DM204 – Spring 2011

Scheduling, Timetabling and Routing

Lecture 3

Single Machine Problems

Marco Chiarandini

Department of Mathematics & Computer Science
University of Southern Denmark

Outline

1. Branch and Bound

2. IP Models

3. Dynamic Programming

4. Local Search

Summary

$1 \mid\mid \sum w_j C_j$: weighted shortest processing time first is optimal

$1 \mid\mid \sum_j U_j$: Moore's algorithm

$1 \mid prec \mid L_{max}$: Lawler's algorithm, backward dynamic programming in $O(n^2)$ [Lawler, 1973]

$1 \mid\mid \sum h_j(C_j)$: dynamic programming in $O(2^n)$

$1 \mid r_j, (prec) \mid L_{max}$: branch and bound

$1 \mid\mid \sum w_j T_j$: local search and dynasearch

$1 \mid\mid \sum w_j T_j$: IP formulations, column generation approaches

$1 \mid s_{jk} \mid C_{max}$: in the special case, Gilmore and Gomory algorithm optimal in $O(n^2)$

Multicriteria

Outline

1. Branch and Bound
2. IP Models
3. Dynamic Programming
4. Local Search

$1 \mid r_j \mid L_{max}$

[Maximum lateness with release dates]

- Strongly NP-hard (reduction from 3-partition)
- might have optimal schedule which is not non-delay

$1 \mid r_j \mid L_{max}$

[Maximum lateness with release dates]

- Strongly NP-hard (reduction from 3-partition)
- might have optimal schedule which is not non-delay
- Branch and bound algorithm (valid also for $1 \mid r_j, prec \mid L_{max}$)
  - **Branching**:
    schedule from the beginning (level k, $n!/(k-1)!$ nodes)
    elimination criterion: do not consider job j_k if:

    $$r_j > \min_{l \in J} \{\max(t, r_l) + p_l\} \qquad J \text{ jobs to schedule, } t \text{ current time}$$

  - **Lower bounding**: relaxation to preemptive case for which EDD is optimal

Branch and Bound

S root of the branching tree

```
LIST := {S};
U:=value of some heuristic solution;
current_best := heuristic solution;
while LIST ≠ ∅
   Choose a branching node k from LIST;
   Remove k from LIST;
   Generate children child(i), i = 1, ..., n_k, and calculate corresponding lower
          bounds LB_i;
   for i:=1 to n_k
      if LB_i < U then
         if child(i) consists of a single solution then
            U:=LB_i;
            current_best:=solution corresponding to child(i)
         else add child(i) to LIST
```

Branch and Bound

Branch and bound *vs* backtracking

- $=$ a state space tree is used to solve a problem.
- $\neq$ branch and bound does not limit us to any particular way of traversing the tree (backtracking is depth-first)
- $\neq$ branch and bound is used only for optimization problems.

Branch and bound *vs* A*

- $=$ In A* the admissible heuristic mimics bounding
- $\neq$ In A* there is no branching. It is a search algorithm.
- $\neq$ A* is best first

Branch and Bound

[Jens Clausen (1999). Branch and Bound Algorithms - Principles and Examples.]

- Eager Strategy:
  1. select a node
  2. branch
  3. for each subproblem compute bounds and compare with incumbent solution
  4. discard or store nodes together with their bounds

  (Bounds are calculated as soon as nodes are available)

- Lazy Strategy:
  1. select a node
  2. compute bound
  3. branch
  4. store the new nodes together with the bound of the father node

(often used when selection criterion for next node is max depth)

Components

1. Initial feasible solution (heuristic) – might be crucial!
2. Bounding function
3. Strategy for selecting
4. Branching
5. Fathoming (dominance test)

Bounding

$$\min_{s \in P} g(s) \leq \left\{ \begin{array}{c} \min_{s \in P} f(s) \\ \min_{s \in S} g(s) \end{array} \right\} \leq \min_{s \in S} f(s)$$

P: candidate solutions; $S \subseteq P$ feasible solutions

- relaxation: $\min_{s \in P} f(s)$
- solve (to optimality) in P but with g

Bounding

$$\min_{s \in P} g(s) \leq \left\{ \begin{array}{l} \min_{s \in P} f(s) \\ \min_{s \in S} g(s) \end{array} \right\} \leq \min_{s \in S} f(s)$$

P: candidate solutions; $S \subseteq P$ feasible solutions

- relaxation: $\min_{s \in P} f(s)$
- solve (to optimality) in P but with g
- Lagrangian relaxation combines the two

Bounding

$$\min_{s \in P} g(s) \leq \left\{ \begin{array}{c} \min_{s \in P} f(s) \\ \min_{s \in S} g(s) \end{array} \right\} \leq \min_{s \in S} f(s)$$

P: candidate solutions; $S \subseteq P$ feasible solutions

- relaxation: $\min_{s \in P} f(s)$
- solve (to optimality) in P but with g
- Lagrangian relaxation combines the two
- should be polytime and strong (trade off)

Strategy for selecting next subproblem

- best first
  (combined with eager strategy but also with lazy)
- breadth first
  (memory problems)
- depth first
  works on recursive updates (hence good for memory)
  but might compute a large part of the tree which is far from optimal

Strategy for selecting next subproblem

- best first
  (combined with eager strategy but also with lazy)
- breadth first
  (memory problems)
- depth first
  works on recursive updates (hence good for memory)
  but might compute a large part of the tree which is far from optimal
  (enhanced by alternating search in lowest and largest bounds combined with branching on the node with the largest difference in bound between the children)
  (it seems to perform best)

Branching

- dichotomic
- polytomic

Branching

- dichotomic
- polytomic

Overall guidelines

- finding good initial solutions is important
- if initial solution is close to optimum then the selection strategy makes little difference
- Parallel B&B: distributed control or a combination are better than centralized control
- parallelization might be used also to compute bounds if few nodes alive
- parallelization with static work load distribution is appealing with large search trees

$1 | | \sum w_j T_j$

- **Branching:**
  - work backward in time
  - elimination criterion:
    if $p_j \leq p_k$ and $d_j \leq d_k$ and $w_j \geq w_k$ then there is an optimal schedule with j before k

$1 \mid \mid \sum w_j T_j$

- **Branching:**
  - work backward in time
  - elimination criterion:
    if $p_j \leq p_k$ and $d_j \leq d_k$ and $w_j \geq w_k$ then there is an optimal schedule with j before k

- **Lower Bounding:**
  relaxation to preemptive case
  transportation problem

$$\min \sum_{j=1}^{n} \sum_{t=1}^{C_{max}} c_{jt} x_{jt}$$

$$\text{s.t.} \sum_{t=1}^{C_{max}} x_{jt} = p_j, \qquad \forall j = 1, \ldots, n$$

$$\sum_{j=1}^{n} x_{jt} \leq 1, \qquad \forall t = 1, \ldots, C_{max}$$

$$x_{jt} \geq 0 \qquad \forall j = 1, \ldots, n; \ t = 1, \ldots, C_{max}$$

[Pan and Shi, 2007]'s lower bounding through time indexed
Stronger but computationally more expensive

$$\min \sum_{j=1}^{n} \sum_{t=1}^{T-1} c_{jt} y_{jt}$$

s.t.

$$\sum_{t=1}^{T-p_j} c_{jt} \leq h_j(t+p_j)$$

$$\sum_{t=1}^{T-p_j} y_{jt} = 1, \qquad \forall j = 1, \ldots, n$$

$$\sum_{j=1}^{n} \sum_{s=t-p_j+1}^{t} y_{jt} \leq 1, \qquad \forall t = 1, \ldots, C_{max}$$

$$y_{jt} \geq 0 \qquad \forall j = 1, \ldots, n; \;\; t = 1, \ldots, C_{max}$$

Complexity resume

Single machine, single criterion problems $\mathbf{1}\,|\,|\,\gamma$:

C_{max}	$\mathcal{P}$
T_{max}	$\mathcal{P}$
L_{max}	$\mathcal{P}$
h_{max}	$\mathcal{P}$
$\sum C_j$	$\mathcal{P}$
$\sum w_j C_j$	$\mathcal{P}$
$\sum U$	$\mathcal{P}$
$\sum w_j U_j$	weakly $\mathcal{NP}$-hard
$\sum T$	weakly $\mathcal{NP}$-hard
$\sum w_j T_j$	strongly $\mathcal{NP}$-hard
$\sum h_j(C_j)$	strongly $\mathcal{NP}$-hard

Outline

1. Branch and Bound
2. IP Models
3. Dynamic Programming
4. Local Search

IP Models
Sequencing variables

$1|prec|\sum w_j C_j$

Sequencing (linear ordering) variables

$$
\begin{aligned}
\min \sum_{j=1}^{n} \sum_{k=1}^{n} w_j p_k x_{kj} + \sum_{j=1}^{n} w_j p_j \\
\text{s.t. } & x_{kj} + x_{jl} + x_{lk} \geq 1 \qquad j, k, l = 1, \ldots, n j \neq k, k \neq l \\
& x_{kj} + x_{jk} = 1 \qquad \forall j, k = 1, \ldots, n, j \neq k \\
& x_{jk} \in \{0, 1\} \qquad j, k = 1, \ldots, n \\
& x_{jj} = 0 \qquad \forall j = 1, \ldots, n
\end{aligned}
$$

IP Models
Completion time

$1|prec|C_{max}$

Completion time variables $\in \mathbb{R}$ and job precedences $\in \mathbb{B}$ for disjunctive constraints

$$\begin{aligned}
\min \sum_{j=1}^{n} w_j z_j & \\
\text{s.t. } & z_k - z_j \geq p_k \quad \text{for } j \to k \in A \\
& z_j \geq p_j, \quad \text{for } j = 1, \ldots, n \\
& z_k - z_j \geq p_k \quad \text{or} \quad z_j - z_k \geq p_j, \ \text{for } (i,j) \in I \\
& z_j \in \mathbf{R}, \quad j = 1, \ldots, n
\end{aligned}$$

IP Models
Time indexed variables

$1||\sum h_j(C_j)$

Time indexed variables

$$\min \sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} h_j(t+p_j)x_{jt}$$

$$\text{s.t.} \quad \sum_{t=1}^{T-p_j+1} x_{jt} = 1, \qquad \text{for all } j = 1,\ldots,n$$

$$\sum_{j=1}^{n} \sum_{s=\max\{0,t-p_j+1\}}^{t} x_{js} \leq 1, \qquad \text{for each } t = 1,\ldots,T$$

$$x_{jt} \in \{0,1\}, \qquad \text{for each } j = 1,\ldots,n;\; t = 1,\ldots,T$$

IP Models
Time indexed variables

$1||\sum h_j(C_j)$

Time indexed variables

$$
\begin{aligned}
\min \sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} & h_j(t+p_j)x_{jt} \\
\text{s.t.} \quad \sum_{t=1}^{T-p_j+1} & x_{jt} = 1, \quad \text{for all } j=1,\ldots,n \\
\sum_{j=1}^{n} \sum_{s=\max\{0,t-p_j+1\}}^{t} & x_{js} \leq 1, \quad \text{for each } t=1,\ldots,T \\
& x_{jt} \in \{0,1\}, \quad \text{for each } j=1,\ldots,n;\ t=1,\ldots,T
\end{aligned}
$$

+ The LR of this formulation gives better bounds than the two preceding
+ Flexible with respect to objective function
− Pseudo-polynomial number of variables

$$\begin{aligned} \max\ & c^T x \\ \text{s.t. } & Ax \le b \\ & Dx \le d \\ & x \in \mathbb{Z}_+^n \end{aligned}$$

$$\begin{aligned} \max\ & c^T x \\ \text{s.t. } & Ax \le b \\ & x \in P \end{aligned} \tag{IP}$$

polytope $P = \{x \in \mathbb{Z}^n : Dx \le d\}$

$$\begin{aligned} \max\ & c^T x \\ \text{s.t. }\ & Ax \le b \\ & Dx \le d \\ & x \in \mathbb{Z}_+^n \end{aligned}$$

$$\begin{aligned} \max\ & c^T x \\ \text{s.t. }\ & Ax \le b \\ & x \in P \end{aligned} \tag{IP}$$

polytope $P = \{x \in \mathbb{Z}^n : Dx \le d\}$

Assuming that P is bounded and has a finite number of points $\{x^s\}, s \in Q$ it can be represented by its extreme points $x^1, \dots, x^k$:

$$x^s = \sum_{k=1}^{K} \lambda_k x^k, \text{ with } \sum_{k=1}^{K} \lambda_k = 1, \lambda_k \ge 0$$

$$\begin{aligned} \max\ & c^T x \\ \text{s.t.}\ & Ax \le b \\ & Dx \le d \\ & x \in \mathbb{Z}^n_+ \end{aligned}$$

$$\begin{aligned} \max\ & c^T x \\ \text{s.t.}\ & Ax \le b \\ & x \in P \end{aligned} \tag{IP}$$

polytope $P = \{x \in \mathbb{Z}^n : Dx \le d\}$

Assuming that P is bounded and has a finite number of points $\{x^s\}, s \in Q$ it can be represented by its extreme points $x^1, \ldots, x^k$:

$$x^s = \sum_{k=1}^{K} \lambda_k x^k, \text{ with } \sum_{k=1}^{K} \lambda_k = 1, \lambda_k \ge 0$$

substituting in (IP) leads to DW master problem:

$$\begin{aligned} \max\ & \sum_k (cx^k)\lambda_k \\ \text{s.t.}\ & \sum_k (Ax^k)\lambda_k \le b \\ & \sum_{k=1}^{K} \lambda_k = 1 \\ & \lambda_k \ge 0 \end{aligned} \tag{MP}$$

Dantzig-Wolfe decomposition

Reformulation:

$$\min \sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} h_j(t+p_j)x_{jt}$$

$$\text{s.t.} \quad \sum_{t=1}^{T-p_j+1} x_{jt} = 1, \quad \text{for all } j = 1, \ldots, n$$

$$x_{jt} \in X \quad \text{for each } j = 1, \ldots, n; \; t = 1, \ldots, T - p_j + 1$$

Dantzig-Wolfe decomposition

Reformulation:

$$\begin{aligned}
\min \sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} & h_j(t+p_j)x_{jt} \\
\text{s.t.} \quad \sum_{t=1}^{T-p_j+1} & x_{jt} = 1, \quad \text{for all } j = 1,\dots,n \\
& x_{jt} \in X \quad \text{for each } j = 1,\dots,n;\ t = 1,\dots,T-p_j+1
\end{aligned}$$

where $X = \left\{ x \in \{0,1\} : \sum_{j=1}^{n} \sum_{s=t-p_j+1}^{t} x_{js} \leq 1, \text{ for each } t = 1,\dots,T \right\}$

Dantzig-Wolfe decomposition

Reformulation:

$$\begin{aligned}
\min \sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} & h_j(t+p_j)x_{jt} \\
\text{s.t.} \quad \sum_{t=1}^{T-p_j+1} & x_{jt} = 1, \quad \text{for all } j = 1, \ldots, n \\
& x_{jt} \in X \quad \text{for each } j = 1, \ldots, n;\ t = 1, \ldots, T - p_j + 1
\end{aligned}$$

where $X = \left\{ x \in \{0,1\} : \sum_{j=1}^{n} \sum_{s=t-p_j+1}^{t} x_{js} \leq 1, \text{ for each } t = 1, \ldots, T \right\}$

$x^l, l = 1, \ldots, L$ extreme points of X.

$$X = \left\{ \begin{array}{lll} x \in \{0,1\} & : & x = \sum_{l=1}^{L} \lambda_l x^l \\ & & \sum_{l=1}^{L} \lambda_l = 1, \\ & & \lambda_l \in \{0,1\} \end{array} \right\}$$

Dantzig-Wolfe decomposition

Reformulation:

$$\min \sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} h_j(t+p_j)x_{jt}$$

$$\text{s.t.} \quad \sum_{t=1}^{T-p_j+1} x_{jt} = 1, \qquad \text{for all } j=1,\ldots,n$$

$$x_{jt} \in X \qquad \text{for each } j=1,\ldots,n;\ t=1,\ldots,T-p_j+1$$

where $X = \left\{ x \in \{0,1\} : \sum_{j=1}^{n} \sum_{s=t-p_j+1}^{t} x_{js} \leq 1,\ \text{for each } t=1,\ldots,T \right\}$

$x^l, l=1,\ldots,L$ extreme points of X.

$$X = \left\{ \begin{array}{lll} x \in \{0,1\} & : & x = \sum_{l=1}^{L} \lambda_l x^l \\ & & \sum_{l=1}^{L} \lambda_l = 1, \\ & & \lambda_l \in \{0,1\} \end{array} \right\}$$

matrix of X is interval matrix

extreme points are integral

they are pseudo-schedules

Dantzig-Wolfe decomposition

Substituting X in original model getting master problem

$$
\begin{aligned}
\min \sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} & h_j(t+p_j)(\sum_{l=1}^{L} \lambda_l x^l) \\
\text{s.t. } \sum_{l=1}^{L} & \left(\sum_{t=1}^{T-p_j+1} x_{jt}^l \right) \lambda_l = 1, \quad \text{for all } j = 1, \ldots, n \\
& \sum_{l=1}^{L} \lambda_l = 1, \\
& \lambda_l \in \{0, 1\}
\end{aligned}
$$

Dantzig-Wolfe decomposition

Substituting X in original model getting master problem

$$\min \sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} h_j(t+p_j)\left(\sum_{l=1}^{L} \lambda_l x^l\right)$$

$$\text{s.t. } \sum_{l=1}^{L} \left(\sum_{t=1}^{T-p_j+1} x_{jt}^l \right) \lambda_l = 1, \quad \text{for all } j = 1, \ldots, n \Longleftarrow \sum_{l=1}^{L} n_j^l \lambda_l = 1$$

$$\sum_{l=1}^{L} \lambda_l = 1,$$

$$\lambda_l \in \{0, 1\}$$

- n_j^l number of times job j appears in pseudo-schedule l

Dantzig-Wolfe decomposition

Substituting X in original model getting master problem

$$\min \sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} h_j(t+p_j)(\sum_{l=1}^{L} \lambda_l x^l)$$

$$\text{s.t.} \sum_{l=1}^{L} \left(\sum_{t=1}^{T-p_j+1} x_{jt}^l \right) \lambda_l = 1, \quad \text{for all } j = 1, \ldots, n \Longleftarrow \sum_{l=1}^{L} n_j^l \lambda_l = 1$$

$$\sum_{l=1}^{L} \lambda_l = 1,$$

$$\lambda_l \in \{0,1\} \Longleftarrow \lambda_l \geq 0 \text{ LP-relaxation}$$

- n_j^l number of times job j appears in pseudo-schedule l

Dantzig-Wolfe decomposition

Substituting X in original model getting master problem

$$\min \sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} h_j(t+p_j)(\sum_{l=1}^{L} \lambda_l x^l)$$

$$\text{s.t. } \sum_{l=1}^{L} \left(\sum_{t=1}^{T-p_j+1} x_{jt}^l \right) \lambda_l = 1, \quad \text{for all } j = 1, \ldots, n \Longleftarrow \sum_{l=1}^{L} n_j^l \lambda_l = 1$$

$$\sum_{l=1}^{L} \lambda_l = 1,$$

$$\lambda_l \in \{0,1\} \Longleftarrow \lambda_l \geq 0 \text{ LP-relaxation}$$

- n_j^l number of times job j appears in pseudo-schedule l
- solve LP-relaxation by column generation on pseudo-schedules x^l

Dantzig-Wolfe decomposition

Substituting X in original model getting master problem

$$
\begin{aligned}
& \min \sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} h_j(t+p_j)\Big(\sum_{l=1}^{L} \lambda_l x^l\Big) \\
\pi \quad & \text{s.t.} \sum_{l=1}^{L} \left(\sum_{t=1}^{T-p_j+1} x_{jt}^l \right) \lambda_l = 1, \quad \text{for all } j = 1, \ldots, n \Longleftarrow \sum_{l=1}^{L} n_j^l \lambda_l = 1 \\
\alpha \quad & \sum_{l=1}^{L} \lambda_l = 1, \\
& \lambda_l \in \{0,1\} \Longleftarrow \lambda_l \geq 0 \text{ LP-relaxation}
\end{aligned}
$$

- n_j^l number of times job j appears in pseudo-schedule l
- solve LP-relaxation by column generation on pseudo-schedules x^l

Dantzig-Wolfe decomposition

Substituting X in original model getting master problem

$$
\begin{array}{lll}
 & \min \sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} h_j(t+p_j)(\sum_{l=1}^{L} \lambda_l x^l) & \\
\pi & \text{s.t. } \sum_{l=1}^{L} \left(\sum_{t=1}^{T-p_j+1} x^l_{jt} \right) \lambda_l = 1, & \text{for all } j = 1, \ldots, n \Longleftarrow \sum_{l=1}^{L} n^l_j \lambda_l = 1 \\
\alpha & \sum_{l=1}^{L} \lambda_l = 1, & \\
 & \lambda_l \in \{0, 1\} \Longleftarrow \lambda_l \geq 0 \text{ LP-relaxation} &
\end{array}
$$

- n^l_j number of times job j appears in pseudo-schedule l
- solve LP-relaxation by column generation on pseudo-schedules x^l
- reduced cost of λ_k is $\bar{c}_k = \sum_{j=1}^{n} \sum_{t=1}^{T-p_j+1} (c_{jt} - \pi_j) x^k_{jt} - \alpha$

Delayed Column Generation
Simplex in matrix form

$$\min \{cx \mid Ax = b, x \geq 0\}$$

In matrix form:

$$\begin{bmatrix} 0 & A \\ -1 & c \end{bmatrix} \begin{bmatrix} z \\ x \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}$$

- $\mathcal{B} = \{1, 2, \ldots, p\}$ basic variables
- $\mathcal{L} = \{1, 2, \ldots, q\}$ non-basis variables (will be set to lower bound = 0)
- $(\mathcal{B}, \mathcal{L})$ basis structure
- $x_\mathcal{B}, x_\mathcal{L}, c_\mathcal{B}, c_\mathcal{L}$,
- $B = [A_1, A_2, \ldots, A_p]$, $L = [A_{p+1}, A_{p+2}, \ldots, A_{p+q}]$

$$\begin{bmatrix} B & L & 0 \\ c_\mathcal{B} & c_\mathcal{L} & 1 \end{bmatrix} \begin{bmatrix} x_\mathcal{B} \\ x_\mathcal{L} \\ -z \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}$$

Simplex algorithm sets $x_{\mathcal{L}} = 0$ and $x_{\mathcal{B}} = B^{-1}b$
B invertible, hence rows linearly independent

$$Bx_{\mathcal{B}} + Lx_{\mathcal{L}} = b \quad \Rightarrow \quad x_{\mathcal{B}} + B^{-1}Lx_{\mathcal{L}} = B^{-1}b \quad \Rightarrow \quad \left[\begin{array}{l} x_{\mathcal{L}} = 0 \\ x_{\mathcal{B}} = B^{-1}b \end{array} \right.$$

The objective function is obtained by multiplying and subtracting constraints by means of multipliers π (the dual variables)

$$z = \sum_{j=1}^{p} \left[c_j - \sum_{i=1}^{p} \pi_i a_{ij} \right] + \sum_{j=1}^{q} \left[c_j - \sum_{i=1}^{p} \pi_i a_{ij} \right] + \sum_{i=1}^{p} \pi_i b_i$$

Each basic variable has cost null in the objective function

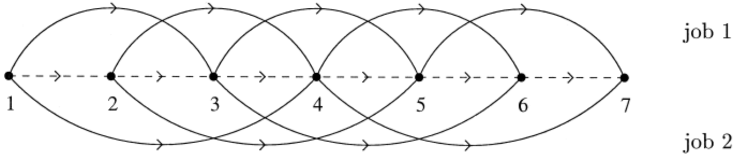
$$c_j - \sum_{i=1}^{p} \pi_i a_{ij} = 0 \qquad \Longrightarrow \qquad \pi = B^{-1}c_{\mathcal{B}}$$

Reduced costs $\bar{c}_j$ of non-basic variables:

$$\bar{c}_j = c_j - \sum_{i=1}^{p} \pi_i a_{ij}$$

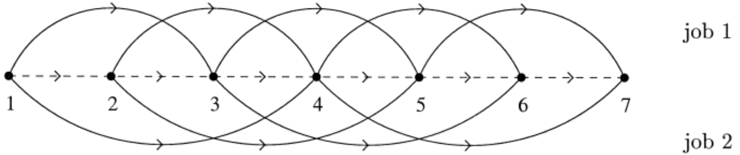
Pricing problem

- Subproblem solved by finding shortest path in a network N with
  - $1, 2, \ldots, T+1$ nodes corresponding to time periods
  - process arcs, for all j, t, $t \rightarrow t + p_j$ and cost $c_{jt} - \pi_j$
  - idle time arcs, for all t, $t \rightarrow t+1$ and cost 0



Pricing problem

- Subproblem solved by finding shortest path in a network N with
  - $1, 2, \ldots, T+1$ nodes corresponding to time periods
  - process arcs, for all j, t, $t \rightarrow t + p_j$ and cost $c_{jt} - \pi_j$
  - idle time arcs, for all t, $t \rightarrow t+1$ and cost 0



- a path in this network corresponds to a pseudo-schedule in which a job may be started more than once or not processed.
- since network is directed and acyclic, shortest path found in $O(nT)$

Further Readings

- the lower bound on the master problem produced by the LP-relaxation of the restricted master problem can be tighten by inequalities

  J. van den Akker, C. Hurkens and M. Savelsbergh. Time-Indexed Formulations for Machine Scheduling Problems: Column Generation. INFORMS Journal On Computing, 2000, 12(2) , 111-124

Further Readings

- the lower bound on the master problem produced by the LP-relaxation of the restricted master problem can be tighten by inequalities

  J. van den Akker, C. Hurkens and M. Savelsbergh. Time-Indexed Formulations for Machine Scheduling Problems: Column Generation. INFORMS Journal On Computing, 2000, 12(2) , 111-124

- *A. Pessoa, E. Uchoa, M.P. de Aragão and R. Rodrigues. Exact algorithm over an arc-time-indexed formulation for parallel machine scheduling problems. 2010, 2, 259-290*

  proposes another time index formulation that dominates this one. They can solve consistently instances up to 100 jobs.

Outline

$1 \,|\,|\, \sum h_j(C_j)$

A lot of work done on $1 \,|\,|\, \sum w_j T_j$
[single-machine total weighted tardiness]

- $1 \,|\,|\, \sum T_j$ is hard in ordinary sense, hence admits a pseudo polynomial algorithm (dynamic programming in $O(n^4 \sum p_j)$)
- $1 \,|\,|\, \sum w_j T_j$ strongly NP-hard (reduction from 3-partition)

$1 \mid\mid \sum h_j(C_j)$

- generalization of $\sum w_j T_j$ hence strongly NP-hard
- (forward) dynamic programming algorithm

J set of jobs already scheduled;

$V(J) = \sum_{j \in J} h_j(C_j)$

$1 \,||\, \sum h_j(C_j)$

- generalization of $\sum w_j T_j$ hence strongly NP-hard
- (forward) dynamic programming algorithm

J set of jobs already scheduled;

$V(J) = \sum_{j \in J} h_j(C_j)$

Step 1: Set $J = \emptyset$, $V(j) = h_j(p_j)$, $\quad j = 1, \ldots, n$

Step 2: $V(J) = \min_{j \in J} \left(V(J - \{j\}) + h_j \left(\sum_{k \in J} p_k \right) \right)$

Step 3: If $J = \{1, 2, \ldots, n\}$ then $V(\{1, 2, \ldots, n\})$ is optimum, otherwise go to Step 2.

$1 \,||\, \sum h_j(C_j)$

- generalization of $\sum w_j T_j$ hence strongly NP-hard
- (forward) dynamic programming algorithm $O(2^n)$

J set of jobs already scheduled;

$V(J) = \sum_{j \in J} h_j(C_j)$

Step 1: Set $J = \emptyset$, $V(j) = h_j(p_j)$, $j = 1, \ldots, n$

Step 2: $V(J) = \min_{j \in J} \left(V(J - \{j\}) + h_j \left(\sum_{k \in J} p_k \right) \right)$

Step 3: If $J = \{1, 2, \ldots, n\}$ then $V(\{1, 2, \ldots, n\})$ is optimum, otherwise go to Step 2.

Outline

1. Branch and Bound
2. IP Models
3. Dynamic Programming
4. Local Search

$1 \mid\mid \sum h_j(C_j)$

Local search

$1 \mid\mid \sum h_j(C_j)$

Local search

- search space (solution representation)
- initial solution
- neghborhood function
- evaluation function
- step function
- termination predicte

$1 \,||\, \sum h_j(C_j)$

Local search

- search space (solution representation)
- initial solution
- neghborhood function
- evaluation function
- step function
- termination predicte

Efficient implementations

$1 || \sum h_j(C_j)$

Local search

- search space (solution representation)
- initial solution
- neghborhood function
- evaluation function
- step function
- termination predicte

Efficient implementations

A. Incremental updates

B. Neighborhood pruning

$1 \,||\, \sum h_j(C_j)$

Neighborhood updates and pruning

- Interchange neigh.: size $\binom{n}{2}$ and $O(|i-j|)$ evaluation each

$1 \,||\, \sum h_j(C_j)$

Neighborhood updates and pruning

- Interchange neigh.: size $\binom{n}{2}$ and $O(|i-j|)$ evaluation each
  - first-improvement: π_j, π_k

    $p_{\pi_j} \leq p_{\pi_k}$ for improvements, $w_j T_j + w_k T_k$ must decrease because jobs in $\pi_j, \ldots, \pi_k$ can only increase their tardiness.

    $p_{\pi_j} \geq p_{\pi_k}$ possible use of auxiliary data structure to speed up the computation
  - best-improvement: π_j, π_k

    $p_{\pi_j} \leq p_{\pi_k}$ for improvements, $w_j T_j + w_k T_k$ must decrease at least as the best interchange found so far because jobs in $\pi_j, \ldots, \pi_k$ can only increase their tardiness.

    $p_{\pi_j} \geq p_{\pi_k}$ possible use of auxiliary data structure to speed up the computation

$1 \mid\mid \sum h_j(C_j)$

Neighborhood updates and pruning

- Interchange neigh.: size $\binom{n}{2}$ and $O(|i-j|)$ evaluation each
  - first-improvement: π_j, π_k

    $p_{\pi_j} \leq p_{\pi_k}$ for improvements, $w_j T_j + w_k T_k$ must decrease because jobs in $\pi_j, \ldots, \pi_k$ can only increase their tardiness.

    $p_{\pi_j} \geq p_{\pi_k}$ possible use of auxiliary data structure to speed up the computation
  - best-improvement: π_j, π_k

    $p_{\pi_j} \leq p_{\pi_k}$ for improvements, $w_j T_j + w_k T_k$ must decrease at least as the best interchange found so far because jobs in $\pi_j, \ldots, \pi_k$ can only increase their tardiness.

    $p_{\pi_j} \geq p_{\pi_k}$ possible use of auxiliary data structure to speed up the computation
- Swap: size $n-1$ and $O(1)$ evaluation each

$1 \,||\, \sum h_j(C_j)$

Neighborhood updates and pruning

- Interchange neigh.: size $\binom{n}{2}$ and $O(|i-j|)$ evaluation each
  - first-improvement: π_j, π_k

    $p_{\pi_j} \le p_{\pi_k}$ for improvements, $w_j T_j + w_k T_k$ must decrease because jobs in $\pi_j, \dots, \pi_k$ can only increase their tardiness.

    $p_{\pi_j} \ge p_{\pi_k}$ possible use of auxiliary data structure to speed up the computation
  - best-improvement: π_j, π_k

    $p_{\pi_j} \le p_{\pi_k}$ for improvements, $w_j T_j + w_k T_k$ must decrease at least as the best interchange found so far because jobs in $\pi_j, \dots, \pi_k$ can only increase their tardiness.

    $p_{\pi_j} \ge p_{\pi_k}$ possible use of auxiliary data structure to speed up the computation
- Swap: size $n-1$ and $O(1)$ evaluation each
- Insert: size $(n-1)^2$ and $O(|i-j|)$ evaluation each

$1 \,||\, \sum h_j(C_j)$

Neighborhood updates and pruning

- Interchange neigh.: size $\binom{n}{2}$ and $O(|i-j|)$ evaluation each
  - first-improvement: π_j, π_k
    - $p_{\pi_j} \le p_{\pi_k}$ for improvements, $w_j T_j + w_k T_k$ must decrease because jobs in $\pi_j, \dots, \pi_k$ can only increase their tardiness.
    - $p_{\pi_j} \ge p_{\pi_k}$ possible use of auxiliary data structure to speed up the computation
  - best-improvement: π_j, π_k
    - $p_{\pi_j} \le p_{\pi_k}$ for improvements, $w_j T_j + w_k T_k$ must decrease at least as the best interchange found so far because jobs in $\pi_j, \dots, \pi_k$ can only increase their tardiness.
    - $p_{\pi_j} \ge p_{\pi_k}$ possible use of auxiliary data structure to speed up the computation
- Swap: size $n-1$ and $O(1)$ evaluation each
- Insert: size $(n-1)^2$ and $O(|i-j|)$ evaluation each
  But possible to speed up with systematic examination by means of swaps: an interchange is equivalent to $|i-j|$ swaps hence overall examination takes $O(n^2)$

Dynasearch

- two interchanges δ_{jk} and δ_{lm} are independent if $\max\{j,k\} < \min\{l,m\}$ or $\min\{l,k\} > \max\{l,m\}$;

Dynasearch

- two interchanges δ_{jk} and δ_{lm} are independent
  if $\max\{j,k\} < \min\{l,m\}$ or $\min\{l,k\} > \max\{l,m\}$;
- the dynasearch neighborhood is obtained by a series of independent interchanges;

Dynasearch

- two interchanges δ_{jk} and δ_{lm} are independent if $\max\{j,k\} < \min\{l,m\}$ or $\min\{l,k\} > \max\{l,m\}$;
- the dynasearch neighborhood is obtained by a series of independent interchanges;
- it has size $2^{n-1} - 1$;

Dynasearch

- two interchanges δ_{jk} and δ_{lm} are independent
  if $\max\{j, k\} < \min\{l, m\}$ or $\min\{l, k\} > \max\{l, m\}$;
- the dynasearch neighborhood is obtained by a series of independent interchanges;
- it has size $2^{n-1} - 1$;
- but a best move can be found in $O(n^3)$ searched by dynamic programming;

Dynasearch

- two interchanges δ_{jk} and δ_{lm} are independent if $\max\{j,k\} < \min\{l,m\}$ or $\min\{l,k\} > \max\{l,m\}$;
- the dynasearch neighborhood is obtained by a series of independent interchanges;
- it has size $2^{n-1} - 1$;
- but a best move can be found in $O(n^3)$ searched by dynamic programming;
- it yields in average better results than the interchange neighborhood alone.

Table 1 Data for the Problem Instance

Job j	1	2	3	4	5	6
Processing time p_j	3	1	1	5	1	5
Weight w_j	3	5	1	1	4	4
Due date d_j	1	5	3	1	3	1

Table 1 Data for the Problem Instance

Job j	1	2	3	4	5	6
Processing time p_j	3	1	1	5	1	5
Weight w_j	3	5	1	1	4	4
Due date d_j	1	5	3	1	3	1

Table 2 Swaps Made by Best-Improve Descent

Iteration	Current Sequence	Total Weighted Tardiness
	1 2 3 4 5 6	109
1	1 2 3 5 4 6	90
2	1 2 3 5 6 4	75
3	5 2 3 1 6 4	70

Table 1 Data for the Problem Instance

Job j	1	2	3	4	5	6
Processing time p_j	3	1	1	5	1	5
Weight w_j	3	5	1	1	4	4
Due date d_j	1	5	3	1	3	1

Table 2 Swaps Made by Best-Improve Descent

Iteration	Current Sequence	Total Weighted Tardiness
	1 2 3 4 5 6	109
1	1 2 3 5 4 6	90
2	1 2 3 5 6 4	75
3	5 2 3 1 6 4	70

Table 3 Dynasearch Swaps

Iteration	Current Sequence	Total Weighted Tardiness
	1 2 3 4 5 6	109
1	1 3 2 5 4 6	89
2	1 5 2 3 6 4	68
3	5 1 2 3 6 4	67

- state (k, π)

- state (k, π)
- π_k is the partial sequence at state (k, π) that has min $\sum wT$

- state (k, π)
- π_k is the partial sequence at state (k, π) that has min $\sum wT$
- π_k is obtained from state (i, π)

$$\begin{cases} \text{appending job } \pi(k) \text{ after } \pi(i) & i = k-1 \\ \text{appending job } \pi(k) \text{ and interchanging } \pi(i+1) \text{ and } \pi(k) & 0 \le i < k-1 \end{cases}$$

- state (k, π)
- π_k is the partial sequence at state (k, π) that has min $\sum wT$
- π_k is obtained from state (i, π)

$$\begin{cases} \text{appending job } \pi(k) \text{ after } \pi(i) & i = k-1 \\ \text{appending job } \pi(k) \text{ and interchanging } \pi(i+1) \text{ and } \pi(k) & 0 \leq i < k-1 \end{cases}$$

- $F(\pi_0) = 0; \qquad F(\pi_1) = w_{\pi(1)} \left(p_{\pi(1)} - d_{\pi(1)}\right)^+;$

$$F(\pi_k) = \min \begin{cases} F(\pi_{k-1}) + w_{\pi(k)} \left(C_{\pi(k)} - d_{\pi(k)}\right)^+, \\ \min\limits_{1 \leq i < k-1} \{F(\pi_i) + w_{\pi(k)} \left(C_{\pi(i)} + p_{\pi(k)} - d_{\pi(k)}\right)^+ + \\ \quad + \sum_{j=i+2}^{k-1} w_{\pi(j)} \left(C_{\pi(j)} + p_{\pi(k)} - p_{\pi(i+1)} - d_{\pi(j)}\right)^+ + \\ \quad + w_{\pi(i+1)} \left(C_{\pi(k)} - d_{\pi(i+1)}\right)^+ \} \end{cases}$$

- The best choice is computed by recursion in $O(n^3)$ and the optimal series of interchanges for $F(\pi_n)$ is found by backtrack.
- Local search with dynasearch neighborhood starts from an initial sequence, generated by ATC, and at each iteration applies the best dynasearch move, until no improvement is possible (that is, $F(\pi_n^t) = F(\pi_n^{(t-1)})$, for iteration t).
- Speedups:
  - pruning with considerations on $p_{\pi(k)}$ and $p_{\pi(i+1)}$
  - maintainig a string of late, no late jobs
  - h_t largest index s.t. $\pi^{(t-1)}(k) = \pi^{(t-2)}(k)$ for $k = 1, \ldots, h_t$ then $F(\pi_k^{(t-1)}) = F(\pi_k^{(t-2)})$ for $k = 1, \ldots, h_t$ and at iter t no need to consider $i < h_t$.

Dynasearch, refinements:

- [Grosso et al. 2004] add insertion moves to interchanges.
- [Ergun and Orlin 2006] show that dynasearch neighborhood can be searched in $O(n^2)$.

Performance:

- exact solution via branch and bound feasible up to 40 jobs [Potts and Wassenhove, Oper. Res., 1985]
- exact solution via time-indexed integer programming formulation used to lower bound in branch and bound solves instances of 100 jobs in 4-9 hours [Pan and Shi, Math. Progm., 2007]
- dynasearch: results reported for 100 jobs within a 0.005% gap from optimum in less than 3 seconds [Grosso et al., Oper. Res. Lett., 2004]

Summary

$1 \,|\,|\, \sum w_j C_j$: weighted shortest processing time first is optimal

$1 \,|\,|\, \sum_j U_j$: Moore's algorithm

$1 \,|\, prec \,|\, L_{max}$: Lawler's algorithm, backward dynamic programming in $O(n^2)$ [Lawler, 1973]

$1 \,|\,|\, \sum h_j(C_j)$: dynamic programming in $O(2^n)$

$1 \,|\,|\, \sum w_j T_j$: local search and dynasearch

$1 \,|\, r_j, (prec) \,|\, L_{max}$: branch and bound

$1 \,|\,|\, \sum w_j T_j$: column generation approaches

$1 \,|\, s_{jk} \,|\, C_{max}$: in the special case, Gilmore and Gomory algorithm optimal in $O(n^2)$

Multiobjective: Multicriteria Optimization

Stochastic scheduling

Multiobjective Scheduling

Multiobjective scheduling

Resolution process and decision maker intervention:

- a priori methods (definition of weights, importance)
  - goal programming
  - weighted sum
  - ...
- interactive methods
- a posteriori methods
  - Pareto optimality
  - ...