EXACT ALGORITHM FOR THE PREEMPTED SINGLE MACHINE SCHEDULING PROBLEM

National Research University Higher School of Economics
Moscow, Russia

DIMAS MUNOZ MONTESINOS

May 5th, 2020

1 INTRODUCTION

Combinatorial optimization is a topic where we need to apply mathematical techniques to find optimal solutions within a finite set of possible solutions. For some problems, there are algorithms that are able to find optimal solutions in polynomial time. However, we face NP-hard problems where it may be necessary to search through an exponential number of solutions [4]. A well-known NP-complete combinatorial optimization problem is the 0-1 Knapsack Problem [5].

In the context of this project, we study an application of the Linear Assignment Problem (LAP) as a relaxation of the single machine scheduling problem, minimizing the total weighted completion time within a new scheme of the Branch-and-Bound Alorithm (BnBA). We provide a computational study of all ingredients of our BnBA, namely, the tolerance based Branching Rules (upper [2], lower and bottleneck tolerances [3]) as well as Lower and Upper Bounds (to the unknown optimal values of our problem) improved by the chosen values of tolerances and heuristic based on the Weighted Shortest Remaining Processing Time (WSRPT) rule [1] respectively. Recent discoveries show that we are able to find optimal solutions for the Scheduling Problem (SP) in instances with hundreds of jobs in very few CPU time.

2 PROBLEM FORMULATION

Given $n_{\geqslant 1}$ job instances, we can formalize each job i as follows:

- We split it into p_i parts so, consequently, each job i takes p_i seconds to be processed. Note that we could have a different number of parts for each job (i.e. different processing times).
- It becomes available for processing at a time $r_i \in \mathbb{Z}^+$ (i.e. strictly positive).
- A weight $w_i \in \mathbb{Z}^+$ is assigned to it (which can be seen as a priority factor).
- We assign a completion time $T_{i,j}$ to each job part j. Also, we denote the completion time of the job as C_i, which can be seen as the completion time of the last job part: $C_i = T_{i,p_i}$.

Then, we describe the Single Machine Scheduling Problem as $1|pmtn;p_i;r_i|\sum w_i C_i$ (where 1 states for a single machine). Additionally, we allow preemptions during a job processing, which means that a job may be interrupted in favor of another job.

i	p_i	w_i	r_i
1	2	2	1
2	3	8	2
3	2	10	3

Table 1: Example of Scheduling Problem with $n = 3$ jobs.

Remark that we have to solve an assignment problem of np job parts to np time moments such that every part j of every job i is assigned to a time moment not earlier than its release date r_i. In 1984, it was proved by Labetoulle et al. [6] that the problem $1|pmtn;p_i;r_i|\sum w_i C_i$ is NP-hard.

t	1	2	3	4	5	6	7
$J_{1,1}$	0	0	0	0	0	0	∞
$J_{1,2}$	∞	2	4	6	8	10	12
$J_{2,1}$	∞	0	0	0	0	∞	∞
$J_{2,2}$	∞	∞	0	0	0	0	∞
$J_{2,3}$	∞	∞	∞	8	16	24	32
$J_{3,1}$	∞	∞	0	0	0	0	∞
$J_{3,2}$	∞	∞	∞	10	20	30	40

Table 2: Problem of Table 1 formulated as LAP.

In addition to what we described before, to find a feasible solution, we must satisfy the following constraints:

- Assignment constraints: for the reason that we have one single machine, every part of every job should be assigned to exactly one position of a schedule and every position should be occupied by exactly one job part. Equivalently:

$$T_{u,v} = T_{x,y} \Leftrightarrow u = x \wedge v = y$$

- Sequence constraints: for every job i, its parts $J_{i,1}, \dots, J_{i,p_i}$ in a schedule must stay in a sequence such that a later part cannot be processed before an earlier part:

$$T_{i,x} < T_{i,y} \text{ such that } x < y$$

- Release date constraints: since we assume that every part j of a every job i takes the same amount of time to be processed, we add two constraints: the first job part cannot be assigned to a time $t_{i,1}$ earlier than r_i and the last job part cannot finish earlier than $r_i + p_i - 1$. If we generalize it for every job part $J_{i,j}$, we obtain that each part cannot start earlier than $r_i + j - 1$ and finish later than $T - p_i + j$ (where $T = \sum_{k=1}^{n} p_k$).

Now that we have defined the constraints, the first step to find a solution to SP is to reduce it to LAP. We build a matrix $M_{T\times T}$ (where $T = \sum_{k=1}^{n} p_k$) such that each row represents the cost of assigning a job part $J_{i,j}$ to a time t (see Table 2). To build the matrix, besides the previous constraints, we assign 0 to a cell if it is the row of a job part such that $1 \leqslant j \leqslant p_i - 1$. Otherwise, it is the row of the last job part and the cell value is equal to $w_i t$ (i.e. it grows up with the time).

Now, assuming that there are no idle time intervals, we define the objetive function as minimizing the value of the cost function:

$$f_{LAP} = \sum w_i C_i$$

t	1	2	3	4	5	6	7
$J_{1,1}$	0	0	0	0	0	0	∞
$J_{1,2}$	∞	2	4	6	8	10	12
$J_{2,1}$	∞	0	0	0	0	∞	∞
$J_{2,2}$	∞	∞	0	0	0	0	∞
$J_{2,3}$	∞	∞	∞	8	16	24	32
$J_{3,1}$	∞	∞	0	0	0	0	∞
$J_{3,2}$	∞	∞	∞	10	20	30	40

Table 3: Feasible solution (not optimal) to the example given in Table 2.

In other words, our objetive is to minimize the total weighted completion time. Note that we only need to take into account the last part of each job i when we calculate the total cost.

We show an example of a feasible solution (which is not optimal) in Table 3. Now, we can proceed to implement a solution in the next section.

3 ALGORITHM DESCRIPTION

3.1 Hungarian Algorithm

Our first step should be to reduce the matrix of LAP such that we can find a solution (which may be feasible or not). For this purpose, we use Hungarian Algorithm[1] which, given a matrix $n \times n$, it executes four steps:

1. For each row, find the lowest element and subtract it from each element in that row.
2. Similarly, for each column, find the lowest element and subtract it from each element in that column.
3. Cover all zeros in the resulting matrix using a minimum number of horizontal and vertical lines. If we require n lines, then it exists an optimal assignment among the zeros (thus the algorithm stops). If we require less than n lines, continue to step 4.
4. Find the smallest element (call it k) that is not covered by a line in the step 3, subtract k from all uncovered elements, and add k to all elements that are covered twice. Then, continue to step 3 again.

t	1	2	3	4	5	6	7
$J_{1,1}$	0	0	0	0	0	0	∞
$J_{1,2}$	∞	0	2	12	6	8	0
$J_{2,1}$	∞	0	0	8	0	∞	∞
$J_{2,2}$	∞	∞	0	8	0	0	∞
$J_{2,3}$	∞	∞	∞	0	0	8	6
$J_{3,1}$	∞	∞	0	8	0	0	∞
$J_{3,2}$	∞	∞	∞	0	2	12	12

Table 4: Hungarian Algorithm applied to LAP from Table 2. Note that we can cover all zeros using 7 lines as a minimum.

1 http://www.hungarianalgorithm.com/hungarianalgorithm.php

Once the Hungarian Algorithm finishes, we proceed to make an assignation and, later, validate if it is feasible or not. If it feasible, the algorithm stops and returns the assignation (which it is optimal). Otherwise, it continues with BnBA.

3.2 Branch-and-Bound Algorithm

If the assignation, that we found with the Hungarian Algorithm, is not feasible, then there are at least two job parts which are not processed in the correct order. In other words, we do not satisfy the Sequence Constraint in two parts (x and y) of the job i: $J_{i,x}$ and $J_{i,y}$.

t	1	2	3	4	5	6	7
$J_{1,1}$	0	0	0	0	0	0	∞
$J_{1,2}$	∞	0	2	12	6	8	0
$J_{2,1}$	∞	0	0	8	0	∞	∞
$J_{2,2}$	∞	∞	0	8	0	0	∞
$J_{2,3}$	∞	∞	∞	0	∞	8	6
$J_{3,1}$	∞	∞	0	8	0	0	∞
$J_{3,2}$	∞	∞	∞	0	2	12	12

Table 5: Solution (not feasible) to the result of the Hungarian Algorithm in Table 4. In this case, we exclude the case $T_{2,3} = 5$ by replacing the cell with ∞.

The branching rule implies excluding either $T_{i,x}$ or $T_{i,y}$ in the assignation and compute again the Hungarian Algorithm for each branch. We continue with this process until we find a feasible solution and, once we find it, we proceed to calculate the cost f_{LAP}.

In order to be sure that we obtain the best solution, we need to expand the other branches. Finally, we keep the solution which minimizes f_{LAP}.

3.3 Code implementation

With this report, we attach a Python script where you can find the algorithm that we described. Using this script, we solved the problem given in Table 1 (reduced to LAP):

t	1	2	3	4	5	6	7
$J_{1,1}$	0	0	0	0	0	0	∞
$J_{1,2}$	∞	2	4	6	8	10	12
$J_{2,1}$	∞	0	0	0	0	∞	∞
$J_{2,2}$	∞	∞	0	0	0	0	∞
$J_{2,3}$	∞	∞	∞	8	16	24	32
$J_{3,1}$	∞	∞	0	0	0	0	∞
$J_{3,2}$	∞	∞	∞	10	20	30	40

Table 6: Solution (optimal and feasible) to the example given in in Table 1. The total cost of this assignation is 44.

REFERENCES

[1] M. Batsyn, B. Goldegorin, P. Pardalos, and P. Sukhov, *Online heuristic for the preemptive single machine scheduling problem of minimizing the total weighted completion time* (2014). Optimization Methods & Software, vol. 29, no. 5, pp. 955–963.

[2] Marcel Turkensteen, Diptesh Ghosh, Boris Goldengorin and Gerard Sierksma. *Tolerance based branch and bound algorithms for the ATSP* (2006). European Journal of Operational Research, 2008, 189 (3) 775–788.

[3] Remco Germs a, BorisGoldengorin a,c, MarcelTurkensteen, *Lower tolerance-based Branch and Bound algorithms for the ATSP* (2011). Computers and Operations Research, 39(2012), 291–298.

[4] Michael Garey and David S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (1979). W. H. Freeman and Company.

[5] Hans Kellerer, Ulrich Pferschy and David Pisinger, *Knapsack problems* (2004).

[6] Labetoulle, J., Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., *Preemptive Scheduling of Uniform Machines Subject to Release Dates* (1984). Progress in Combinatorial Optimization, pp. 245–261. Academic Press, Toronto.