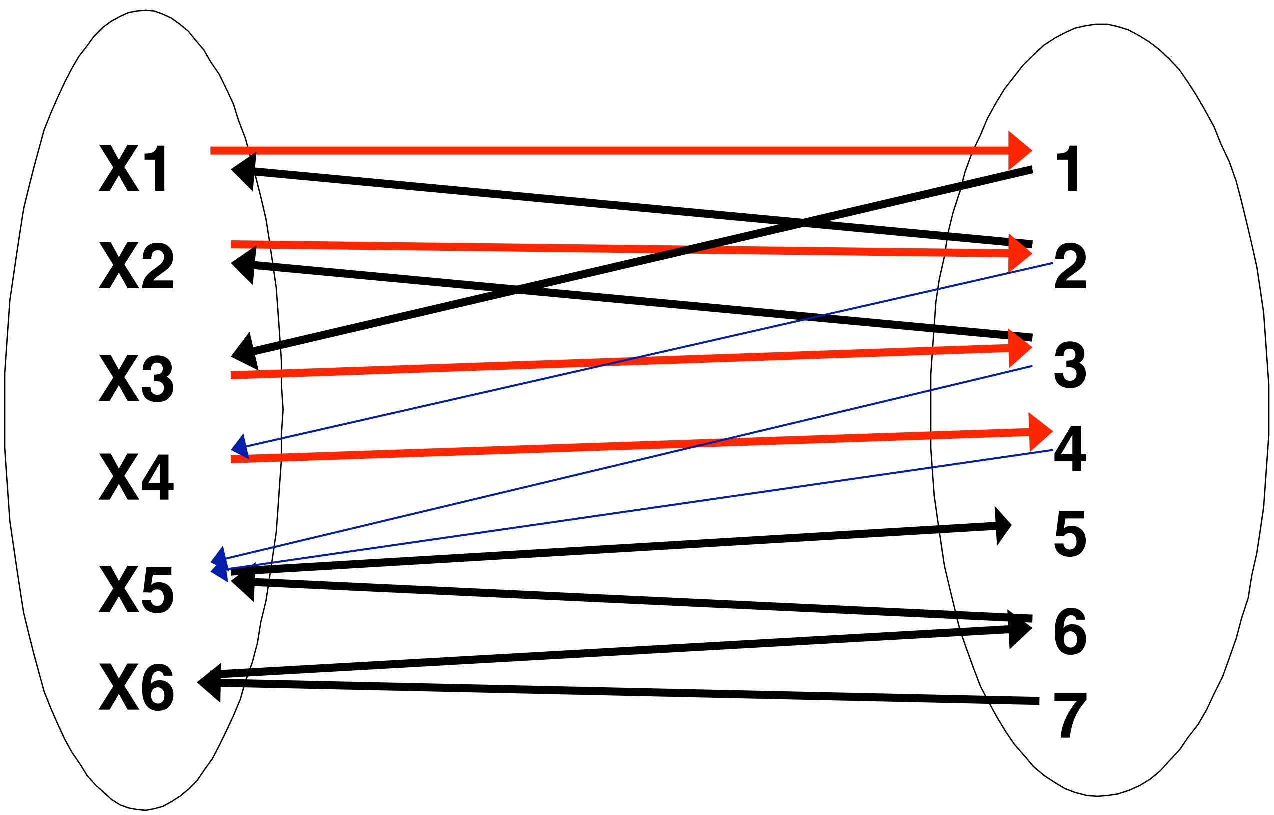
Alldifferent: Pruning



Alldifferent: Pruning



Alldifferent: Pruning



Outline

- Alldifferent
- Binary Knapsack
- Disjunctive Constraint

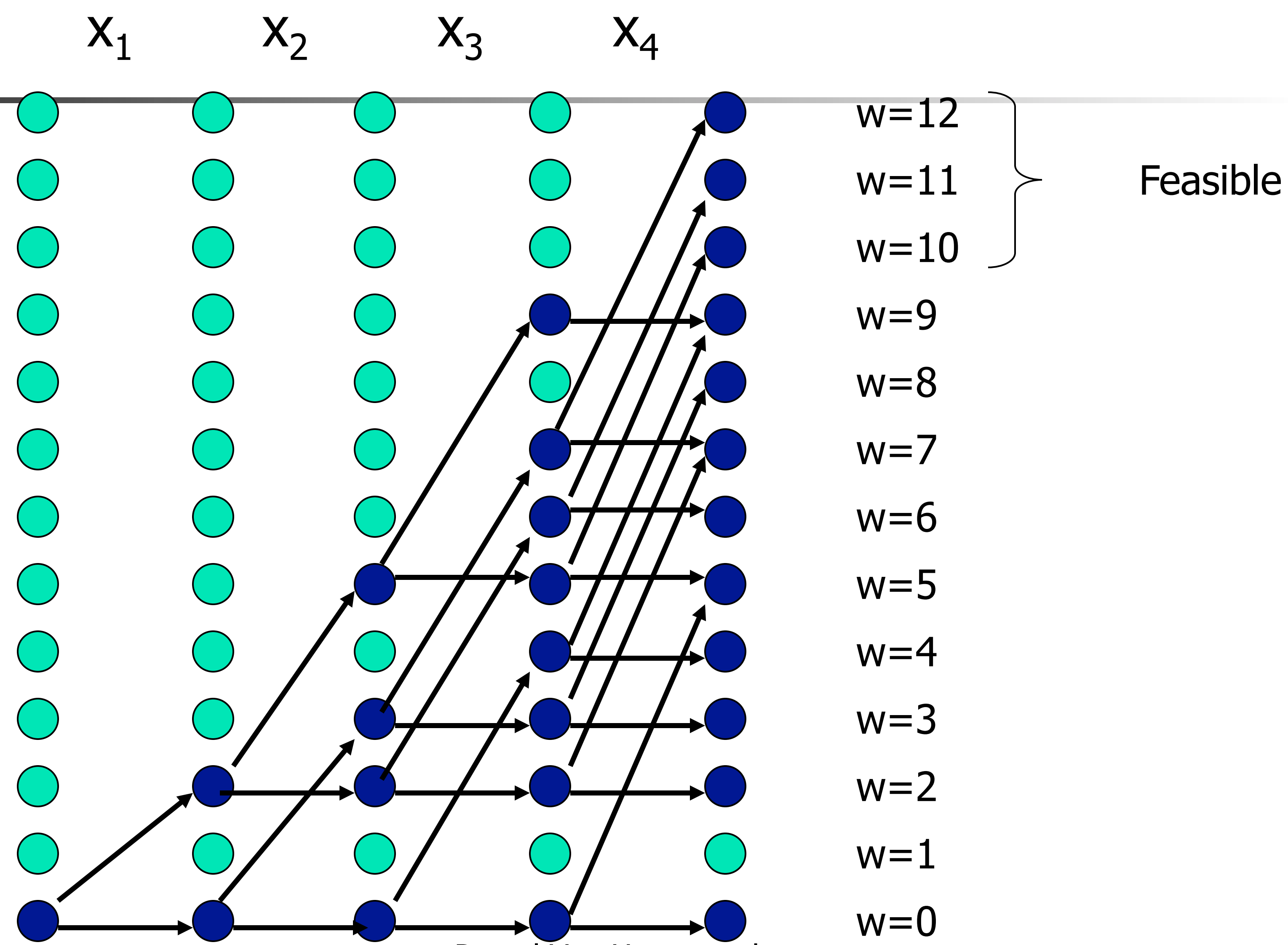
Binary Knapsack

- The problem
  - l <= sum(k in R) w[k] x[k] <= u
  - 0 <= x[k] <= 1
- Example
  - $10 \leq 2\,x_1 + 3\,x_2 + 4\,x_3 + 5\,x_4 \leq 12$
  - Is it feasible?
  - Can some values be pruned?

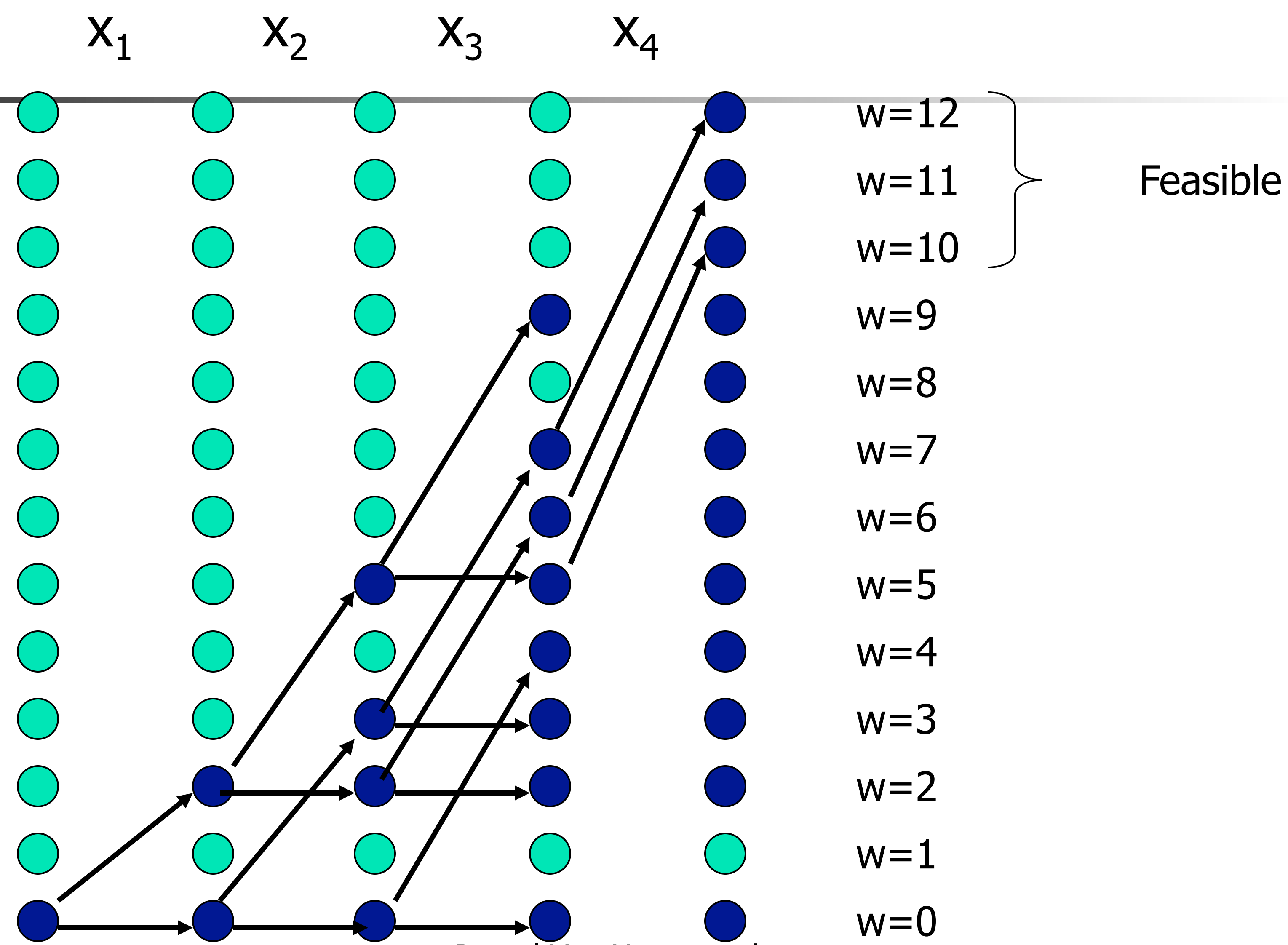
Binary Knapsack: Feasibility

- The problem
  - l <= sum(k in R) w[k] x[k] <= u
  - 0 <= x[k] <= 1
- Algorithm
  - use a dynamic program
  - pseudo-polynomial

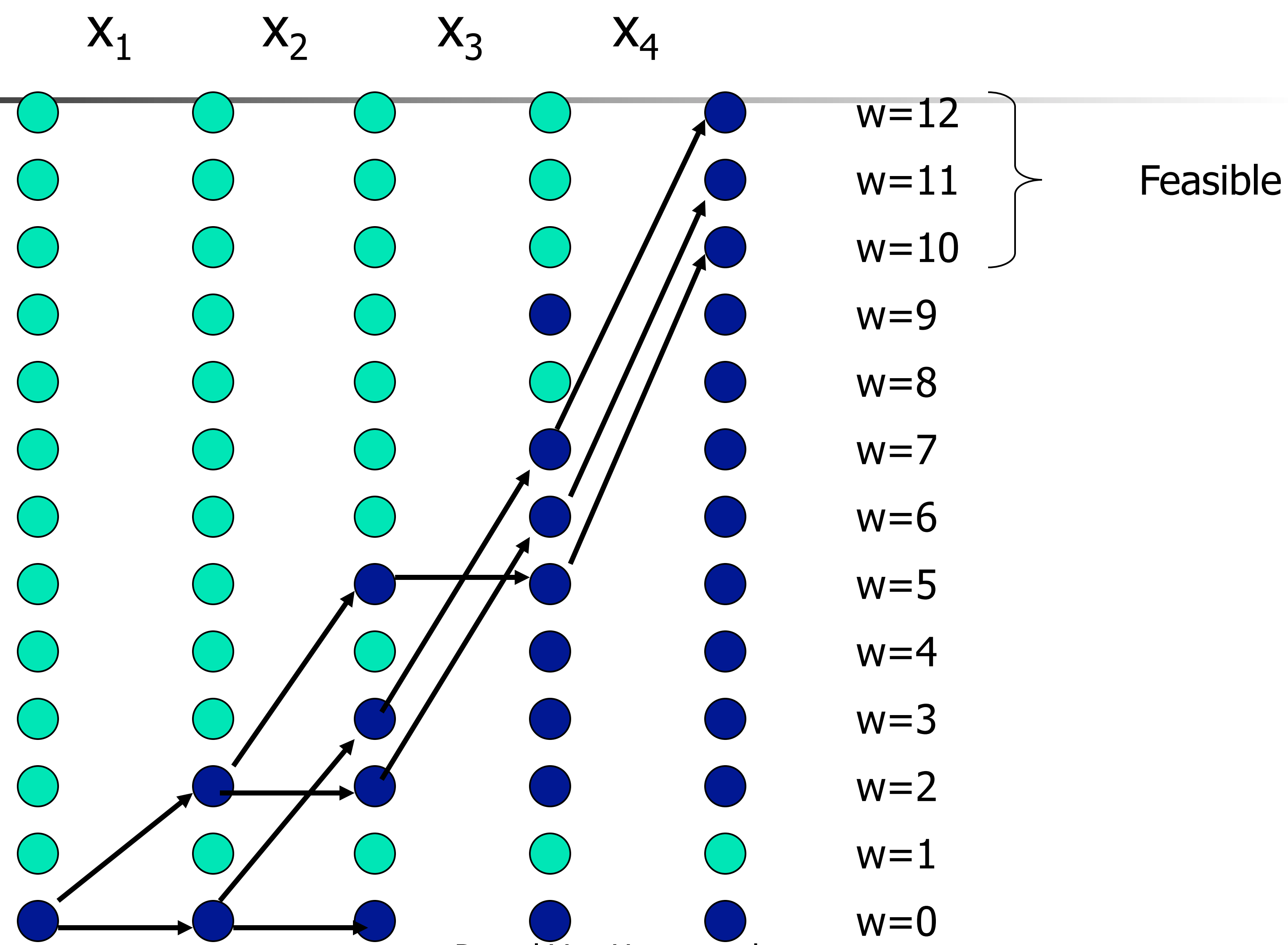
Binary Knapsack: Feasibility



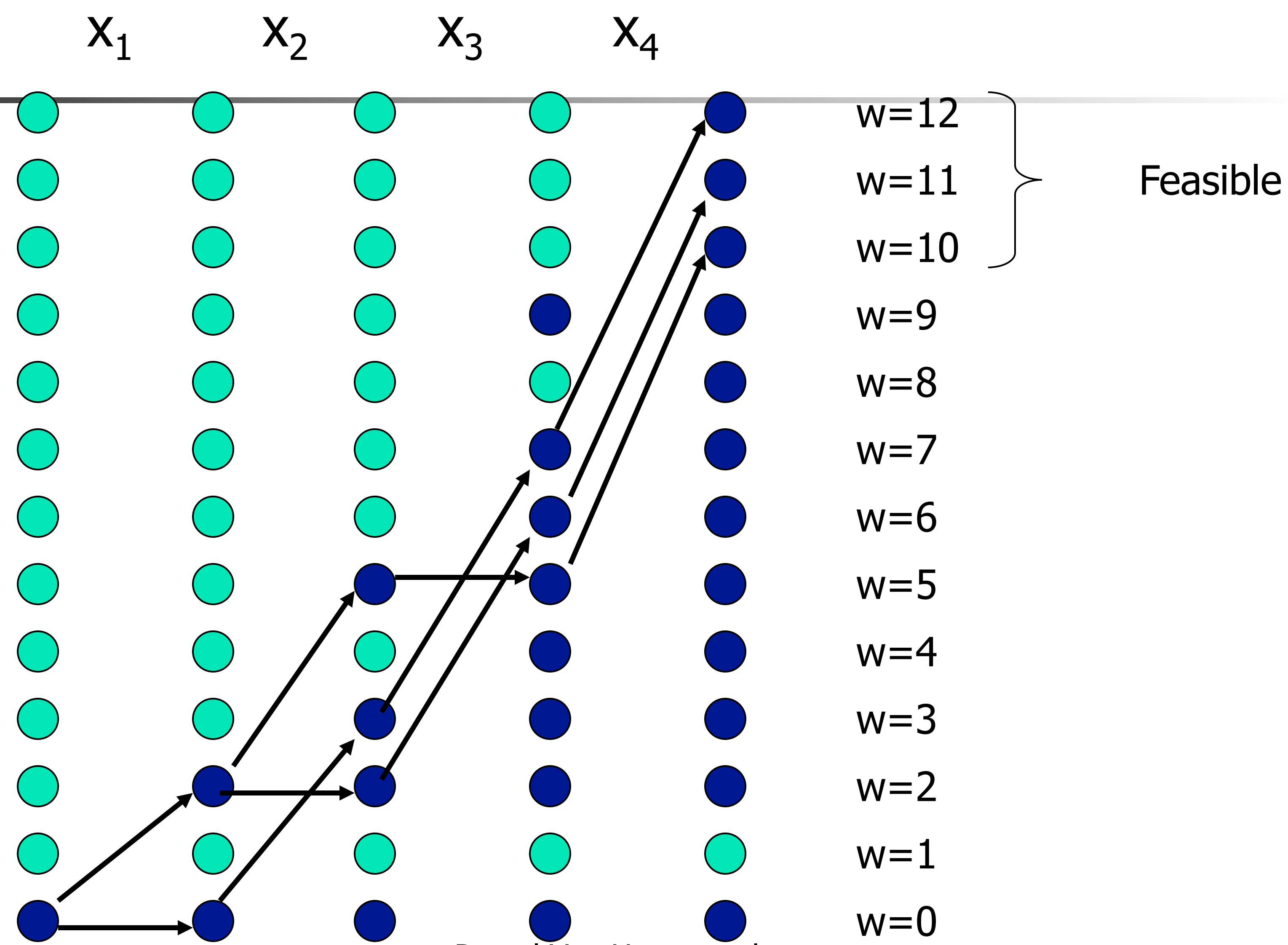
Binary Knapsack: Feasibility



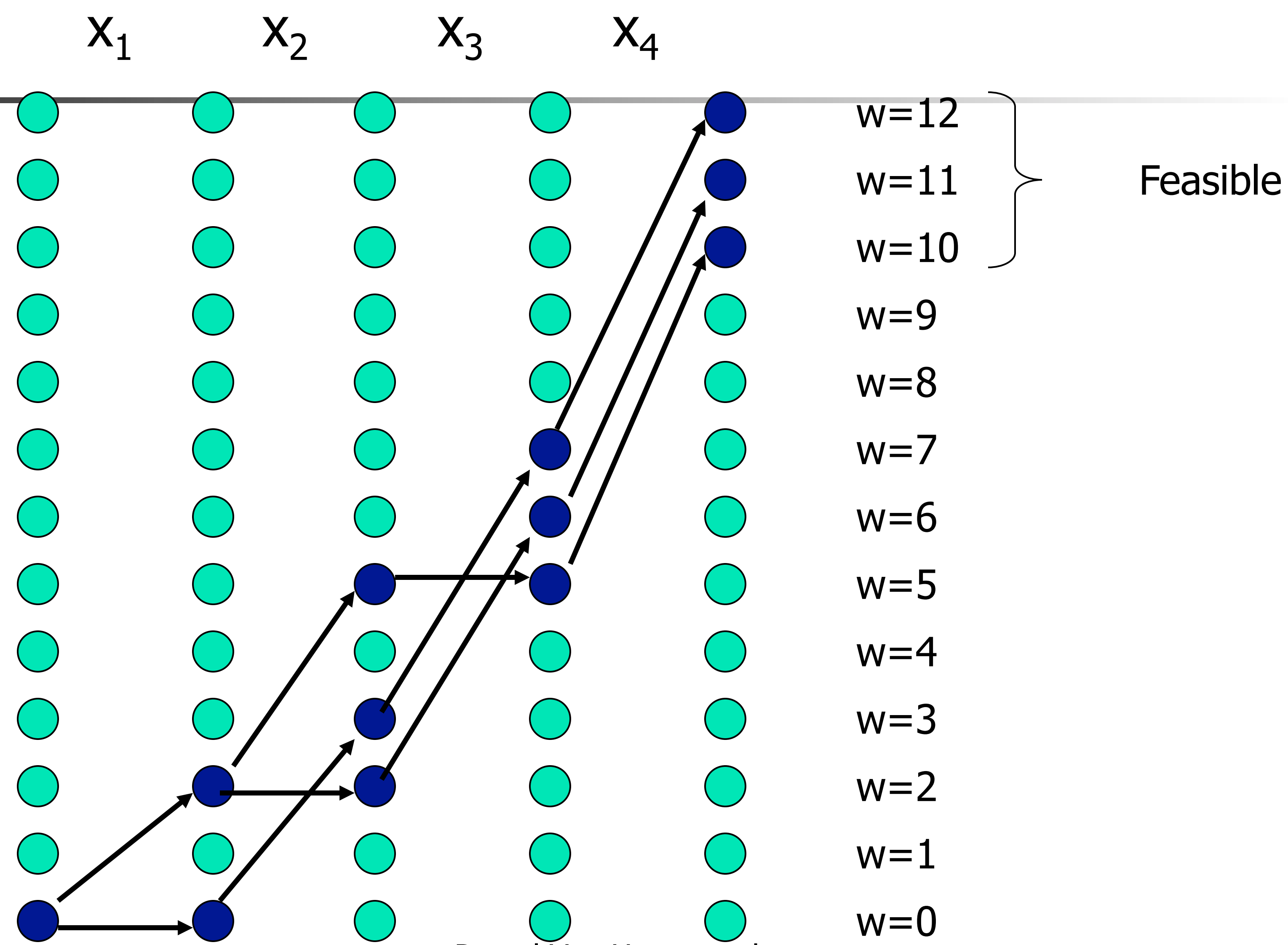
Binary Knapsack: Feasibility



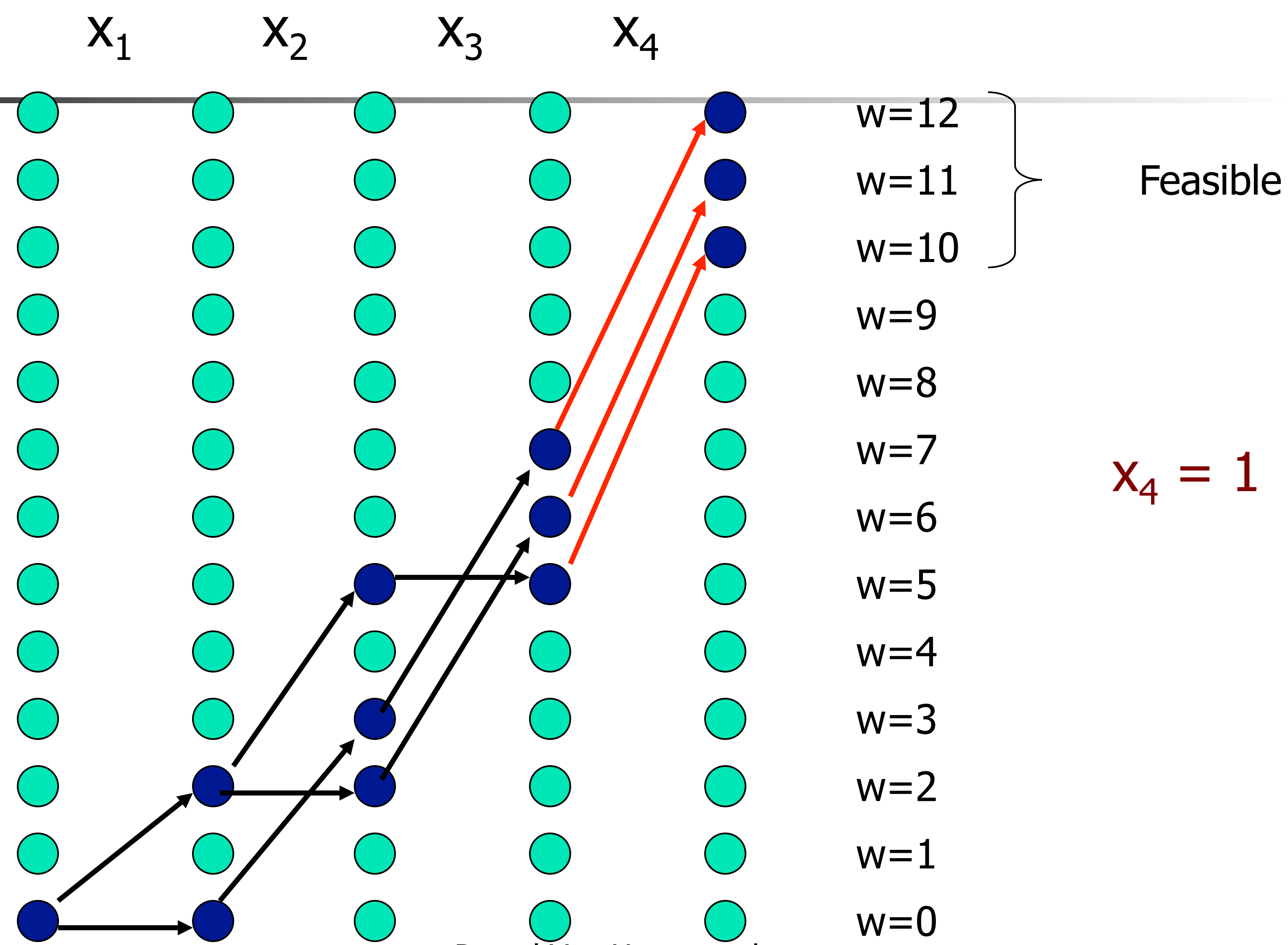
Binary Knapsack: Feasibility



Binary Knapsack: Pruning



Binary Knapsack: Pruning



Outline

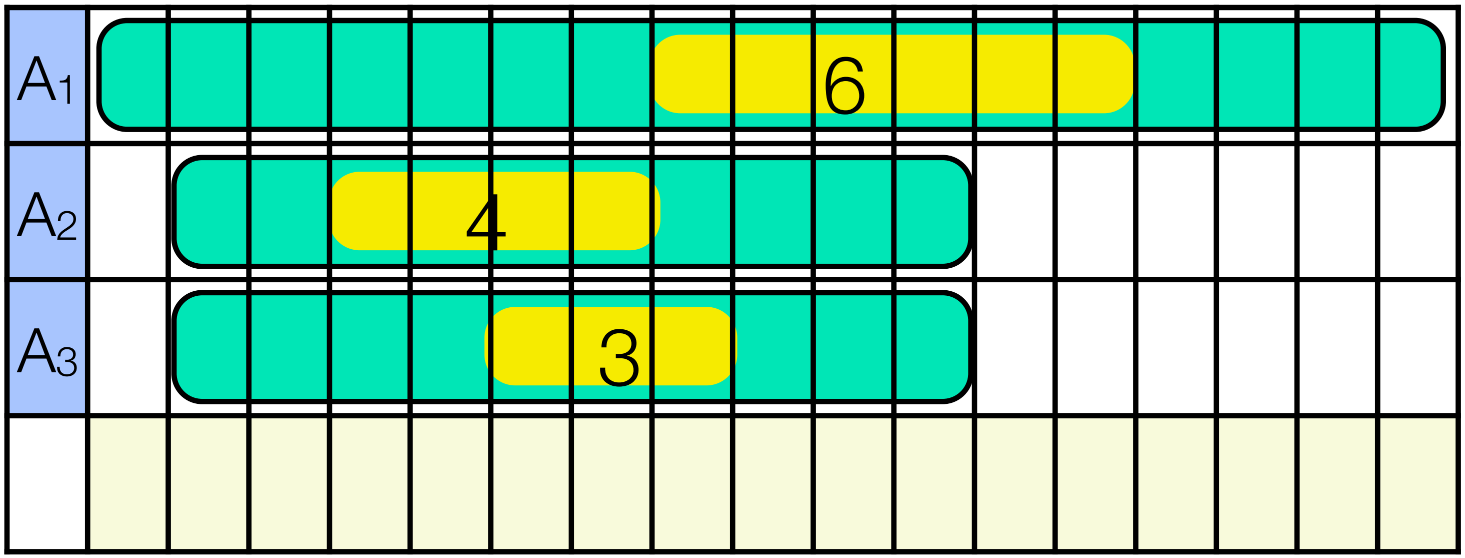
- Alldifferent
- Binary Knapsack
- Disjunctive Constraint

Disjunctive Scheduling

- One Machine Feasibility
  - given a set of tasks T
  - each task t has an earliest starting date
  - each task t has a latest starting date
  - Can I schedule them so that no two tasks a and b overlap?

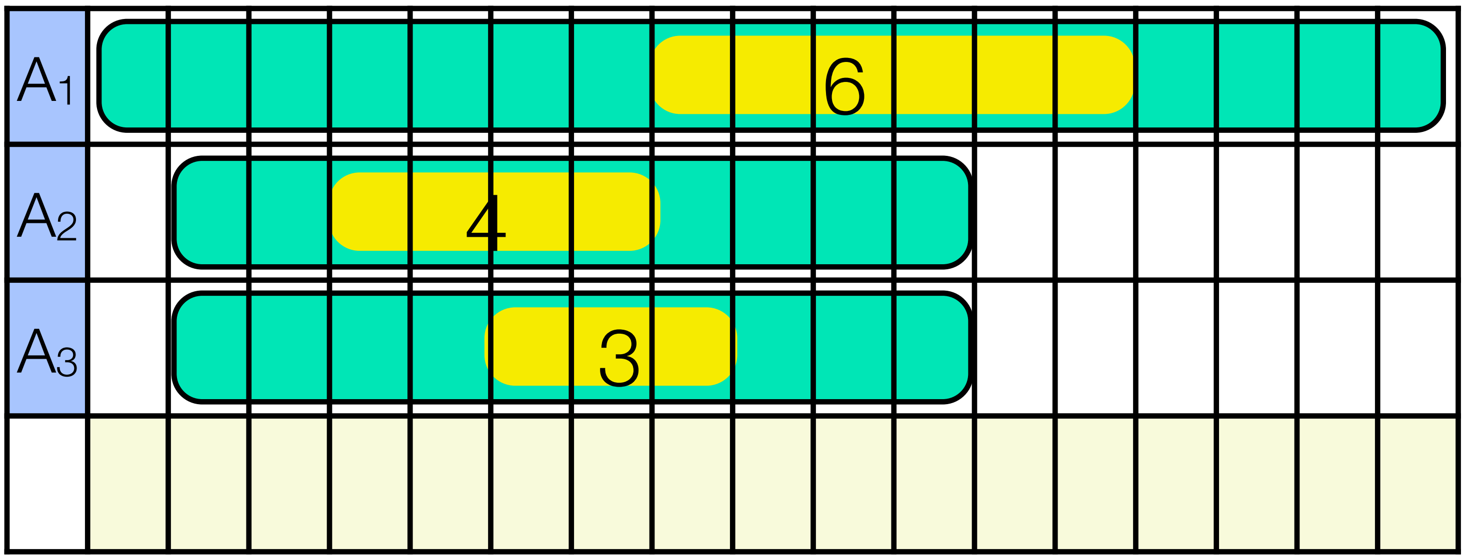
Disjunctive Constraint

- One-Machine Feasibility



Disjunctive Constraint

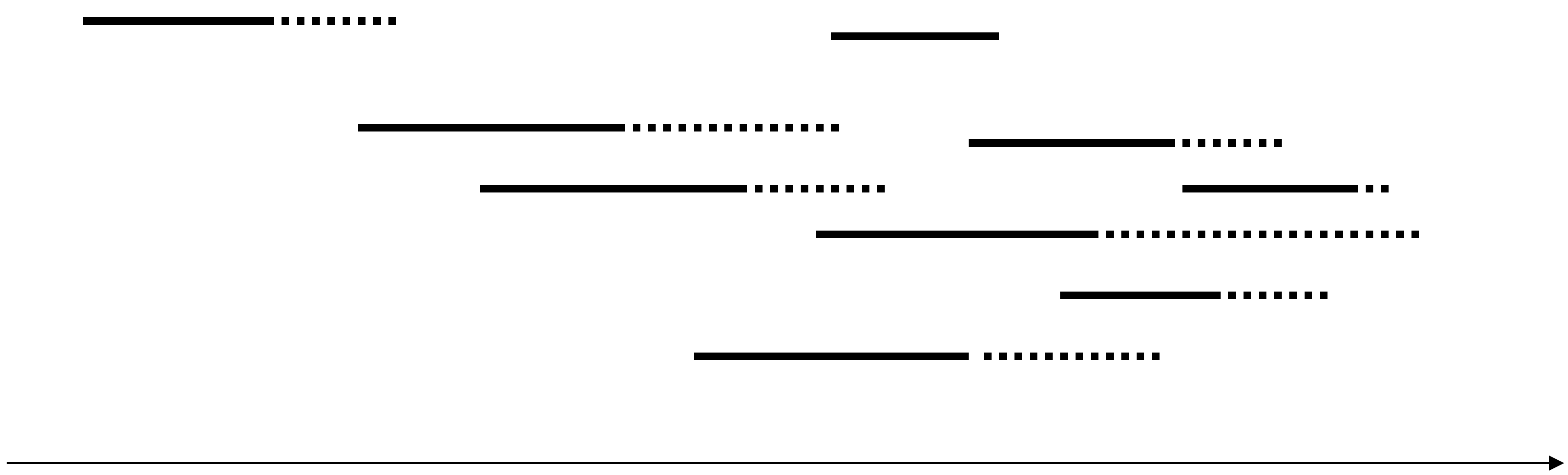
- One-Machine Feasibility



- One-Machine Feasibility is NP-Complete

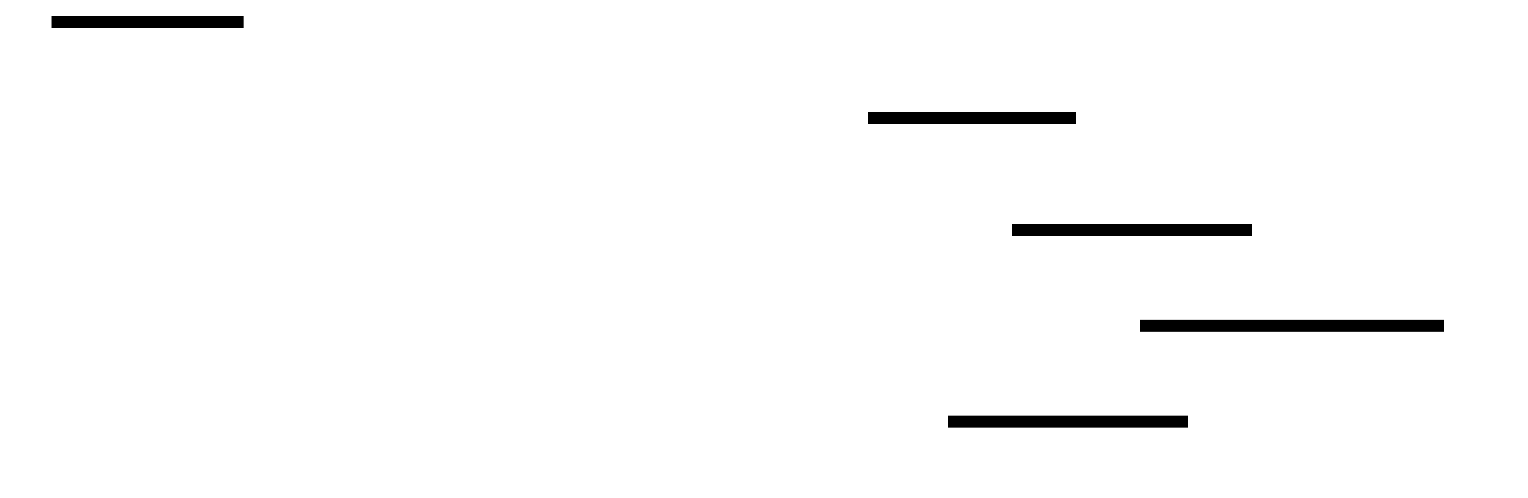
Disjunctive Feasibility

- Feasibility test: F(T)
  - R(T) + P(T) $\leq$ D(T)



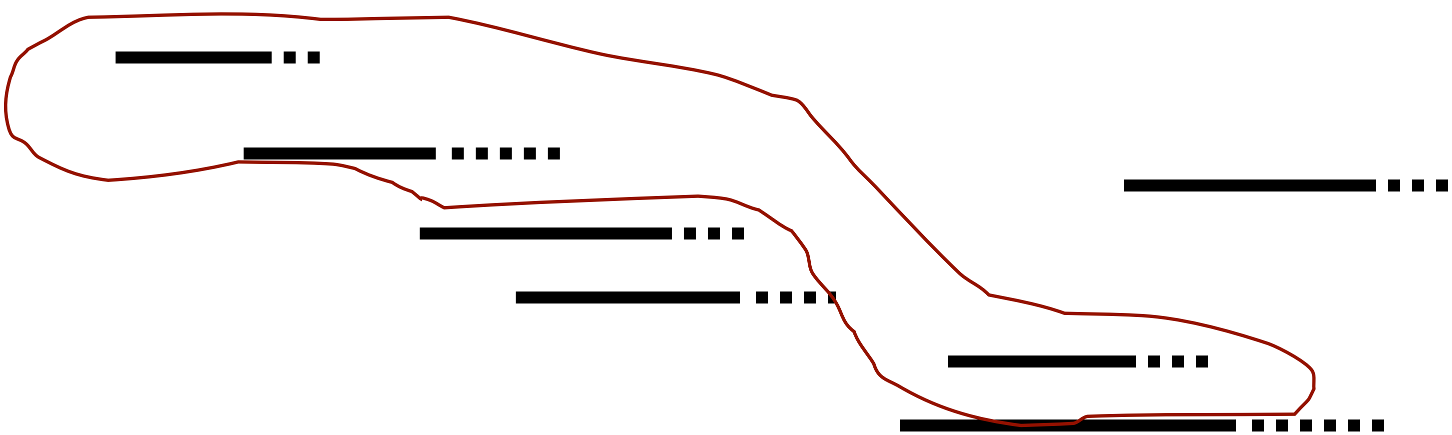
Disjunctive Feasibility

- Feasibility test: F(T)
  - R(T) + P(T) <= D(T)



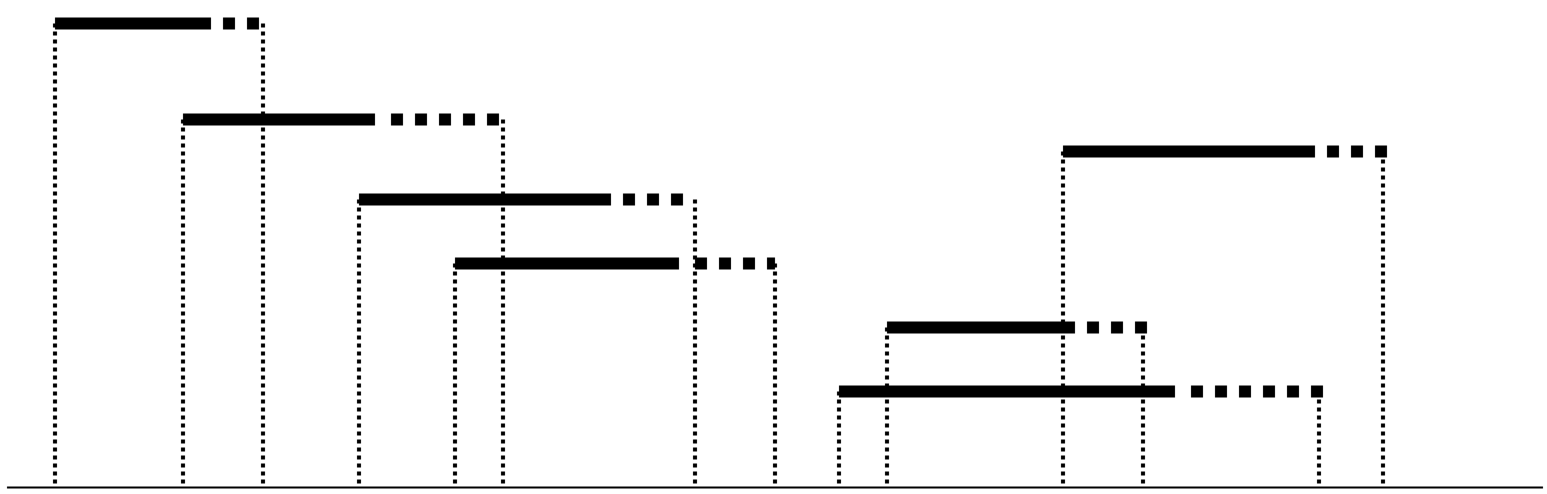
Disjunctive Feasibility

- A better feasibility test
  - apply F(S) to each subset S of T



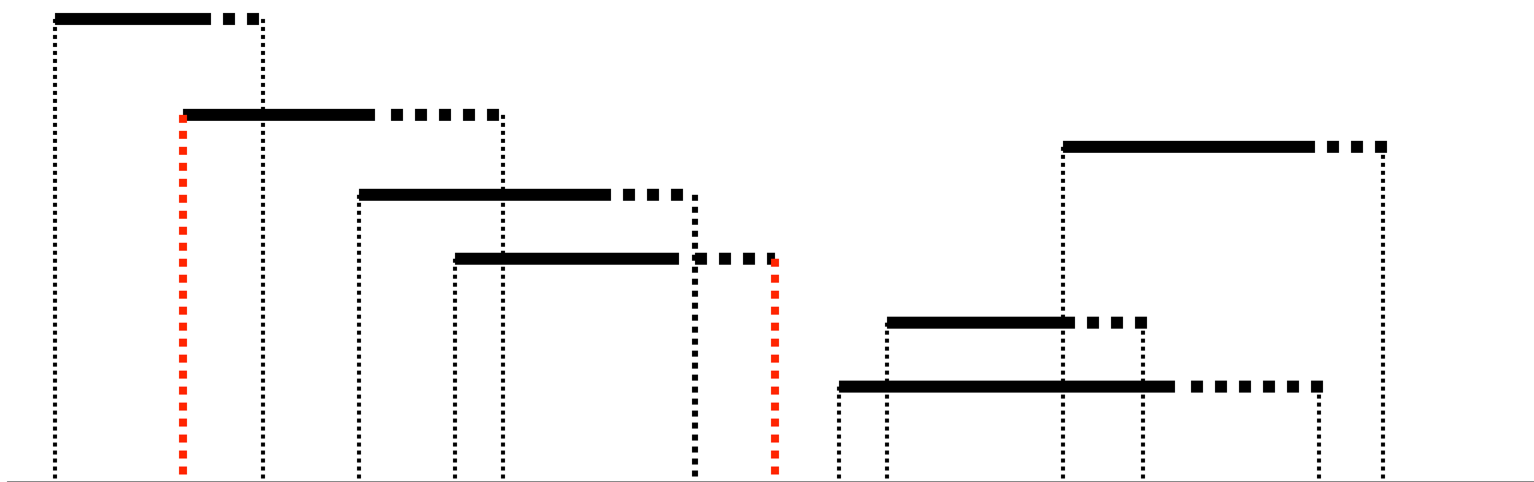
Disjunctive Feasibility

- A better feasibility test
  - apply F(S) to each subset S of T



Disjunctive Feasibility

- Task Intervals
  - $t[x,y] = \{ t \mid R(t) \geq R(x) \;\&\; D(t) \leq D(y) \}$
  - Intuition? Lifting

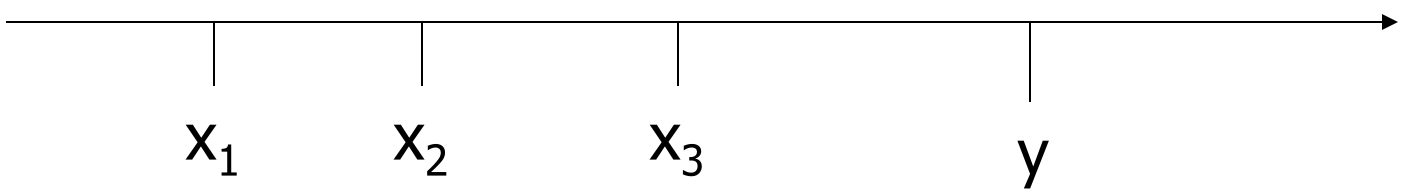


Disjunctive Feasibility

- Task Intervals
  - $t[x,y] = \{ t \mid R(t) \geq R(x) \ \& \ D(t) \leq D(y) \}$
- Feasibility
  - apply $F(t[x,y])$ for all x, y
- Complexity (?)
  - $O(|S|^3)$

Disjunctive Feasibility

- Can we do better?



Disjunctive Feasibility

- Compute all the x for a given y in linear time
- Scan backwards and accumulate the durations

```
d := 0;
forall(x in decreasing order of R(x)) {
  if D(x) <= D(y) then
       p := p + p(x)
       if (R(x) + p > R(y))
          failure;
```

Disjunctive Feasibility

- Can we do even better?
  - better than $O(|S|^2)$

- Yes, there exists an $O(n \log n)$ algorithm

Disjunctive Feasibility

- Can we do even better?
  - better than $O(|S|^2)$

- Jackson's heuristic to the one-machine problem
  - prefer the task with the tighest due date $L(t)$
  - greedy algorithm

Jackson’s heuristic

```
t := min(i in T) R(i);
S :={}; TS := T;
while (TS != {}) do
   P:= { j in TS | R(j) <= t };
   select i in P with minimal D(t);
   schedule i at time t;
   TS := TS \ { i};
   t := max(t + p(i),min(s in TS) R(s));
```

Jackson’s Heuristic

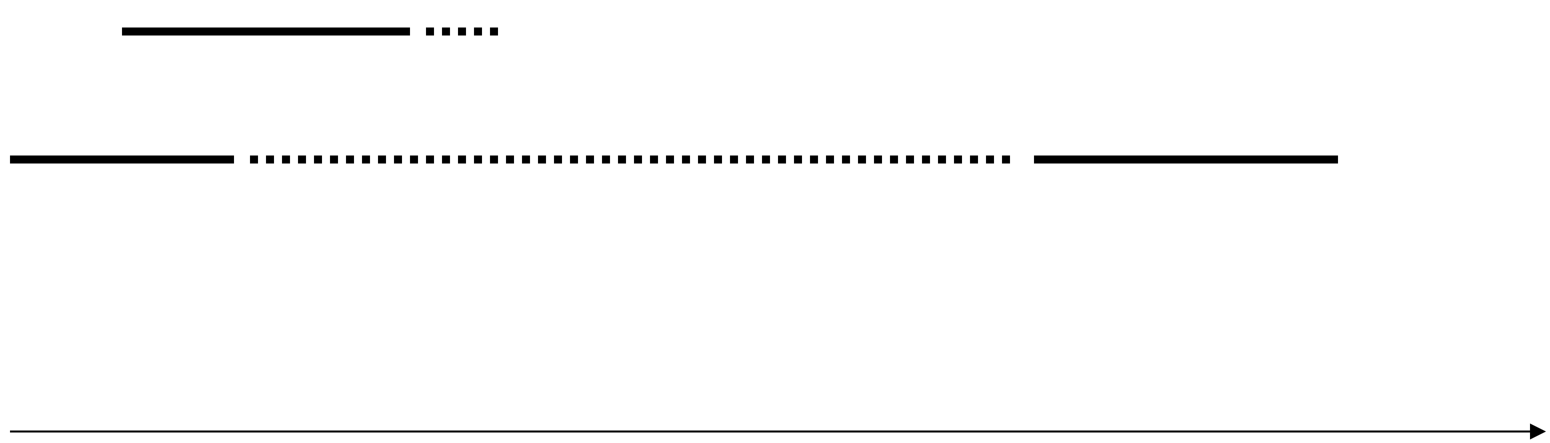
i	1	2	3	4	5	6	7	8	9	10
R	29	288	83	0	20	84	0	117	76	0
p	78	28	91	81	22	2	46	46	69	85
D	712	806	523	426	649	590	630	624	548	545

Jackson's Heuristic

- Start with 4
  - time becomes 81
  - available tasks: { 1, 5, 7, 9, 10}
- Continue with 10
  - time becomes 166,
  - available tasks: { 1, 3, 5, 6, 7, 8, 9 }
- Continue with 3
  - time is 257
  - available tasks: { 1, 5, 6, 7, 8, 9 }
- Final solution: 4, 10, 3, 9, 6, 8, 7, 5, 1, 2
- Complexity?
- Limitation?

Jackson’s heuristic

Too greedy !



Preemptive Scheduling

- Basic idea
  - Take Jackson's heuristic
  - apply to the preemptive problem
- What is the preemptive problem?
  - tasks can be interrupted
- Relationship between preemptive and non-preemptive problem?
  - relax, relax,

Preemptive scheduling

```
t := min(i in T) R(i);
S :={}; TS := T;
while (TS != {}) do
   P:= { j in TS | R(j) <= t };
   select i in P with minimal D(t);
   delta := min(p(i),min(j in TS\P)R(j)-t);
   p(i) := p(i) - delta;
   [schedule i at time t for delta units]
   if (p(i) == 0)
      TS := TS \ { i};
   t := max(t + delta,min(s in TS) R(s);
```

i	1	2	3	4	5	6	7	8	9	10
E	29	288	83	0	20	84	0	117	76	0
d	78	28	91	81	22	2	46	46	69	85
L	712	806	523	426	649	590	630	624	548	545

Jackson's Preemptive Schedule

- Time 0; P= {4,7,10}
  - schedule 4 for 20 units
- Time 20; P={4,5,7,10}
  - schedule 4 for 9 units
- Time 29; P={1,4,5,7,10}
  - schedule 4 for 47 units
- Time 76; P= {1,4,5,7,9,10}
  - schedule 4 for 5 units
- Time 81; P={1,4,5,9,10}
  - schedule 10 for 2 units
- Time 83; P={1,4,5,7,9,10}
  - schedule 3 for 1 unit

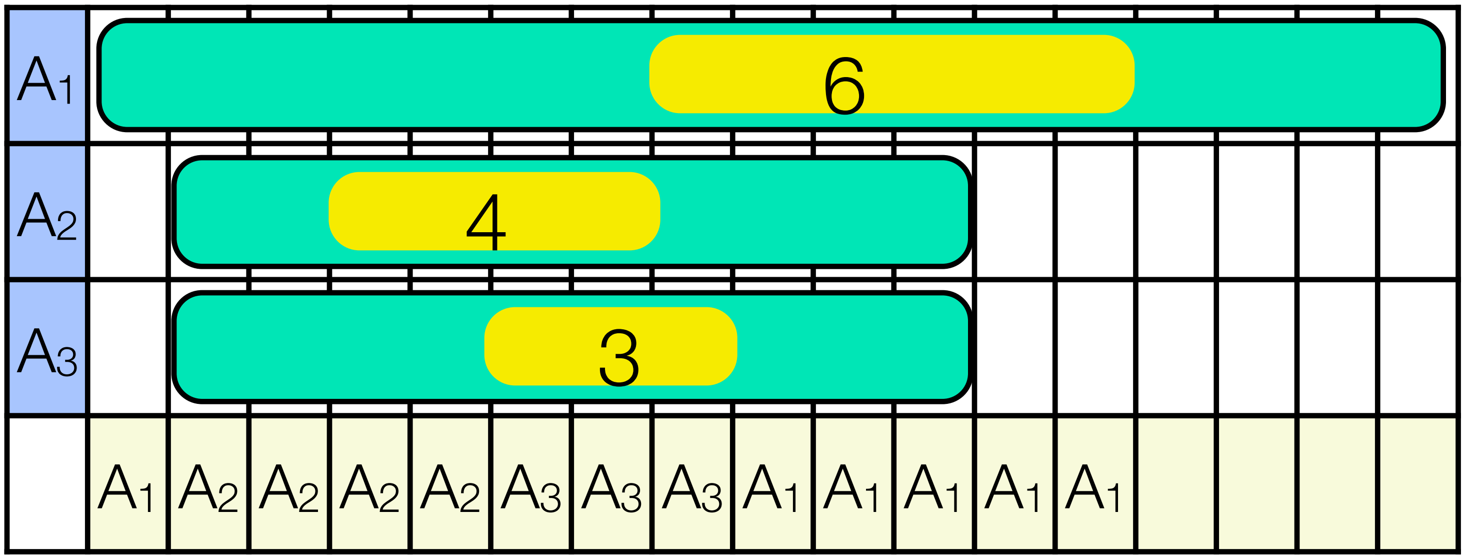
Jackson's Preemptive Schedule

- Why do we care?
  - O(n log n)
- Seriously, why do we care?
  - equivalent to the task interval feasibility
- The test
  - forall(s in 2^S) F(S)

is equivalent to preemptive one-machine scheduling

Disjunctive Constraint

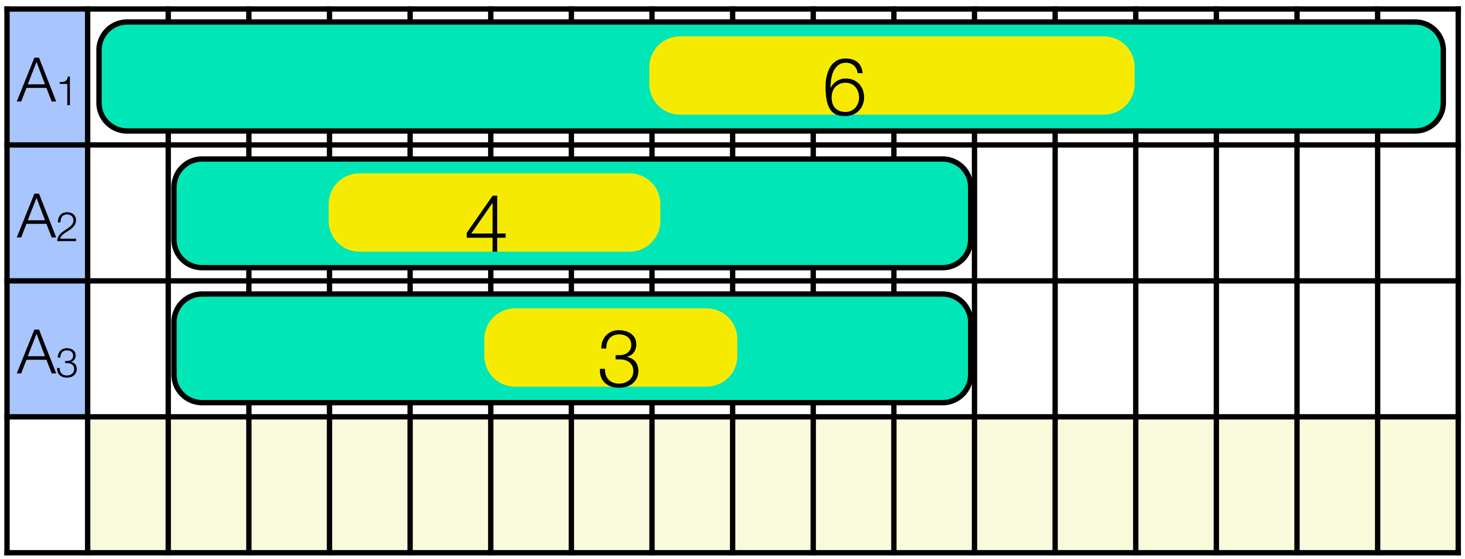
- Relax! One-Machine Preemptive Feasibility



- One-Machine Preemptive Feasibility can be computed in O(n log n) time.

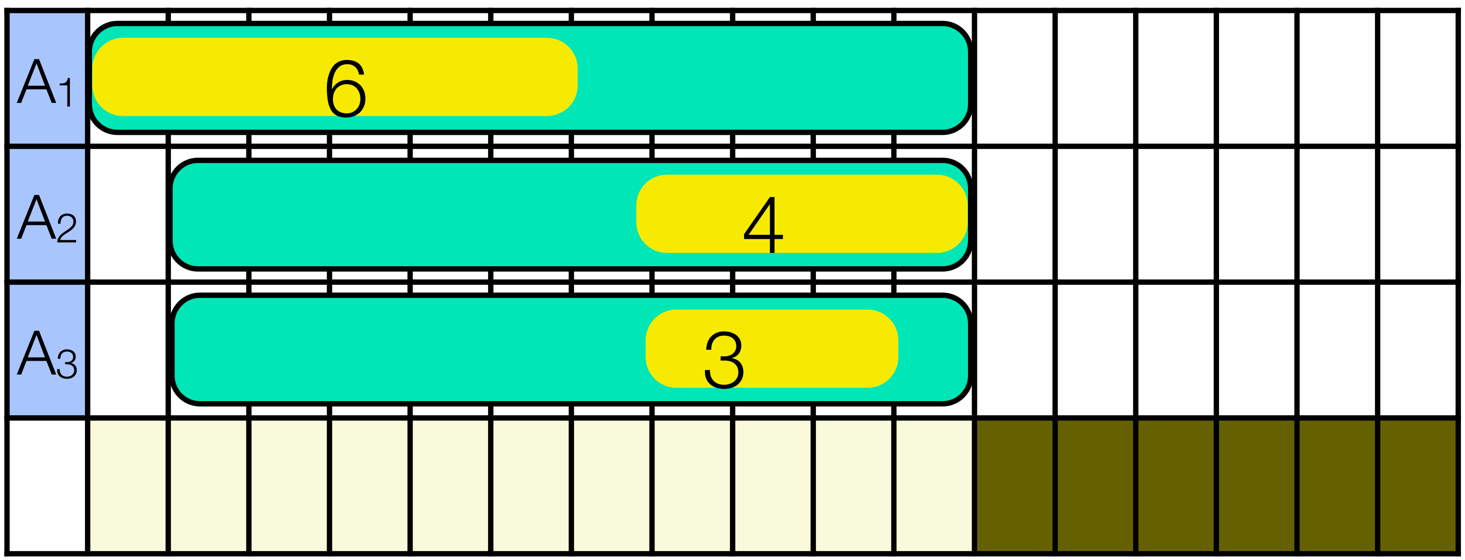
Edge Finder

- Pruning: Must A_1 start after A_2 and A_3?



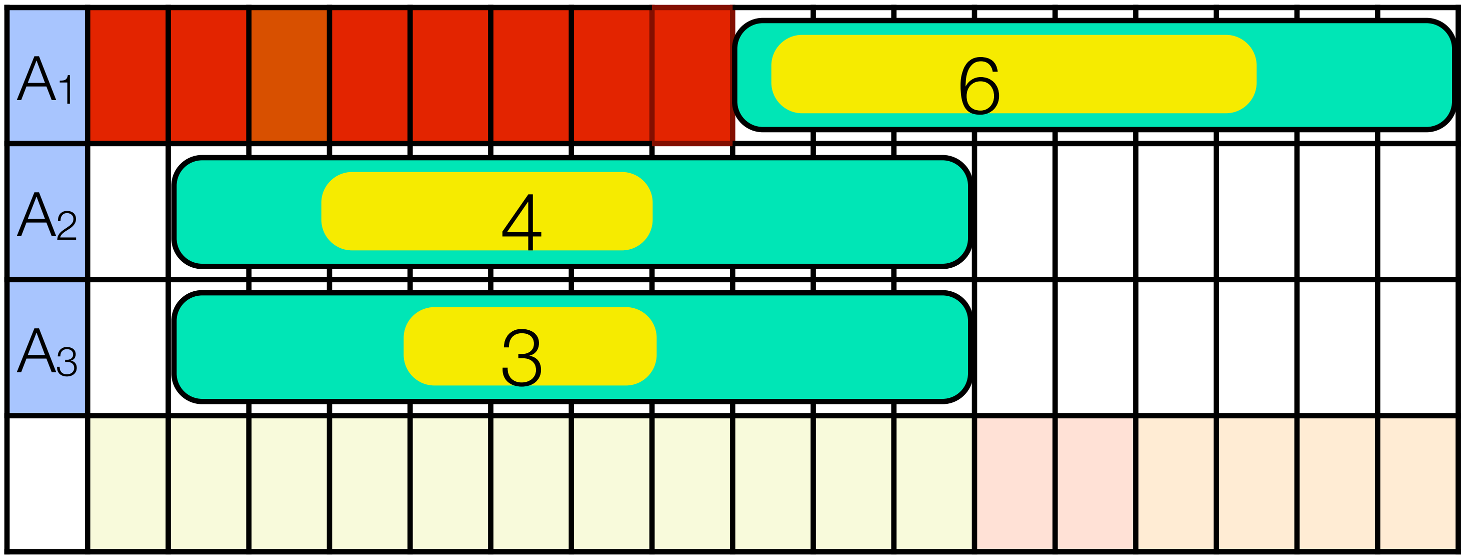
Edge Finder

- Pruning: Must A_1 start after A_2 and A_3?



Edge Finder

- Pruning: A_1 must start after A_2 and A_3



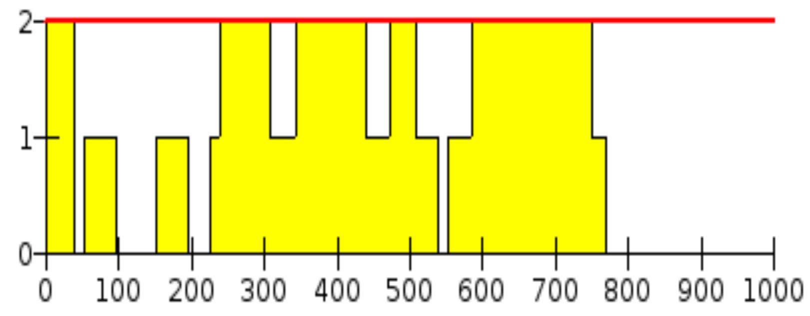
The One Machine Problem

- Input
  - all the tasks executing on the machine
- Each task i
  - release date: r_i
  - due date: d_i
  - processing time: p_i
  - capacity: c_i
  - energy: $e_i = c_i \times p_i$

Cumulative Resource

- The sum of the capacities of the tasks executing at time t cannot exceed the capacity of the resource

$$\forall t \sum_{i:s_i \leq t \leq s_i+p_i} c_i \leq C$$



Edge Finder

- Notations

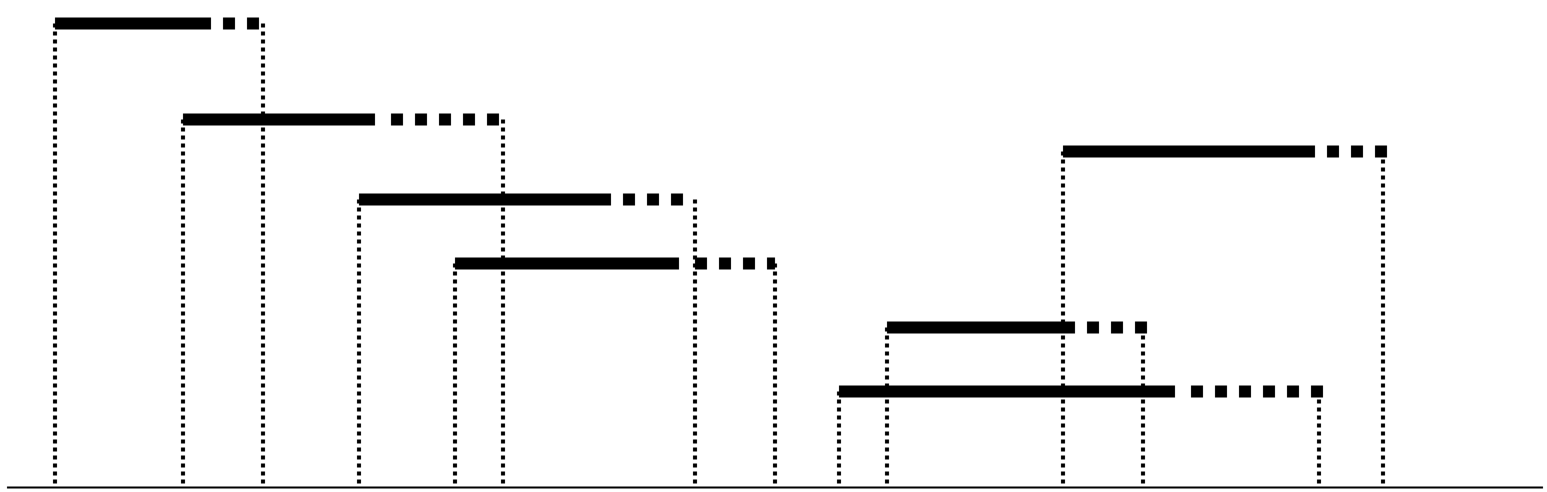
$$r_\Omega = \min_{i \in \Omega} r_i \qquad d_\Omega = \max_{i \in \Omega} d_i$$

$$p_\Omega = \sum_{i \in \Omega} p_i \qquad e_\Omega = \sum_{i \in \Omega} e_i$$

One Machine Feasibility

- EF-Feasibility

$$\forall \Omega \subseteq T : r_\Omega + e_\Omega \leq d_\Omega$$



Edge Finder

- Pruning
  - Determine if a task must finish after a set of tasks
- Two parts
  - condition
    - same for disjunctive and cumulative problems
  - update of the release date
    - different for disjunctive and cumulative problems

Edge Finder: Condition

if

$$r_{\Omega \cup \{i\}} + e_{\Omega \cup \{i\}} > d_\Omega$$

then

$$i \text{ must terminate after } \Omega$$

Disjunctive Edge Finder

if

$$\alpha(\Omega, i) \equiv r_{\Omega \cup \{i\}} + e_{\Omega \cup \{i\}} > d_\Omega$$

then

$$s_i \geq r_\Omega + p_\Omega$$

Disjunctive Edge Finder

if

$$r_{\Omega \cup \{i\}} + e_{\Omega \cup \{i\}} > d_{\Omega}$$

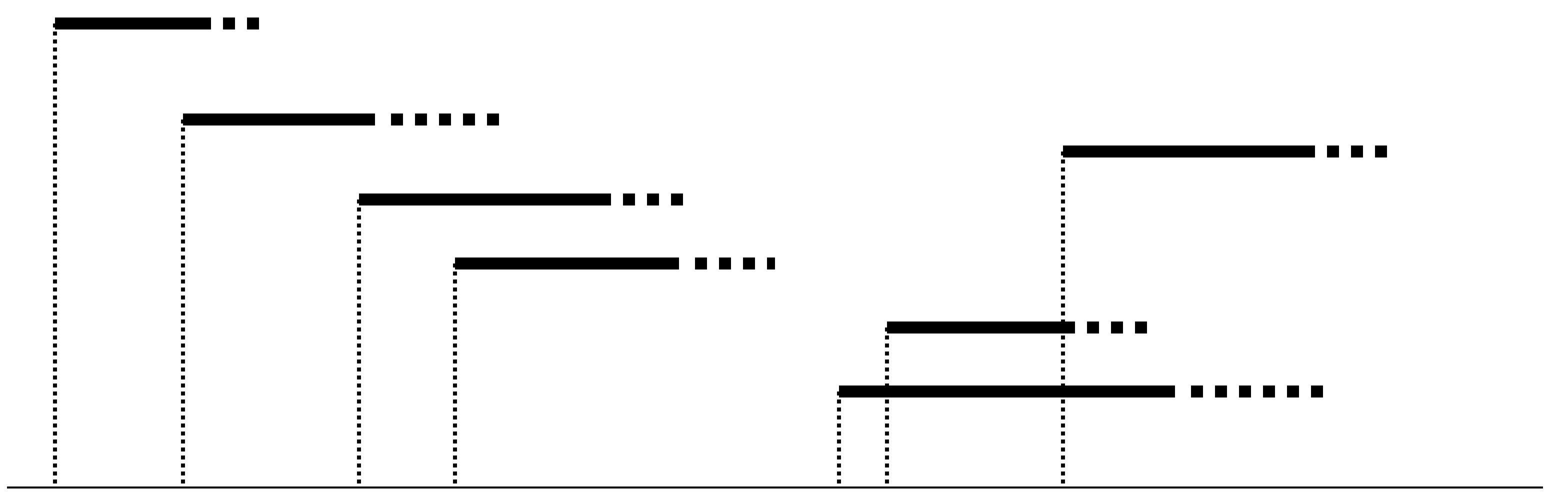
then

$$s_i \geq \max_{\Theta \subseteq \Omega} r_{\Theta} + p_{\Theta}$$

Disjunctive Edge Finder

- Update

$$s_i \geq \max_{\Theta \subseteq \Omega} r_\Theta + p_\Theta$$



Disjunctive Edge Finder

- Update

$$s_i \geq \max_{\Theta \subseteq \Omega} r_\Theta + p_\Theta$$

- Dominance rule

$$s_i \geq \max_{\substack{\Theta \subseteq \Omega \\ d_\Theta = d_\Omega}} r_\Theta + p_\Theta$$

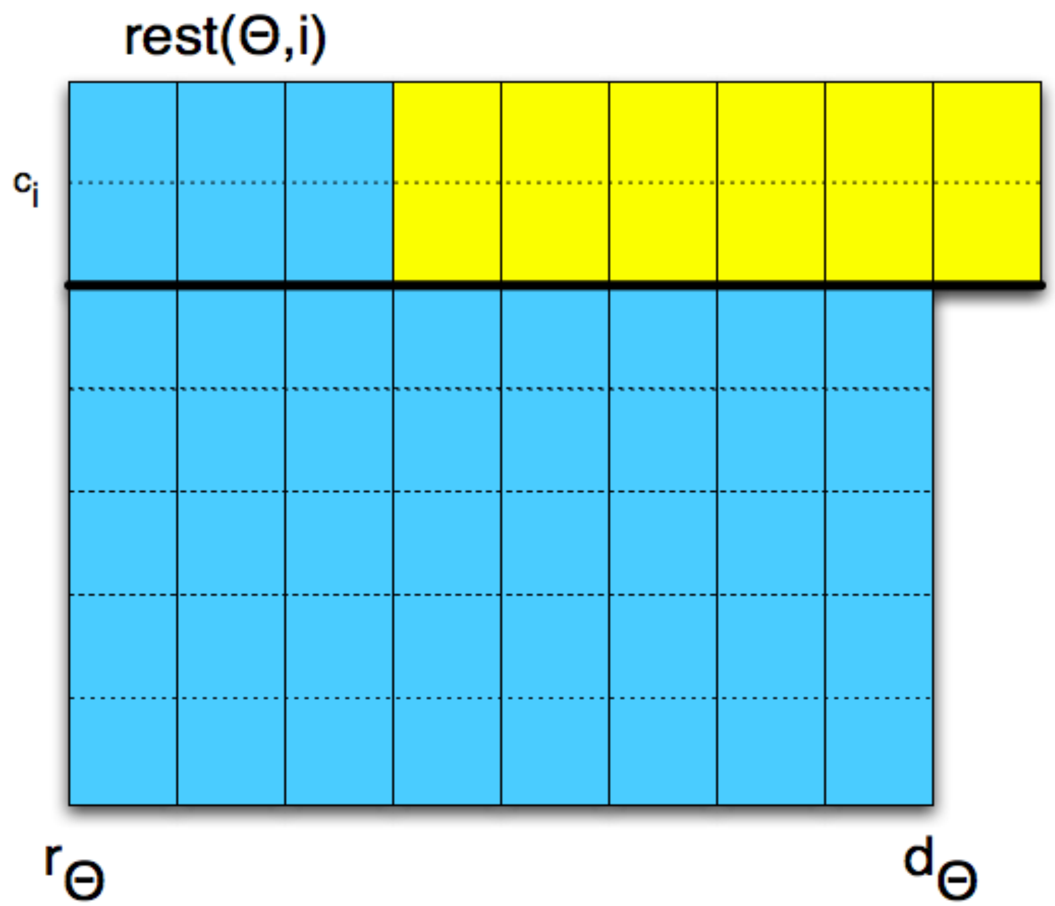
Cumulative Edge Finder

$$\text{if} \quad r_{\Omega \cup \{i\}} + e_{\Omega \cup \{i\}} > d_\Omega \ \text{then}$$

$$s_i \geq \max_{\substack{\Theta \subseteq \Omega \\ rest(\Theta, c_i) > 0}} r_\Theta + \left\lceil \frac{1}{c_i} rest(\Theta, c_i) \right\rceil$$

$$rest(\Theta, c) = e_\Theta - (C - c_i)(d_\Theta - r_\Theta)$$

Cumulative Edge Finder



Cumulative Edge Finder

- So far we only consider one set

$$s_i \geq \max_{\substack{\Omega \subseteq T \\ i \notin \Omega \\ \alpha(\Omega, i)}} \max_{\substack{\Theta \subseteq \Omega \\ rest(\Theta, c_i) > 0}} r_\Theta + \left\lceil \frac{1}{c_i} rest(\Theta, c_i) \right\rceil$$

Cumulative Edge Finder

- After applying a couple of valid dominances

$$s_i \geq \max_{\substack{L,U \in T \\ \alpha(\Omega_L^U, i) \\ d_U < d_i \\ r_L = r_{\Omega_L^U \cup \{i\}}}} \max_{\substack{l,u \in T \\ d_{\Omega_l^u} \leq d_U \\ rest(\Omega_l^u, c_i) > 0}} r_l + \left\lceil \frac{1}{c_i} rest(\Omega_l^u, c_i) \right\rceil$$

- A trivial $O(n^6)$ algorithm

Cumulative Edge Finder

- Focus on the innermost loop

$$
\max_{\substack{L,U \in T \\ \alpha(\Omega_L^U, i) \\ d_U < d_i \\ r_L = r_{\Omega_L^U \cup \{i\}}}} \quad \max_{\substack{l,u \in T \\ d_{\Omega_l^u} \le d_U \\ rest(\Omega_l^u, c_i) > 0}} r_l + \left\lceil \frac{1}{c_i} rest(\Omega_l^u, c_i) \right\rceil
$$

- It does not depend on Ω_L^U except for d_U

Precomputation

- For each capacity c and U in T, compute

$$rt[c,U] = \max_{\substack{l,u \in T \\ d_{\Omega_l^u} \leq d_U \\ rest(\Omega_l^u, c) > 0}} r_l + \left\lceil \frac{1}{c} rest(\Omega_l^u, c) \right\rceil$$

Precomputation

- For each capacity c and l,u in T, compute

$$rt[c,l,u] = \max_{\substack{l \leq x \;\&\; y \leq u \\ rest(\Omega_x^y, c) > 0}} r_l + \left\lceil \frac{1}{c} rest(\Omega_x^y, c) \right\rceil$$

- How? Dynamic programming

RT[c,x,y]

		y[1]	y[2]						y[n]
x[1]	$-\infty$								
x[2]	$-\infty$								
x[3]	$-\infty$								
	$-\infty$								
	$-\infty$								
	$-\infty$								
	$-\infty$								
x[n]	$-\infty$								
	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$	$-\infty$

Precomputation

$$RT[c, x-1, y+1] = max \begin{cases} RT[c, x, y+1] \\ RT[c, x-1, y] \\ r_{X[x-1]} + \left\lceil \frac{1}{c} f(rest(\Omega_{x-1}^{y+1}, c)) \right\rceil \end{cases}$$

where $$f(x) = \begin{cases} x & \text{if } x > 0 \\ -\infty & \text{otherwise} \end{cases}$$

$O(n^2 k)$ where k is the number of distinct capacities

Cumulative Edge Finder

- After applying a couple of valid dominances

$$s_i \geq \max_{\substack{L,U \in T \\ \alpha(\Omega_L^U, i) \\ d_U < d_i \\ r_L = r_{\Omega_L^U \cup \{i\}}}} \max_{\substack{l,u \in T \\ d_{\Omega_l^u} \leq d_U \\ rest(\Omega_l^u, c_i) > 0}} r_l + \left\lceil \frac{1}{c_i} rest(\Omega_l^u, c_i) \right\rceil$$

- A trivial $O(n^6)$ algorithm

The Cumulative Edge Finder

```
for x = 1 to n do NR[x] = r[x];
for y = 1 to n-1 do
   E = 0;
   for x = n downto 1 do
      if d[x] < d[y] then E = E + e[x];
      E[x] = E;
   for x = 1 to n do
      for i = x to n do
         if E[x] + e[i] > C (d[y] - r[x])
            if d[i] > d[y] then
               NR[i] = max(NR[i],R[c[i],y]);
```

$$O(n^3)$$

The Cumulative Edge Finder

```
for x = 1 to n do NR[x] = r[x];
for y = 1 to n-1 do
    E = 0;
    for x = n downto 1 do
        if d[x] < d[y] then E = E + e[x];
        E[x] = E;
    for x = 1 to n do
        for i = x to n do
            if E[x] + e[i] > C(d[y] - r[x])
                if d[i] > d[y] then
                    NR[i] = max(NR[i],RT[c[i],y]);
```

Look, ma!
no x

The Cumulative Edge Finder

```
for x = 1 to n do NR[x] = r[x];
for y = 1 to n-1 do
   E = 0;
   for x = n downto 1 do
      if d[x] < d[y] then E = E + e[x];
      E[x] = E;
   CEF = - infinity;
   for i = 1 to n do
      CEF = max(CEF,E[i]-C(d[y]-r[i]);
      if d[i] > d[y] & CEF + e[i] > 0 then
         NR[i] = max(NR[i],RT[c[i],y);
```

Look, ma!
no x anywhere

$$O(n^2)$$