The Total Tardiness Scheduling Problem

Huy Nguyen - CS 6420 - WSU

Consider any resource, such as a classroom, a person's day, a vehicle or a machine, etc., there are a set of independent input jobs that require processing sequentially by the resource. The jobs are equal release, meaning they are all available by the time of scheduling[4]. The schedule is non-preemptive; Preemption in scheduling is to interrupt a running job j_1, run another job j_2, and the remaining of j_1 will be run later.

At the time of arrival, each job has a deadline d_i and a processing time p_i, which are specifically given by external application or the system designer[1]. The deadline is a fixed absolute time point in the future, by which the job j_i must finish. The processing time p_i is the amount of time it takes to finish processing job j_i[6]. Once a job is scheduled, the completion time C_i is known; The job j_i starts to run at the time it is scheduled, takes the amount of time p_i and finishes at the completion time C_i[5].

Given the set of jobs $\{j_1, j_2, .., j_n\}$, the deadline $\{d_1, d_2, .., d_n\}$, and the processing time $\{p_1, p_2, .., p_n\}$, after obtaining a certain schedule S, there are the completion times $\{C_1, C_2, \ldots, C_n\}$. The tardiness of any scheduled job is defined as $T(j_i, S) = max(0, C_i - d_i)$. For n jobs scheduled, the Total Tardiness of the schedule is the sum of each job's tardiness, $TT(S) = \sum_{i=1}^{n} T(j_i, S)$. The problem of total tardiness(TTP) is to schedule the given jobs in the processing order that minimizes the $TT(S)$. The schedule with the smallest TTP is called the optimal schedule[5].

For this particular problem, the schedule is non-preemptive because [7] shows that for a single machine, optimal preemptive schedule has the same TT compared to optimal non-preemptive one. That is not the case for multiple machine such as multi-core machine, or parallel machine, though[7].

The nature of the TTP is combinatorial. Selecting the order for n independent jobs to run sequentially makes the number of choices to be in the realm of factorial. Running the search in factorial time is almost impossible for a large input set. [5] shows that the TTP is NP-hard, and there is an optimal solution to any set of input jobs. Being an NP-complete problem means that it theoretically is possible to check whether or not there is a global optimal schedule for a set of inputs using a Nondeterministic Turing Machine in polynomial time. The solution can then be verified by a Deterministic Turing Machine in polynomial time[8, chapter 2].

Multiple attempts have been made in optimizing algorithms for finding optimal output running in polynomial time. There are exact solution algorithms and heuristic solution algorithms. Dynamic programming(DP) and Branch-and-bound($B\&B$) are algorithms for exact solutions. There has been extensive research in both DP and $B\&B$ to reduce the running time. For DP, Lawler proposed the powerful algorithm to run with the worst case of $O(n^4P)$, where n is the number of jobs and $P = \sum p_i$[9].

$B\&B$ is the more powerful and widely used algorithm. It can be used for many other combinatorial problems and has been adapted to solve the TTP. The idea of $B\&B$ is that it will actually go and construct all possible combination of the given set of jobs. It prevents the factorial explosion by pruning branches that violate some certain well-defined lower bounds. It is hard to define the complexity of $B\&B$ because it depends on the input jobs as well as the efficiency of the bounding algorithm[10]. The most efficient B&B version is the hybrid between itself and DP, where $B\&B$ decomposes the problem into subproblems until they are small enough to run DP. However, even the most efficient approach can only run as many as 500 jobs[11].

Due to the high complexity of the exact solution, Heuristic solutions are proposed to provide local optimal solutions. The tardiness property makes it hard to calculate the performance of any heuristic algorithm[6]. However, heuristic approaches runs in much lower polynomial times, which is much more practical and can be designed for certain purposes and different systems[4]. [4] also categorizes Heuristic algorithm into Construction heuristic, local search heuristic and decomposition heuristics.

Construction Heuristic builds the schedule by making n pass through the unscheduled jobs, each pass it picks one suitable job and adds it into the schedule; Two approaches of this heuristic method is Shortest Processing Time and Earliest Deadline. Local Search Heuristic starts with a feasible solutions(such as one given by Earliest Deadline), then it uses other techniques such as swapping next-by jobs, moving job, or reversing short sequence, to find other sequences with lower Total Tardiness. Decomposition Heuristic is a Divide-and-Conquer technique, where it breaks the problem down into sub-problem and solves each of the sub-problems independently. All three approaches have the complexity of $O(n^2)$, which is two order of complexity lower than the most effective DP approach[4].

In conclusion, TTP is the problem in scheduling, aiming to minimize the tardiness of the schedule of a set of$n$ input jobs. TTP has been proven to be NP-hard. For a single resource(machine), there is no different in having preemption in scheduling as compared to non-preemption. It is NP-hard, so the exact algorithms such as DP and $B\&B$, despite extensive research to lower the upper bound, remain impractical. Heuristic approaches are varied and provides different local optimal solutions to different objectives. It is hard to give the performance to any of the heuristic approaches.

References

1. Effah, E., Yussiff, A.-L., Darkwah, A. A., Ephrim, L., Seyram, A., MacCarthy, C., Ganyo, R. M., Aggrey, G., & Aidoo, J. K. (2025). *Exploring the Landscape of CPU Scheduling Algorithms: A Comprehensive Survey and Novel Adaptive Deadline-Based Approach*. *23*(1).
2. Zouaoui, S., Boussaid, L., & Mtibaa, A. (2016). CPU scheduling algorithms: Case & comparative study. *2016 17th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering (STA)*, 158–164. https://doi.org/10.1109/STA.2016.7951997
3. Gonzalez, M. J. (1977). Deterministic Processor Scheduling. *ACM Computing Surveys*, *9*(3), 173–204. https://doi.org/10.1145/356698.356700
4. Koulamas, C. (1994). The Total Tardiness Problem: Review and Extensions. *Operations Research*, *42*(6), 1025–1041. https://doi.org/10.1287/opre.42.6.1025
5. Du, J., & Leung, J. Y.-T. (1990). Minimizing Total Tardiness on One Machine Is NP-Hard. *Mathematics of Operations Research*, *15*(3), 483–495.
6. Lee, I. S. (2013). Minimizing total tardiness for the order scheduling problem. *International Journal of Production Economics*, *144*(1), 128–134. https://doi.org/10.1016/j.ijpe.2013.01.025
7. McNaughton, R. (1959). Scheduling with Deadlines and Loss Functions. *Management Science*, *6*(1), 1–12. https://doi.org/10.1287/mnsc.6.1.1
8. Garey, M. R., & Johnson, D. S. (1990). *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co.
9. Lawler, E. L. (1977). A “Pseudopolynomial” Algorithm for Sequencing Jobs to Minimize Total Tardiness*. In P. L. Hammer, E. L. Johnson, B. H. Korte, & G. L. Nemhauser (Eds.), *Annals of Discrete Mathematics* (Vol. 1, pp. 331–342). Elsevier. https://doi.org/10.1016/S0167-5060(08)70742-8
10. Tyagi, N., & Chandramouli, R. P. T. and A. B. (2016). Single Machine Scheduling Model with Total Tardiness Problem. *Indian Journal of Science and Technology*, *9*(37), 1–14. https://doi.org/10.17485/ijst/2016/v9i37/97527
11. Szwarc, W., Grosso, A., & Croce, F. D. (2001). Algorithmic paradoxes of the single-machine total tardiness problem. *Journal of Scheduling*, *4*(2), 93–104. https://doi.org/10.1002/jos.69