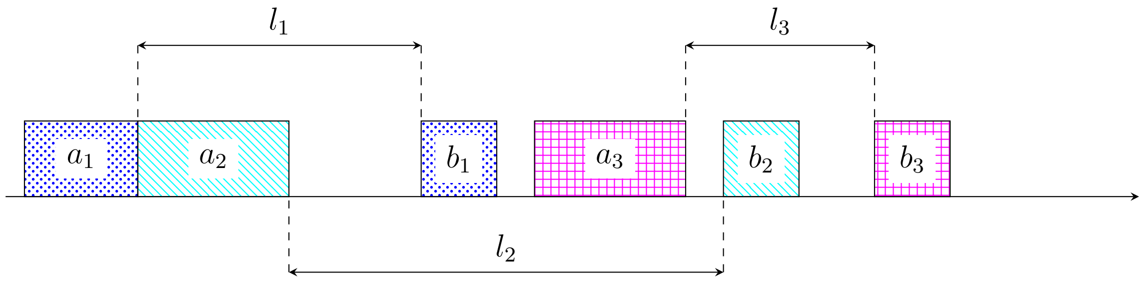
Coupled task scheduling

Anna Markó
Supervisor: Györgyi Péter

2023. december 10.

1 Introduction

The coupled task scheduling problem refers to scheduling n jobs, each consisting of two tasks, on a single machine. The machine can only process one task at a time. Each job is characterized by three parameters: a_j, l_j, and b_j, where a_j represents the processing time of the first task of the job j, b_j represents the processing time of its second task, and L_j denotes the time interval that must elapse between the processing of the two tasks.



A scheduling can be represented as $\sigma = (s_1, s_2, \ldots, s_n)$, where s_j denotes the starting time of the job j. Let C_j represent the completion time of job j. The problem is studied based on various objective functions, with the most investigated being $C_{\max}$, where the objective function is $\max_j C_j$.

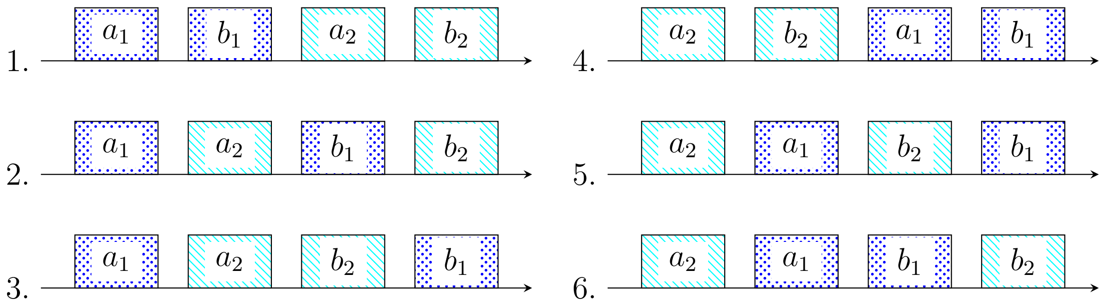
Throughout the semester, I focused on the objective function $\sum C_j$. For a general problem with parameters a_j, l_j, and b_j, it is NP-hard, and currently, no approximation algorithm is known for it. However, there are several variants that are also NP-hard, but approximation algorithms are known for them[**2**]. Examples include:

3-approx	2-approx	1.5-approx
a, l_j, b	$a, l_j, b, b \leq a$	$1, l_j, 1$
a_j, L, b_j	a_j, p_j, p_j	p_j, L, p_j
	p_j, p_j, b_j	p_j, p_j, p_j

My goal was to create an IP solver for the problem with parameters a_j, L, and b_j, and to use it to investigate how effective a 3-approximation algorithm is in practice. Another objective was to search for instances of the problem in which the approximation algorithm produces particularly poor results.

2 IP solver

I used the Sherali and Smith model as the basis for my IP solver[1]. Two jobs can be scheduled in six different ways relative to each other:



Let's assign variables $y_{i,j,k}$ to every pair (i, j) for every k, where $1 \le k \le 6$. These variables represent how the job i and job j are positioned relative to each other in a given schedule. Let s_i denote the starting time of the job i. We define task-dependent constants $r_{i,j,k}$ for every (i, j, k) triplet, which are intended to indicate the conditions that must be satisfied when the relative position of jobs i and j is k. Let M be a sufficiently large constant (an upper bound on the time difference between the starting times of two jobs). Let $t_i := a_i + L + b_i$.

$$
\begin{aligned}
r_{i,j,1} &= M & \qquad r_{i,j,4} &= -t_j \\
r_{i,j,2} &= a_i + b_i - a_j & \qquad r_{i,j,5} &= -a_j \\
r_{i,j,3} &= a_i + b_i - t_j & \qquad r_{i,j,6} &= -a_j
\end{aligned}
$$

Given this information, we may formulate the problem as follows.

$$
\begin{aligned}
\text{Minimize} \quad & \sum_{i=1}^{n} s_i + t_i \\
\text{subject to} \quad & s_j - s_i \le \sum_{k=1}^{6} r_{i,j,k} y_{i,j,k} && 1 \le i \le n, 1 \le j \le n \\
& \sum_{k=1}^{6} y_{i,i,k} = 0 && 1 \le i \le n \\
& \sum_{k=1}^{6} y_{i,j,k} = 1 && 1 \le i \le n, 1 \le j \le n, i \neq j \\
& y_{i,j,1} = y_{j,i,4} && 1 \le i \le n, 1 \le j \le n \\
& y_{i,j,2} = y_{j,i,5} && 1 \le i \le n, 1 \le j \le n \\
& y_{i,j,3} = y_{j,i,6} && 1 \le i \le n, 1 \le j \le n \\
& y_{i,j,k} \in \{0, 1\} && 1 \le i \le n, 1 \le j \le n \\
& s_j \in \mathbb{Z}_+ && 1 \le i \le n
\end{aligned}
$$

In the case of the (a_j, L, b_j) problem, only four out of the outlined six cases are possible, as the third and sixth cases are not feasible. Accordingly, the integer programming (IP) formulation simplifies. I implemented the IP in Python, initially using the Python MIP package. Unfortunately, with this package, I could only solve problems with a small number of jobs. Therefore, I switched to the Gurobi optimization solver.

3 Approximation algorithm

For the (a_j, L, b_j) problem, David Fisher and Péter Györgyi published a 3-approximation algorithm in their recent work[**2**], and I have implemented it.

Algorithm 1:

Input : $(a_j, b_j) \quad j = 1 \dots n,\ L$
Output: $(s_j)_{j=1}^{n} \quad j = 1 \dots n$

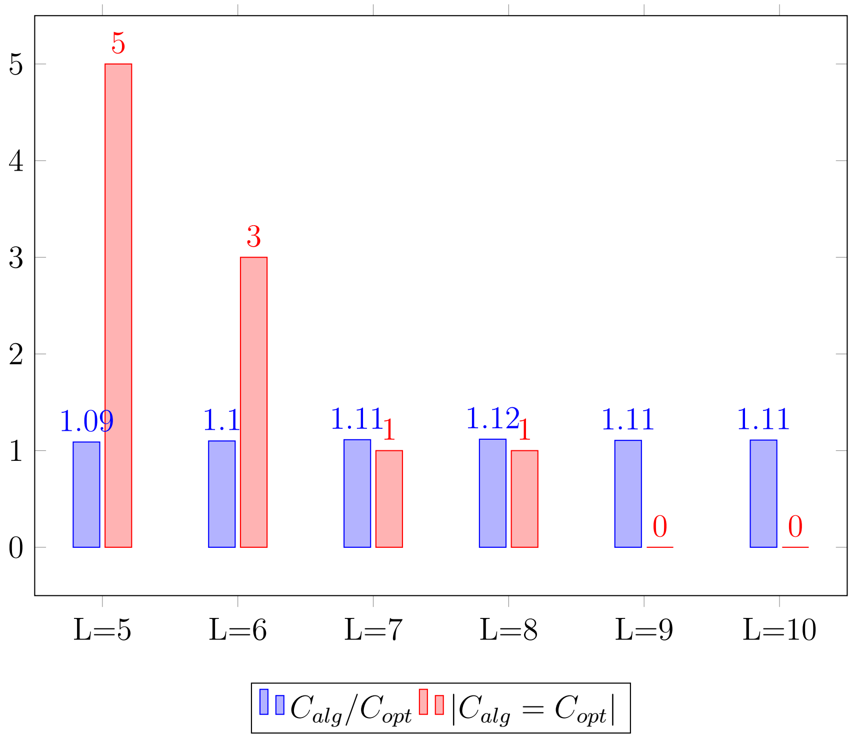
Sort the jobs in non-decreasing order of $a_j + b_j$;
$s_1 := 0$;
for $j = 2 \dots n$ **do**
    if *a_j can be scheduled immediately after a_{j-1} without overlapping into the processing time of other tasks* **then**
        Schedule it this way;
        $s_j := s_{j-1} + a_{j-1}$
    else
        if *b_j can be scheduled immediately after b_{j-1} without overlapping into the processing time of other tasks* **then**
            Schedule it this way;
            $s_j := s_{j-1} + a_{j-1} + b_{j-1} - a_j$
        else
            Start a_j immediately after b_{j-1};
            $s_j := s_{j-1} + a_{j-1} + b_{j-1} + L$;

---

4 Results

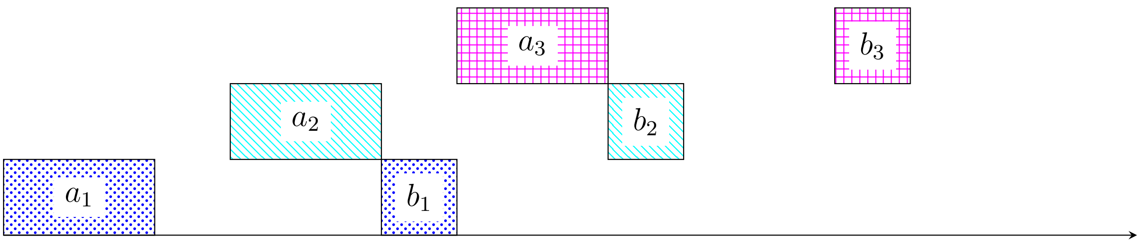
Unfortunately, my IP solver can currently only handle tasks with a maximum of 7 jobs within an acceptable time frame, so I worked with such inputs. I ran both my IP solver and the approximation algorithm for 50 inputs in 6 different cases based on L, where I randomly selected a_j and b_j values between 1 and 10, and L took the values 5, 6, 7, 8, 9, and 10. In the following bar chart, you can see the results of these tests. I denoted the optimum as C_{opt} and the result obtained by the algorithm as C_{alg}. For each of the six cases, I plotted the average values of C_{alg}/C_{opt} and the number of cases where the two values coincided.

It can be observed that the average values have become very similar. Out of the 300 inputs on which I tested, in none of the cases did the C_{alg}/C_{opt} ratio reach 1.25.

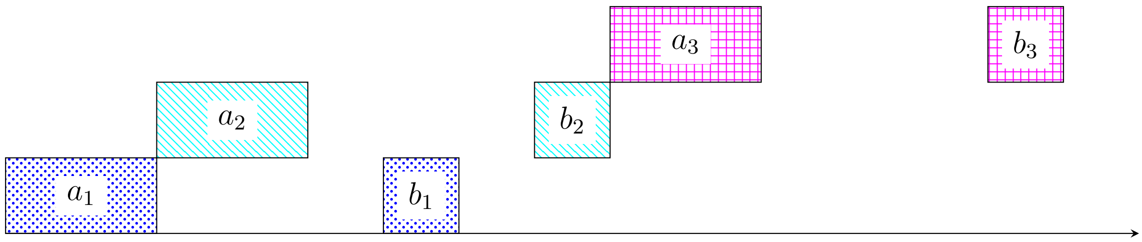


In a given schedule, jobs form a block if their starting times directly follow each other, and for any two directly following jobs, it holds true that the later scheduled job starts earlier than the earlier one would have finished. I examined how blocks are formed in the inputs where the C_{alg}/C_{opt} ratio is relatively high. I noticed that the algorithm often produces worse results even when the order of job starting times is the same in both schedules. This can occur when it is worthwhile to delay a job by an amount of time in a way that it still forms a block with the previous job, and the next job can still fit into the block. So, for the case of three consecutive jobs, let b_2 start later than the completion of b_1 by an amount that allows a_3 to fit between the two tasks. The following diagrams illustrate this situation, where the processing time of the first part of each job is 2, the processing time of the second part is 1, and L is 3 (the arrangement is for the clarity of the blocks):

The optimal schedule:



The schedule obtained by the algorithm:



It is noticeable that when scheduling n jobs with the same parameters, in the optimal schedule, each job forms a block, while in the schedule obtained by the algorithm, every other job starts a new block.

For this task, I managed to achieve the highest value for the C_{alg}/C_{opt} ratio. The obtained ratio for n jobs is:

$$\frac{2(n+1)(2n+1)}{3n(n+3)} \approx \frac{4}{3}.$$

5 Further plans

In the next semester, I plan to enhance my IP solver to handle tasks with multiple jobs within an acceptable time frame, possibly experimenting with multiple models. In addition, I aim to make further progress in identifying instances where the algorithm significantly underperforms.

References

[1] D. Fischer, P. Györgyi: *Approximation algorithms for coupled task scheduling minimizing the sum of completion times* Ann. Oper. Res. 328(2): 1387-1408 (2023)

[2] Hanif D. Sheralia, J. Cole Smith. *Interleaving two-phased jobs on a single machine.* Discrete Optimization 2 348 – 361 (2005)