Combinatorial Optimization
Lab 11: Bratley's problem

Industrial Informatics Department
Czech Technical University in Prague
https://industrialinformatics.fel.cvut.cz/

April 29, 2024

Outline of the tutorial:

- Revision – scheduling, $\alpha|\beta|\gamma$ notation (10 minutes)
- Brately's algorithm: application of a branch-and-bound (B&B) (45 minutes)
- Assignment of HW4 (5 minutes)

This tutorial is about scheduling of the tasks on resources. At the beginning we revise the scheduling in general, including the Graham's notation. Afterwards, we focus on problem $1|r_j, \tilde{d}_j|C_{\max}$, which is formally introduced and solved by Bratley's algorithm. Finally, we apply our knowledge and use the Bratley's algorithm to solve the last homework.

1 Revision of scheduling

In scheduling[1] problems, we are trying to find assignment of tasks to resources in time. Tasks can be characterized by some parameters, e.g., release time, deadline/due date, processing time, etc. Considering the machine environment, there can be one machine or multiple machines, which might or might not be related somehow. We might be looking for a feasible schedule, or for an optimal schedule with respect to some objective function. The output of the scheduling is an assignment of tasks to resources in time and it is mostly depicted as a Gantt chart [1] where an independent variable is a discrete time (x-axis) and dependent variable is utilization of resources/processors (y-axis).

1.1 Graham's notation

As there are many different scheduling problems, it is sometimes hard to find a relevant research and see if the problem was already solved or not. For the categorization of scheduling problems, there exists a Graham's (or Graham-Blazewicz) notation, also called $\alpha|\beta|\gamma$ notation [1, 2]. This notation is trying to characterize the scheduling problem based on 3 aspects:

- α contains the characteristics of resources – the number (1 to ∞) and type (P for parallel identical, Q for parallel uniform, etc.),
- β specifies the characteristics of tasks and additional resources (r_j means that each task has release time, *pmtn* means that preemption is allowed, etc.),
- γ denotes an optimality criterion (C_{max} for minimizing the schedule length, L_{max} for minimizing the maximal lateness, $\sum U_j$ for minimizing the number of tasks exceeding their due date, etc.).

[1] Note that there is a difference between the *planning* and *scheduling*. In planning, we select individual actions leading from a starting state to the goal state – we are interested in 'what actions to choose' and 'when'. Whereas in scheduling, we want to know 'when' to perform a given set of actions (given time constraints, resource constraints, objective function etc.).

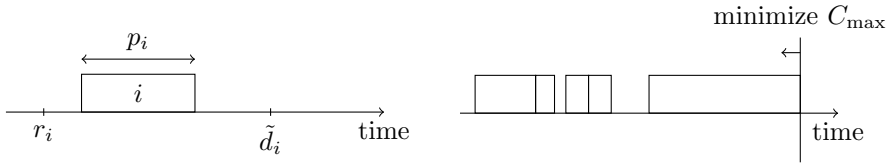
Scheduling zoo There are many scheduling problems, which have been categorized using this standard notation. Scheduling zoo (http://schedulingzoo.lip6.fr/) is a webpage, where you can see some of the scheduling problems together with links to relevant articles describing their complexities. Try to look through the pages, and see the diversity of the scheduling problems.

2 Monoprocessor scheduling and Bratley's algorithm

Now, we will concentrate on a problem of $1|r_j, \tilde{d}_j|C_{\max}$, i.e., the problem with a single machine and tasks characterized by release times and deadlines, where we want to minimize a total length of the schedule.

2.1 Problem statement

Input: We are given a set to tasks $\mathcal{T} = \{T_1, \ldots, T_n\}$, where each task $T_i \in \mathcal{T}$ is characterized by its release time r_i, deadline $\tilde{d}_i$ and processing time p_i, see Figure 1(a).



(a) A single task (b) A schedule

Figure 1: Illustration of the studied problem.

Output: We want to find a feasible schedule (start times of the individual tasks; tasks cannot overlap) such that the completion time of the last task ($C_{\max}$) is minimal, see Figure 1(b).

Complexity: The problem is $\mathcal{NP}$-hard, which can be shown by a polynomial reduction from 3-partition problem. The proof was shown during the lectures.

Question: What would happen if we had problems $1|-|C_{\max}$ or $1|r_j, \tilde{d}_j, pmnt|-$?[a]

[a] Answer: Both would be simple – solvable in polynomial time.

Note: Remember that we have already solved the problem by an ILP (the catering problem). Maybe you have found out that the performance of the ILP model we came up with is not very good (model with big-M might be able to solve, say, 30-40 tasks). A better ILP model (position based model) was discussed during the lectures. Although an ILP is a powerful modeling approach, we might speed up the solving process by designing a specialized algorithm, which will be using pruning techniques designed specifically for the studied problem. We will design such an algorithm now.

2.2 Bratley's algorithm

Bratley's algorithm is based on a branch-and-bound procedure.

Question: Did you meet B&B anywhere?[a]

[a] Answer: Yes, as we have already discussed, an ILP can be solved by the B&B algorithm (branching on the fractional values of the variables and bounding by LP relaxations). Some of you might have also met B&B algorithm for Hamiltonian path problem (taught by prof. Demlova during the course of Logic and Graphs – she called it 'metoda větví a mezí').

Branch and bound (revision): The Branch and Bound algorithm is used to solve discrete optimization problems [1]. This algorithm gradually constructs a tree of partial solutions which are expanded further (branch). If B&B finds an infeasible partial solution or a partial solution such that it is worse than the best found solution, the node with this partial solution is not expanded (bound). Each node of the search tree corresponds to one partial solution and the leaves represent a complete solution. The subtree can be eliminated if:

1. It does not contain any feasible solution.
2. It does not contain an optimal solution.

Basically, the first case implies the second case. However, it can be appropriate to consider them separately for the algorithm construction.
Generally there are two main methods of the solution space searching: breadth-first search and depth-first search. In our case, i.e., the application of the Branch and Bound algorithm in the scheduling, it is advantageous to use the depth-first search since each obtained feasible solution increases the probability of eliminating other parts of the tree without their complete search.

So what does Bratley's algorithm do exactly? Well, it employs smart search over the possible permutations of the tasks. It constructs a tree of partial solutions and prunes these, which are non-optimal.

Question: Why is it enough to use the permutations only, without the information about the start times?[a]

[a] Answer: Permutation represents an order of tasks in the schedule. For a given permutation, we go left-to-right and construct a schedule by setting a start time of a job 'as soon as possible'. Clearly, it is not optimal to shift any task to the right. Hence, we are able to reconstruct a full schedule given the order of the tasks. The only remaining question is 'which order is the optimal one' – this is not a simple question; we need to enumerate all possible permutations and see.

Now, let's look at the permutation tree constructed by the Bratley's algorithm (Figure 2).

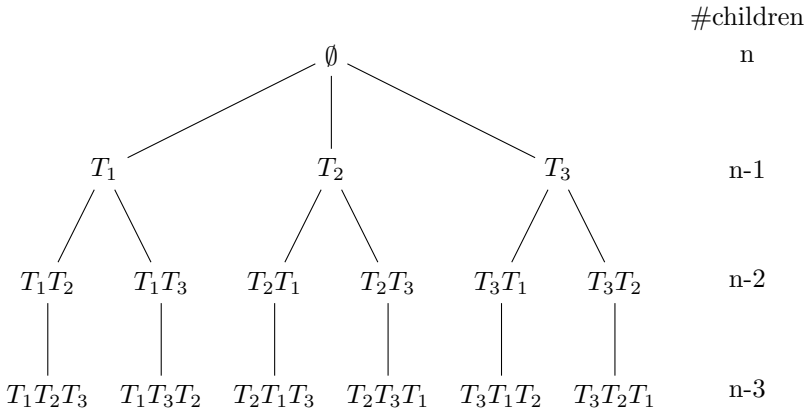


Figure 2: Example of a complete permutation tree for $n = 3$ tasks.

It can be seen, that the complete permutation tree has $n!$ leaves. To evaluate them all would be very inefficient. However, usually we do not need to do that. We might be able to prune some nodes before expanding them further.

We can try to derive some pruning rules using the objective function or the tasks constraints. In each node of the tree, we can compute a lower bound LB (based on the current partial solution) and compare it to global upper bound UB (which is obtained from some feasible solution/approximation algorithm/estimation). On the other hand, we can also prune the current node using the characteristics of non-scheduled tasks, e.g., when some nonscheduled task would surely miss its deadline, it would be meaningless to expand this node further.

Question: Which nodes can be pruned? Can you devise some simple rules, which might help us?[a]

[a]Try to think about it now, the rules will be explained in the following text.

At each node, we will remember

c – length of the partial schedule,

V – a set of non-scheduled tasks.

Now, we can write the rules:

1. **Missed deadline**

   It might happen that unassigned task would miss its deadline when assigned to the current schedule, if that is the case, prune this node. (It is meaningless to continue, because in the future, some task would surely miss its deadline).

$$(\exists\, T_j \in V : \max\{c, r_j\} + p_j > d_j) \Rightarrow \text{prune this node.} \tag{1}$$

2. **Bound on the solution**

   We might have already found some feasible solution, which might not be optimal. However, we can use its quality as an upper bound (UB). We can calculate lover bound (LB) of the current solution and prune this node if $LB \geq UB$.

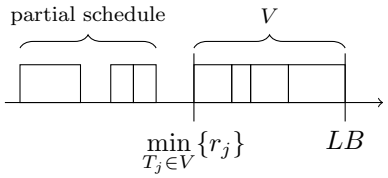


Figure 3: Visualization of the LB.

$$LB = \max\left\{c, \min_{T_j \in V}\{r_j\}\right\} + \sum_{T_j \in V} p_j \tag{2}$$

Note that we are basically relaxing on release times of the tasks in V (except for the minimal one). Then, the order of the tasks is not important and we basically solve (optimally) $1|-|C_{\max}$, which is a relaxation of the original problem.

Question: What to do if we do not have any feasible solution?[a]

[a]Answer: We can use $UB = \max_{T_j \in V}\{\tilde{d}_j\}$, but then the inequality must be strict (we cannot prune the solution, where $LB = UB$ since we do not have a feasible solution for current UB).

3. **Decomposition**

   We might be able to detect, that the partial solution we have in the current node is optimal, therefore it might not be necessary to backtrack.

$$(c \leq \min_{T_j \in V}\{r_j\}) \Rightarrow \text{do not backtrack.} \tag{3}$$

That is because tasks in V need to be scheduled, but cannot be scheduled sooner, and so tasks in $\mathcal{T} \setminus V$ will not affect the final $C_{\max}$. We can start the algorithm again for tasks in V and simply concatenate the solution with the current partial solution of $\mathcal{T} \setminus V$.

2.3 Example

Let us have 4 tasks characterized by parameters given in Table 1.

T_i	p_i	r_i	d_i
T_1	2	4	7
T_2	1	1	5
T_3	2	1	6
T_4	2	0	4

Table 1: Parameters of the tasks.

Exercise: Try to solve this instance of the problem by hand.[a]

[a] The solution tree will be shown on the following page.

The solution tree follows:

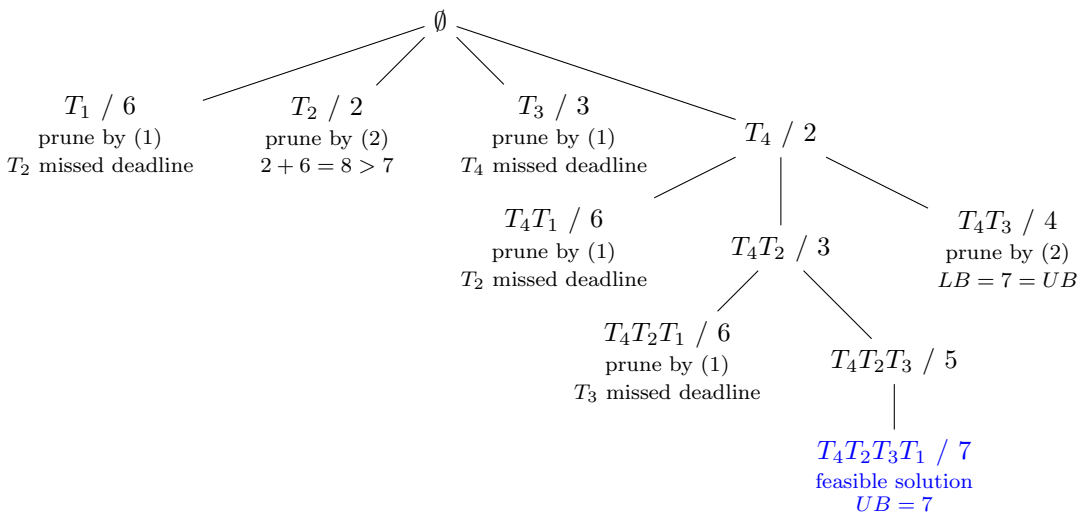


Figure 4: Solution to the example instance; nodes are labelled by the current partial permutation / c.

3 Assignment of HW4

Your task is to implement branch-and-bound algorithm for Bratley's problem. To pass the evaluation, you should implement all three elimination rules as discussed, otherwise your runtime might be too high.

3.1 Input/Output format

Your program will be called with two arguments: the first one is absolute path to input file and the second one is the absolute path to output file which has to be created by your program.

Let n be the number of tasks. Then the input file has $n + 1$ lines and has the following form

$$
\begin{array}{lll}
n & & \\
p_1 & r_1 & \tilde{d}_1 \\
p_2 & r_2 & \tilde{d}_2 \\
\vdots & & \\
p_n & r_n & \tilde{d}_n
\end{array}
$$

One space is used as a separator between values on one line. All the values in the input file are non-negative integers.

If the input instance is infeasible, then the output file consists of the single line containing -1. On the other hand, if the input instance is feasible, then the output file consists of n lines and has the following form

$$
\begin{array}{l}
s_1 \\
s_2 \\
\vdots \\
s_n
\end{array}
$$

where s_i is the optimal start time of task T_i. All the values in the output file are integers.

Example 1

Input:

```
4
2 4 7
1 1 5
2 1 6
2 0 4
```

Output:

```
5
2
3
0
```

Example 2

Input:

```
3
2 4 7
1 1 2
2 1 2
```

Output:

```
-1
```

Summary

This time, we revised scheduling in general. Afterwards, we concentrated on a single $\mathcal{NP}$ hard problem. Previously, we have designed an ILP model to solve the problem. Now, we came up with a specialized algorithm based on a general branch-and-bound procedure. The practical understanding of the algorithm will be demonstrated by solving the HW4.

References

[1] J. Blazewicz, *Scheduling Computer and Manufacturing Processes*. Springer, second ed., 2001.

[2] R. Graham, E. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan, "Optimization and approximation in deterministic sequencing and scheduling theory: a survey," *Annals of Discrete Mathematics*, vol. 5, pp. 287–326, 1979.