DMP204
SCHEDULING,
TIMETABLING AND ROUTING

Lecture 11
Single Machine Models, Branch and Bound

Marco Chiarandini

Outline

Outline

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

[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

[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 processed node

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

Components

- Initial feasible solution (heuristic) – might be crucial!
1. Bounding function
2. Strategy for selecting
3. Branching
- Fathmoing (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
- 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
  (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

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

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

$1 \mid \mid \sum w_j T_j$

- **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.

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

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

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

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

$1 \mid s_{jk} \mid C_{max}$

[Makespan with sequence-dependent setup times]

- general case is NP-hard (traveling salesman reduction).
- special case:

  parameters for job j:
  - a_j initial state
  - b_j final state

  such that:

$$s_{jk} \propto |a_k - b_j|$$

  [Gilmore and Gomory, 1964] give an $O(n^2)$ algorithm

- assume $b_0 \leq b_1 \leq \ldots \leq b_n$ ($k > j$ and $b_k \geq b_j$)
- one-to-one correspondence with solution of TSP with $n+1$ cities
  city 0 has a_0, b_0
  start at b_0 finish at a_0
- tour representation $\phi : \{0, 1, \ldots, n\} \mapsto \{0, 1, \ldots, n\}$
  (permutation map, single linked array)
- Hence,

$$\min \quad c(\phi) = \sum_{i=1}^{n} c_{i,\phi(i)} \tag{1}$$

$$\phi(S) \neq S \qquad \forall S \subset V \tag{2}$$

- find ϕ^* by ignoring (2)
  make ϕ^* a tour by interchanges chosen solving a min spanning tree and applied in a certain order

- Interchange δ^{jk}

$$\delta^{jk}(\phi) = \{\phi' \mid \phi'(j) = \phi(k), \quad \phi(k) = \phi(j), \quad \phi'(l) = \phi(l), \quad \forall l \neq j, k\}$$

- Cost

$$\begin{aligned} c_\phi(\delta^{jk}) &= c(\delta^{jk}(\phi)) - c(\phi) \\ &= \| \, [b_j, b_k] \cap [a_{\phi(j)}, a_{\phi(k)}] \, \| \end{aligned}$$

- **Theorem:** Let ϕ^* be a permutation that ranks the a that is $k > j$ implies $a_{\phi(k)} \geq a_{\phi(j)}$ then

$$c(\phi^*) = \min_{\phi} c(\phi).$$

- **Lemma:** If ϕ is a permutation consisting of cycles $C_1, \ldots, C_p$ and δ^{jk} is an interchange with $j \in C_r$ and $k \in C_s$, $r \neq s$, then $\delta^{jk}(\phi)$ contains the same cycles except that C_r and C_s have been replaced by a single cycle containing all their nodes.

- **Theorem:** Let $\delta^{j_1k_1}, \delta^{j_2k_2}, \ldots, \delta^{j_pk_p}$ be the interchanges corresponding to the arcs of a spanning tree of G_{ϕ^*}. The arcs may be taken in any order. Then ϕ',

$$\phi' = \delta^{j_1k_1} \circ \delta^{j_2k_2} \circ \ldots \circ \delta^{j_pk_p}(\phi^*)$$

  is a tour.

- The $p-1$ interchanges can be found by greedy algorithm (similarity to Kruskal for min spanning tree)

- **Lemma:** There is a minimum spanning tree in G_{ϕ^*} that contains only arcs $\delta^{j,j+1}$.

- Generally, $c(\phi') \neq c(\delta^{j_1k_1}) + c(\delta^{j_2k_2}) + \ldots + c(\delta^{j_pk_p})$.

- $$\text{node } j \text{ in } \phi \text{ is of} \begin{cases} \text{Type I}, & \text{if } b_j \leq a_{\phi(j)} \\ \text{Type II}, & \text{otherwise} \end{cases}$$

$$\text{interchange } jk \text{ is of} \begin{cases} \text{Type I}, & \text{if lower node of type I} \\ \text{Type II}, & \text{if lower node of type II} \end{cases}$$

- Order:
  interchanges in Type I in decreasing order
  interchanges in Type II in increasing order

- Apply to ϕ^* interchanges of Type I and Type II in that order.

- **Theorem:** The tour found is a minimal cost tour.

Resuming the final algorithm [Gilmore and Gomory, 1964]:

Step 1: Arrange b_j in order of size and renumber jobs so that $b_j \leq b_{j+1}$, $j = 1, \ldots, n$.

Step 2: Arrange a_j in order of size.

Step 3: Define ϕ by $\phi(j) = k$ where k is the $j+1$-smallest of the a_j.

Step 4: Compute the interchange costs $c_{\delta^{j,j+1}}$, $j = 0, \ldots, n-1$

$$c_{\delta^{j,j+1}} = \| \, [b_j, b_{j+1}] \cap [a_{\phi(j)}, a_{\phi(i)}] \, \|$$

Step 5: While G has not one single component, Add to G_ϕ the arc of minimum cost $c(\delta^{j,j+1})$ such that j and $j+1$ are in two different components.

Step 6: Divide the arcs selected in Step 5 in Type I and II.
Sort Type I in decreasing and Type II increasing order of index.
Apply the relative interchanges in the order.

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 $O(2^n)$

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

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

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

$1 \mid \mid \sum w_j T_j$: column generation approaches

Multiobjective: Multicriteria Optimization

Complexity resume

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

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