Batch Scheduling

Pierre, Slava, Ming-Ming, Jules

Grenoble Alpes University

17.01.2023

Outline

1. Dynamic programming : Knapsack problem with width

2. Heuristic Tree Search : Knapsack problem with width and conflicts

3. Column Generation and Dynamic Programming : the Single machine batch scheduling problem with Makespan objective

4. Column Generation and Heuristic Tree Search

Dynamic programming : Knapsack problem with width

We consider the Knapsack problem with width:

Input

- n items; for each item $j = 1, \ldots, n$
  - a weight $w_j \in \mathbb{N}^*$
  - a width $l_j \in \mathbb{N}$
  - a profit $p_j \in \mathbb{N}^*$
- a capacity $C \in \mathbb{N}^*$

Problem

- Select a subset of items such that the total weight of the selected items does not exceed the knapsack capacity
- Objective: maximize the total profit of the selected items minus the maximum width among the selected items

Algorithm description

- Initialization;
- Sort the items by increasing width $[l_1, l_2, ..., l_n]$;
- Compute a basic knapsack with all the items;
  - Initialize f(i, c) with 0 for all i, c;
    (f(i, c) is the max value for capacity = c and number of items = i)
  - Iterate over items i
  - For c in $0, .. \min(w[i], C) : f[i][c] = f[i-1][c]$
  - For c in w[i], .. capacity: $f[i][c] = \max(f[i][c - w[i]] + p[i], f[i-1][c])$
- Find $i \in \{1, ..., n\}$ s.t $f(i, C) - l_i$ is maximal ;
- Return the solution corresponding to this index.

Complexity

- Sorting is done in $O(n \log n)$
- Solving the knapsack problem with dynamic programming is done in O(nC)
- Overall, **complexity is O(nC)** (since usually C is much bigger than n)

Heuristic Tree Search : Knapsack problem with width and conflicts

We consider a Knapsack problem with width and conflicts.

Input

- n items; for each item $j = 1, \ldots, n$
  - a weight $w_j \in \mathbb{N}^+$
  - a width $l_j \in \mathbb{N}^+$
  - a profit $p_j \in \mathbb{N}^+$
- a capacity $C \in \mathbb{N}^+$
- a graph G such that each node corresponds to an item

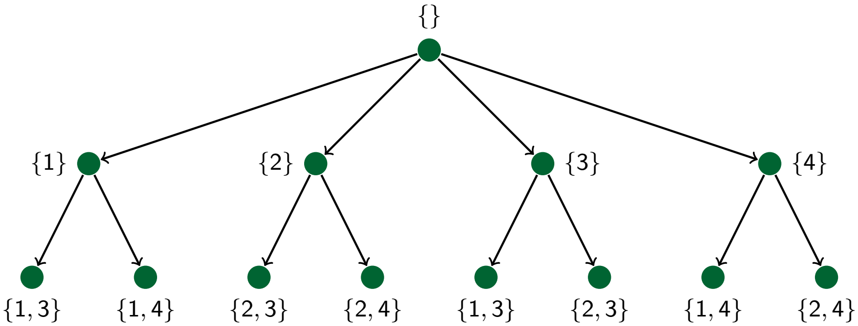
Problem

- Select a subset of items such that the total weight of the selected items does not exceed the knapsack capacity
- If there exists an edge between vertex j_1 and j_2 in G, then item j_1 and item j_2 must not be both selected
- Maximize the total profit of the selected items minus the maximum width amon the selected items

Tree Generation

Number of items: 4

Conflicts: $[(1,2),(3,4)]$



Bound

Conflicts: $[(2, 3)]$

Item	**1**	**2**	**3**	**4**	**5**
p_j	7	3	5	1	2

Bound

Conflicts: $[(2, 3)]$

Item	**1**	**2**	**3**	**4**	**5**
p_j	7	3	5	1	2

Node 1:

- Items: $\{1, 2\}$
- **Value: 10**
- Potential new items: $\{4, 5\}$
- **Potential increase:** $\boldsymbol{p_4} + \boldsymbol{p_5} = 3$

Bound

Conflicts: $[(2,3)]$

Item	**1**	**2**	**3**	**4**	**5**
p_j	7	3	5	1	2

Node 1:

- Items: $\{1,2\}$
- **Value: 10**
- Potential new items: $\{4,5\}$
- **Potential increase:** $\boldsymbol{p_4} + \boldsymbol{p_5} = 3$

Node 2:

- Items: $\{1,3,5\}$
- **Value: 14**
- Potential new items: $\{4\}$
- Potential increase: $p_4 = 1$

Bound

Conflicts: $[(2, 3)]$

Item	**1**	**2**	**3**	**4**	**5**
p_j	7	3	5	1	2

Node 1:

- Items: $\{1, 2\}$
- **Value: 10**
- Potential new items: $\{4, 5\}$
- **Potential increase:** $p_4 + p_5 = 3$

Node 2:

- Items: $\{1, 3, 5\}$
- **Value: 14**
- Potential new items: $\{4\}$
- Potential increase: $p_4 = 1$

Value(Node 2) $\geq$Value(Node 1) + Potential Increase(Node 1)

Bound

Conflicts: $[(2,3)]$

Item	**1**	**2**	**3**	**4**	**5**
p_j	7	3	5	1	2

Node 1:

- Items: $\{1,2\}$
- **Value: 10**
- Potential new items: $\{4,5\}$
- **Potential increase:** $\boldsymbol{p_4 + p_5 = 3}$

Node 2:

- Items: $\{1,3,5\}$
- **Value: 14**
- Potential new items: $\{4\}$
- Potential increase: $p_4 = 1$

Value(Node 2) $\geq$ **Value(Node 1) + Potential Increase(Node 1)**

We do not need to explore **Node 1**.

Domination rule

Conflicts: $[(2,3)]$

Item	**1**	**2**	**3**	**4**	**5**
p_j	7	3	5	1	2
w_j	3	3	2	1	1

Node 1:

- Items: $\{1,2\}$
- **Value: 10**
- **Weight: 6**
- Potential new items: $\{4,5\}$

Node 2:

- Items: $\{1,3\}$
- **Value: 12**
- **Weight: 5**
- Potential new items: $\{4,5\}$

Potential new items(Node 1) = Potential new items(Node 2)

Value(Node 2) $\geq$ Value(Node 1) & Weight(Node 1) $\geq$ Weight(Node 2)

We do not need to explore **Node 1**.

We consider the Single machine batch scheduling problem with Makespan objective:

Input :

- n jobs; for each job $j = 1, \ldots, n$, a processing time $p_j \in \mathbb{N}^*$ and a size $s_j \in \mathbb{N}^*$
- a batch capacity $Q \in \mathbb{N}^*$

Problem

Partition the jobs into batches and sequence the batches such that:

- each job must be in exactly one of the batches
- the processing time of a batch is equal to the longest processing time among all jobs it contains
- the total size of the jobs in a batch does not exceed its capacity
- Objective: minimize the makespan of the schedule

Example

Job	1	2	3	4	5	6
p_j	4	2	2	1	3	1
s_j	1	1	2	2	2	1

Solution

- $Q = 3$, $n = 6$.
- Batch 1 : jobs 1 and 5, $p^1_{max} = 4$, size $= 3$.
- Batch 2 : jobs 2 and 3, $p^2_{max} = 2$, size $= 3$.
- Batch 3 : jobs 4 and 6, $p^3_{max} = 1$, size $= 3$.
- The makespan of this solution is 7.

Université Grenoble Alpes

Exponentional formulation

Modelisation

- Let us define the K feasible patterns (batches) such that :
$$\forall j \in \{1, ..., n\},\ \forall k \in K,\ x_j^k = 1 \text{ if job } j \text{ is in the batch } k.$$
- For all $k \in K$, we define
$p_{max}^k = \max(p_j^k | \textit{job } j \textit{ is in batch } k) = \max(p_j^k x_j^k,\ j \in \{1, ..., n\})$ which is the maximum processing time among the jobs in batch k.
- Variables :
  - $y^k \in \mathbb{N}, \forall k = 1, ..., n$
  - $y^k = 1$ if batch k is scheduled, otherwise $y^k = 0$
- Objective: $\min \sum_{k=1}^K p_{max}^k y^k$
- Constraints: $\sum_{k=1}^K x_j^k y^k = 1 \qquad \forall j = 1, ..., n$ (each job must be in exactly one of the batches)

Remark

The second and third constraints are in creation of patterns.

Algortihm Description

Description

- To solve this MIP model we use column generation. The main difficulty is to choose the column of minimun reduced cost, it's the pricing problem.
- We then want to create a new column (batch) that maximize the profits (but minimize the maximum processing time), respecting the constraints of a batch. This problem is exactly the knapsack with width problem with instance $(w_j, l_j, p_j) = (s_j, p_j, v_j)\ \forall j \in \{1, \ldots, n\}$ where v_j is the value of dual variable j of the previous primal MIP.

Column Generation and Heuristic Tree Search

We consider the Single machine batch scheduling problem with conflicts and Makespan objective:

Input :

- n jobs; for each job $j = 1, \ldots, n$, a processing time $p_j \in$ and a size $s_j \in^+$
- a batch capacity $Q \in \mathbb{N}$
- a graph G such that each node corresponds to a job

Problem

Partition the jobs into batches and sequence the batches such that:

- each job must be in exactly one of the batches
- the processing time of a batch is equal to the longest processing time among all jobs it contains
- the total size of the jobs in a batch does not exceed its capacity
- if there exists an edge between vertex j_1 and vertex j_2 in G, then job j_1 and job j_2 must not be in the same batch
- Objective: minimize the makespan of the schedule

Pricing problem algorithms

Column generation algorithm: Limited discrepancy search

Time limit: 60 seconds

Number of instances: 10

Size of the instances: $\approx$ 50 jobs

Gap to the best							
	Beam256	Beam128	Beam64	Beam32	Beam16	BFS0.05	BFS0.1
Gap	∞	1.75%	0.25%	1.13%	2.53%	0.68%	1.37%

Comparison between problems 3 and 4

Instance 100

Problem 3 (without conflicts):
Value of the solution: 9430

Problem 4 (with conflicts):
Value of the solution: 9247

The set of feasible solutions for the problem 4 is included in the set of feasible solutions for the problem 3, yet we found a better solution for problem 4.