Branch and Bound Coursework

Oussama Rakye Abouelksim (or22), Stefanos Ioannou (si122)

December 26, 2022

1 Question 1 and 2

Precedence constraints have been handled with a table of dependencies, where if $G(i, j) = 1$ if and only if there exists an edge from node i to j. As we are allocating jobs from the end of the schedule, job i can only be allocated if all jobs that are preceded by i are allocated. Every job can only be allocated if it does not precede any other job or if $G(i, j) = 1 \Rightarrow j \in S, \forall j$ where S is the partial schedule. Therefore, all jobs that are preceded by i are already allocated and i can then be allocated.

Table 1 shows the mean and Standard Deviation of the processing times, produced by *gettimes.py*. To create partial solutions, during the branch-and-bound (BNB), the search space of the algorithm is limited to jobs that are feasible given the current schedule, with the exception of the very first job, i.e. job 31. To create a complete and feasible solution, the longest schedule maintained throughout BNB[1], is used as a partial solution if it reaches the maximum amount of 30,000 iterations. If a missing job is feasible given the longest schedule, then it is appended to the start of the schedule. If at any point, more than one missing job is feasible, i.e. they can both be added to the schedule, only the job with the highest due date is added. This process continues until all jobs are scheduled.

At the first iteration, the first node is (31) with tardiness 0. At the second iteration, the current node is (1, 31) with tardiness 39.0220 seconds (4 d.p.). At the penultimate and last iteration, the current node is (29, 15, 14, 28, 13, 11, 27, 26, 25, 12, 24, 5, 2, 1, 31) and (29, 14, 28, 13, 11, 27, 26, 25, 12, 24, 5, 2, 1, 31) respectively. Both have a total tardiness of 368.6644 seconds (4 d.p.).

The best feasible schedule found overall is (30, 4, 3, 23, 22, 21, 10, 9, 8, 7, 6, 20, 19, 18, 17, 16, 14, 29, 28, 27, 26, 25, 24, 15, 13, 12, 11, 5, 2, 1, 31) with total tardiness of 423.3319 seconds (4 d.p.). This is not the global optimum solution as the schedule (30, 4, 3, 23, 22, 21, 10, 9, 8, 7, 6, 14, 20, 19, 18, 17, 16, 29, 28, 27, 26, 25, 24, 15, 13, 12, 11, 5, 2, 1, 31) achieves better tardiness at 391.451 seconds.

The above implementation takes approximately 2.18 seconds to terminate on a personal machine. The maximum size of the pending list is 26154 nodes.

The heuristic schedule used is (30, 4, 10, 23, 3, 9, 20, 22, 8, 19, 21, 7, 18, 29, 6, 17, 27, 28, 14, 16, 26, 13, 15, 25, 11, 12, 24, 5, 2, 1, 31), with total tardiness of 594.4855. Table 2 shows the response time of the heuristic schedule and BNB. The schedule generated with the heuristics, even though is faster to generate, has a final tardiness 40.43% higher than using BNB.

2 Question 3

The unbounded version of the branch-and-bound algorithm, finds the optimal solution after 73856 iterations, with the (global optimal) tardiness of 542 seconds.

Several alternative methods were considered to decrease the number of iterations. Normally, a complete A^* search finds the optimal solution using an admissible heuristic, i.e. one that always underestimates the cost to the goal, to get closer to the optimal solution decreasing the number of iterations needed. Considering A^* search, multiple heuristics have been tried, however, no admissible heuristic that sufficiently reduced the number of iterations has been found. Using an inadmissible heuristic may speed up searching, however, it is not guaranteed to be complete. For instance, approximating the total schedule with the current tardiness, so that the value of a node is $v_i = T_i * \frac{total\_number\_jobs}{|S_i|}$ where $|S_i|$ is the number of jobs in the sub-schedule. This produces a solution of total tardiness 561 and in 32 iterations. The latter heuristic adds a lot of weight to the heuristic, so a weighted heuristic has

[1] If two incomplete schedules have the same length, maintain the schedule with the least tardiness.

been introduced namely $v_i = T_i + \alpha T_i * \frac{total\_number\_jobs - |S_i|}{|S_i|}$, where α is the weight of the heuristics. Note that if $\alpha = 1$, then both formulas are equivalent. Experimenting with different values of alphas, $\alpha = 0.075$ gives a total tardiness of 542 in 16814 iterations. Evidently, this heuristic establishes a trade-off between the total tardiness and the number of iterations, i.e. increasing α decreases the amount of iterations, but also decreases the final tardiness. In order to improve the estimation of the final tardiness, Moore-Hodgson has been used to calculate the number of tardy jobs in the set of unscheduled jobs, namely using $v_i = T_i + U_i * \frac{T_i}{|S_i|}$. This approach returns a total tardiness of 555 after 46 iterations. Beam-width with node priorities has also been implemented. However, nodes are discarded according to an estimation, and the optimal solution may be skipped in this process.

To further improve the solution, the upper bound on the total tardiness of trial solutions was used. Depth First Search (DFS) is used to search for the optimal schedule. DFS pops a node from the stack, generates its children and puts the children back into the stack. The algorithm maintains an upper bound on the tardiness of trial solutions. The children of a node are added if and only if the total tardiness of the children is lower than the upper bound. Fathoming is performed on the stack, to remove any nodes with total tardiness higher than the upper bound. Once the stack is empty, i.e. there are no more nodes to expand, the full schedule with the lowest tardiness found is returned. Moreover, this implementation features two main improvements that reduce the search space and guide the search closer to the optimal solution. When expanding nodes, the children are added to the stack greedily based on total tardiness. Therefore, it applies DFS prioritizing the nodes with the lowest tardiness. We define that a sub-schedule S_i dominates over a S_j if and only if all jobs in S_i are scheduled in S_j, all jobs in S_j are scheduled in S_i and S_i has lower or equal tardiness than S_j. This implies that the ordering defined in the dominant sub-schedule is better or equal to the dominated schedule. Hence, all dominated schedules can be rejected and pruned as they will give no better schedule than the dominating one. This is performed by storing the dominant sub-schedules in a table and updating every time a new or better sub-schedule is found. These improvements guarantee an optimal solution as nodes will only be pruned if they are equally bad or worse, i.e. have higher total tardiness than the upper bound for that sub-schedule. In addition, this solution uses no admissible heuristic. This returns (30, 4, 3, 10, 9, 14, 20, 19, 23, 22, 21, 18, 8, 7, 6, 17, 16, 29, 28, 27, 26, 25, 24, 15, 11, 13, 12, 5, 2, 1, 31), with the optimal tardiness of 542 after 233 iterations. The runtime of the algorithm is 0.032 seconds.

A Appendix

Type	**Mean and Standard Deviation of Processing Time, s**
vii	18.8566 ± 0.7569
blur	5.7161 ± 0.2901
night	22.5914 ± 0.4856
onnx	2.9634 ± 0.1677
emboss	1.7741 ± 0.6133
muse	14.6653 ± 0.4502
wave	11.0234 ± 0.6386

Table 1: Mean and Standard Deviation of the Processing Times of each Node Type produced for **gettimes.py** script

	Branch and Bound	**Heuristic Scheduling**
Completion Time of Workflow 1:	226.5676 ± 0.7953 s	226.0162 ± 6.7350
Tardiness of Workflow 1:	432.2419 ± 5.7955 s	501.0174 ± 36.7666
Script finish time:	0:11:20.795608	0:11:19.143694

Table 2: Shows the Completion, Tardiness and Script Finish Time of the branch-and-bound algorithm (BNB) and the Heuristic Search.