Combinatorial optimization
Lab No. 11
Scheduling and the Branch and Bound algorithm

Industrial Informatics Research Center
http://industrialinformatics.cz/

Abstract

The topic of this seminar is scheduling of tasks on resources. We focus on $1|r_j, \tilde{d}_j|C_{max}$ problem, which will be solved by Bratley's algorithm.

1 Scheduling

Scheduling, generally speaking, deals with assigning tasks to resources in time. Let's have a set of tasks, a set of resources (processors) on which these tasks can be executed, a set of constraints and an optimality criterion. Each task, typically denoted as T_i, is characterized by the time needed by resource to process it - *processing time* p_i. Moreover, task T_i can be specified by the following parameters:

- The time at which task T_i is ready for processing - *release time* r_i,
- A time limit by which task T_i should be completed - *due date* d_i,
- A time limit by which task T_i must be completed - *deadline* $\tilde{d}_i$,

$\alpha|\beta|\gamma$ notation, so called Graham-Blazewicz, is commonly used to describe the scheduling problems [1, 2]:

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

For example, $P2||C_{max}$ is a formal notation of the scheduling problem where two parallel identical processors are available and objective is to minimise the schedule length (even this "simple" problem is $\mathcal{NP}$-hard). 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 utilisation of resources/processors (y-axis).

2 Branch and Bound algorithm

The Branch and Bound (B&B) 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:

a) It does not contain any feasible solution.

b) 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:

a) Breadth-first search.

b) Depth-first search.

In our case - 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.

3 The Bratley's algorithm

An example of the use of the Branch and Bound algorithm in the scheduling is a problem formally denoted as $1|r_j, \tilde{d_j}|C_{max}$. It is scheduling on a single processor while each task has a specified release time and deadline, i.e. a time window in which the task has to be processed. The objective is to minimise the schedule length. This problem also belongs to the $\mathcal{NP}$-hard problems, so unless $\mathcal{P} = \mathcal{NP}$ there is no hope of finding deterministic exact algorithm that runs in polynomial time. The problem can be solved, for example, by *Bratley's algorithm* [1] which is based on the branch and bound algorithm. The goal is to search the solution space in the form of a tree where each end-node (leaf) represents one solution of the problem. Other nodes are partial solutions consisting of some tasks which are ordered. The tree can be pruned on conditions that:

1) Some tasks exceed their deadline.

2) The estimate of the result of the best possible solution in the subtree is worse than the best achieved result so far.

3) A partial solution is optimal.

Ad 1) Let's have node n. If there is no option to add arbitrary task at node n without exceeding its deadline then the partial schedule will never result in the feasible solution. Therefore, all descendants of node n can be eliminated. This follows from the fact that if any of these tasks exceeds its deadline at node n, it will certainly exceed its deadline if scheduled later.
Ad 2) The upper bound of the schedule length has to be calculated at the beginning, e.g. $UB = \max \tilde{d_j}$ for $\forall j$. At each node, the estimate of the best possible final solution (lower bound) based on the current partial solution can be determined as $LB = \max\{\min_{\forall j \in V}\{r_j\}, c\} + \sum_{\forall j \in V} p_j$, where c is the current schedule length (i.e. the completion time of the last task) and V is the set of tasks being not scheduled yet. If $LB \geq UB$ than the particular part of the tree can be eliminated because the further branching of the current node cannot find the optimal solution.
Ad 3) If the *release time property* is satisfied then the partial schedule is optimal and only the current node is expanded further. Going back in the tree of partial solutions cannot return a better schedule. Release time property says that if the length of the partial solution is less than or equal to the lowest value of the release time of hitherto unscheduled tasks, the partial solution is optimal.

An input example of the problem $1|r_j, \tilde{d_j}|C_{max}$ is on the left side in Figure 1. It is a problem with four tasks having their own processing time, release time and deadline. The problem is solved by Bratley's algorithm based on the branch and bound algorithm. The final solution in the form of a Gantt chart is on the right side in Figure 1. The search tree has $4! = 24$ leaves. If input has 10 tasks the number of combinations of ordered tasks (i.e. leaves of the tree) will be 3628800 and in the case of 15 tasks it will be much more - $1.3 \cdot 10^{12}$. So it is clearly seen that the number of possible combinations is already very high for relatively small instances and searching the entire solution space is completely insoluble.

Input:
p=`(2,1,2,2)`
r=`(4,1,1,0)`
$\tilde{d}$=`(7,5,6,4)`

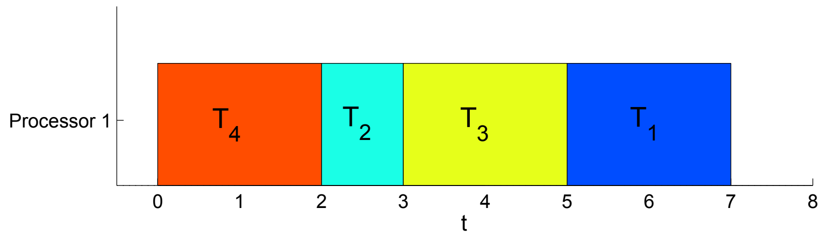

Figure 1: An example of the problem $1|r_j, \tilde{d}_j|C_{max}$ and its solution

In this instance, we will construct a search tree as follows:

1) Create four branches from the root node where each of the resulting node corresponds to the assignment of one task to the schedule at the first position.
2) Branch the previous nodes to three new nodes because one position is already assigned.
3) Branch the previous nodes to two new nodes.
4) At the last step, there is only one branch from previous nodes to the leaves of the tree.

The number of leaves is 24 (see above) and the tree has 5 levels including the root node. The search tree which was pruned based on the above mentioned conditions is depicted in Figure 2. The result in the form of a Gantt chart is on the right side in Figure 1.

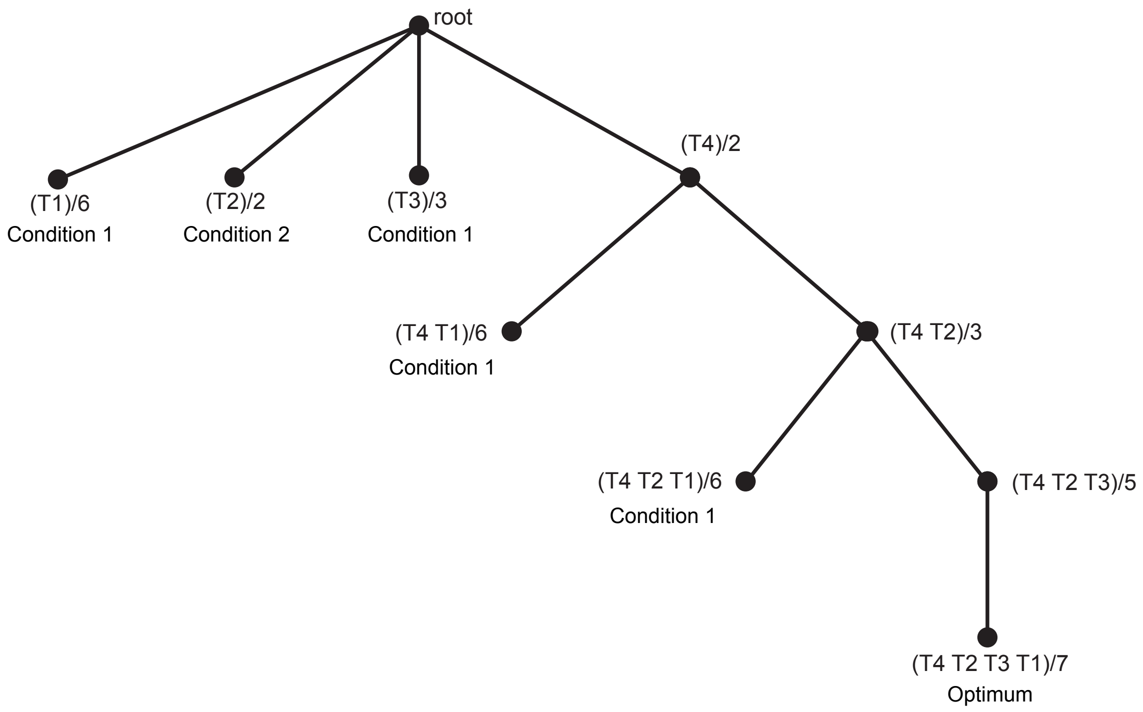

Figure 2: The pruned search tree

A seminar assignment: Let's have this input: p=(3,4,1,2), r=(5,2,1,0), $\tilde{d}$=(10,7,4,3). Solve this problem graphically (as shown in Figure 2) using Bratley's algorithm.

A homework assignment: Implement Bratley's algorithm. The input is given by three vectors — processing time p, release time r and deadline $\tilde{d}$. The output assignment of tasks to start times in an optimal schedule minimizing the makespan C_{max}. Upload your source codes to the Upload System.

Hint: To pass the evaluation successfully, you need to implement all three elimination rules, as given above. Otherwise, the running time of your algorithm will be damn too high!

3.1 Input and 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 (the output file has to be created by your program).

The input file has following format. The first line of the input file consist of a single natural number n giving the number of tasks. Each of the following n lines consists of three positive integers separated by space giving processing time p_i, release time r_i and deadline $\tilde{d}_i$, in this order.

If the input instance is feasible, then the output file consists of n lines. i-th line consists of the single non-negative integer, denoting the start time of task T_i. If the input instance is infeasible, then the output file consists of the single line containing `-1`.

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
```

References

[1] J. Blazevicz, *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.