34

Minimizing Total Weighted Error for Imprecise Computation Tasks and Related Problems

Joseph Y-T. Leung
New Jersey Institute of Technology

34.1 Introduction

Scheduling problems with due date related objectives are usually concerned with penalties such as the weighted number of late jobs (i.e., $\sum w_j U_j$), or the weighted amount of time between the completion time of the late job and its due date (i.e., $\sum w_j T_j$). In some applications, however, it is more meaningful to consider penalties involving the weighted number of tardy units (i.e., the weighted number of time units that are late), regardless of how late these units are. This is the case, for example, in a computerized control system, where data are collected and processed periodically. Any data that are not processed before the arrival of the next batch will be lost, and the lost data will have a negative effect on the accuracies of the calculations that are used to control the real-time process. Another example can be found in processing perishable goods, such as harvesting. In this case, jobs represent different stretches of land that need to be harvested. Because of differences in climate and soil conditions and crop culture, the different stretches need to be harvested during different time periods. Crops will perish after its due date, which will cause financial loss. In this application, minimizing the weighted number of tardy units is more meaningful than the other objectives.

Blazewicz [1] was the first to study this problem. He formulated the problem as follows. We are given m parallel processors and n jobs. Each job j has a ready time r_j, due date d_j, processing time p_j, and weight w_j. With respect to a schedule, a job is said to be *late* if it is completed after its due date; otherwise, it is said to be *on-time*. The number of tardy units of job k is the amount of processing of job k done after its due date, and is denoted by Y_k. The problem is to find a schedule such that the total weighted number of tardy units (i.e., $\sum w_j Y_j$) is minimized. As an extension of the $\alpha \mid \beta \mid \gamma$ notation, we denote the nonpreemptive

1-58488-397-9/$0.00+$1.50

version of this problem as $P \mid r_j \mid \sum w_j Y_j$ and the preemptive version as $P \mid pmtn, r_j \mid \sum w_j Y_j$. For the unweighted case, the above two problems will be denoted by $P \mid r_j \mid \sum Y_j$ and $P \mid pmtn, r_j \mid \sum Y_j$, respectively.

The Imprecise Computation Model [2–5] was introduced in the real-time systems community to allow for the trade-off of the accuracy of computations in favor of meeting the deadline constraints of jobs. In this model, each job is logically composed of two subjobs, mandatory and optional. The optional subjob cannot start until the mandatory subjob has finished execution. Mandatory subjobs are required to complete by their deadlines, while optional subjobs can be left unfinished. If a job has an unfinished optional subjob, it incurs an error equal to the processing time of its unfinished portion. The goal is to find a schedule that minimizes the total weighted error. The Imprecise Computation Model is particularly suitable to model iterative algorithms, where the mandatory subjob corresponds to the work needed to set up an initial solution and the optional subjob corresponds to the iterations used to improve the quality of the solution. In this model, it is clear that mandatory subjobs must finish by its deadline, while optional subjobs need not.

Each imprecise computation job j is represented by two subjobs: mandatory (M_j) and optional (O_j). Both jobs have ready times r_j, due date d_j, and weight w_j. M_j has processing time m_j while O_j has processing time o_j. Let $p_j = m_j + o_j$.

The problem of minimizing the total weighted number of tardy units is a special case of imprecise computation in which the processing times of the mandatory subjobs are zero. For a single processor, an algorithm for the total weighted number of tardy units can be used to solve the imprecise computation problem. This can be done by setting the weight of each mandatory subjob to be higher than any optional subjob. Because the mandatory subjobs have higher weights than any optional subjob, the mandatory subjobs are guaranteed to complete (if there is a feasible schedule to complete all the mandatory subjobs). Note that this method cannot be used in multiprocessor systems since a job's mandatory subjob and optional subjob cannot be executed in parallel on different processors.

Blazewicz [1] developed a linear programming solution for the problem $P \mid pmtn, r_j \mid \sum w_j Y_j$, thereby establishing polynomial time complexity of the problem. He also extended the linear programming approach to solve $Qm \mid pmtn, r_j \mid w_j Y_j$. However, the algorithm for uniform processors is only polynomial time for each fixed m. Later, Blazewicz and Finke [6] formulated a minimum-cost-maximum-flow solution to solve both $P \mid pmtn, r_j \mid \sum w_j Y_j$ and $Q \mid pmtn, r_j \mid \sum w_j Y_j$. Using Orlin's $O(|A| \log |V| (|A| + |V| \log |V|))$-time algorithm for the minimum-cost-maximum-flow problem [7], where A and V denote the edge set and vertex set, respectively, $P \mid pmtn, r_j \mid \sum w_j Y_j$ and $Q \mid pmtn, r_j \mid \sum w_j Y_j$ can be solved in $O(n^4 \log n)$ and $O(m^2 n^4 \log mn + m^2 n^3 \log^2 mn)$ times, respectively. These algorithms will be described in the next section.

For a single processor, Hochbaum and Shamir [8] gave an $O(n \log n)$-time algorithm for $1 \mid pmtn, r_j \mid \sum Y_j$ and an $O(n^2)$-time algorithm for $1 \mid pmtn, r_j \mid \sum w_j Y_j$. Later, Leung et al. [9] gave an even faster algorithm for $1 \mid pmtn, r_j \mid \sum w_j Y_j$ that runs in $O(n \log n + kn)$ time, where k is the number of distinct weights. We shall describe this algorithm in the next section.

Potts and van Wassenhove [10] gave an $O(n \log n)$-time algorithm for $1 \mid pmtn \mid \sum Y_j$ and showed that $1 \mid\mid \sum Y_j$ is NP-hard in the ordinary sense. They also gave a pseudo-polynomial time algorithm for $1 \mid\mid \sum Y_j$. Based on the pseudo-polynomial time algorithm, they later gave two fully polynomial approximation schemes for this problem [11].

In the real-time community, Chung et al. [12] gave a network flow approach to solve the total error problem for imprecise computation; their algorithm runs in $O(n^2 \log^2 n)$ time. For a single processor, Shih et al. [13] gave an $O(n^2 \log n)$-time algorithm for the weighted case, and an $O(n \log n)$-time algorithm for the unweighted case.

In [13], Shih et al. proposed an added constraint (called the 0/1-constraint) to be put on the Imprecise Computation Model, where each optional subjob is either fully executed or entirely discarded. This added constraint is motivated by some applications. For example, many jobs can be solved by either a fast or a slow algorithm, with the slow algorithm producing better quality results than the fast one. Due to deadline constraints, it might not be possible to execute the slow algorithm for every job. The problem of scheduling

jobs with primary (slow algorithm) and alternate (fast algorithm) versions can be transformed into one of scheduling with 0/1-constraint [14]. The processing time of the mandatory subjob is the processing time of the fast algorithm, while the processing time of the optional subjob is the difference between the processing times of the slow algorithm and the fast one.

With the 0/1-constraint, two problems were proposed in [13]: (1) minimize the total error and (2) minimize the number of *imprecisely scheduled* jobs (i.e., jobs whose optional subjobs are discarded). For a single processor, Shih et al. [13] showed that minimizing the total error is NP-hard and minimizing the number of imprecisely scheduled jobs is polynomial-time solvable if the optional subjobs have identical processing times. Ho et al. [15] later showed that minimizing the total error is solvable in pseudo-polynomial time, while minimizing the number of imprecisely scheduled jobs can be solved in $O(n^9)$ time.

Motivated by the computational complexity of the problems, Ho et al. [15] proposed two approximation algorithms, one for minimizing the total error and the other for minimizing the number of imprecisely scheduled jobs. Both algorithms have time complexity $O(n^2)$. The one for minimizing the total error has a worst-case bound of 3, which is tight. The one for minimizing the number of imprecisely scheduled jobs has a worst-case bound of 2, which is also tight. Interestingly, the number of precisely scheduled jobs in an optimal schedule is also at most twice the number produced by the algorithm. Both algorithms will be described in Section 34.3.

The problem of minimizing the total weighted number of tardy units has been extended to flow shops and open shops [16–18]. Blazewicz et al. [17] considered the problem $F2 \mid d_i = d \mid \sum w_j Y_j$; see also [18]. They showed that the problem is NP-hard in the ordinary sense and gave a pseudo-polynomial time algorithm for it. Blazewicz et al. [16] also considered the open shop problem. They showed that $O \mid pmtn, r_j \mid \sum w_j Y_j$ and $O2 \mid d_j = d \mid \sum Y_j$ are both polynomial-time solvable, while $O2 \mid d_j = d \mid \sum w_j Y_j$ is NP-hard in the ordinary sense. We shall describe these results in Section 34.4.

Minimizing the maximum weighted number of tardy units has also been studied. Ho et al. [19] gave an $O(n^2)$-time algorithm for a single processor and an $O(n^3 \log^2 n)$-time algorithm for multiprocessors. They also considered other dual criteria optimization problems [19]. These results are described in the next chapter.

34.2 Total Weighted Tardy Units

In this section we shall concentrate on identical and uniform processors. We first show that $1 \mid\mid \sum Y_j$ is NP-hard in the ordinary sense; see also [10]. We then give an $O(n \log n + kn)$-time algorithm for $1 \mid pmtn \mid \sum w_j Y_j$, where k is the number of distinct weights; see also [9]. Finally, we describe a network flow approach to solve $P \mid pmtn \mid \sum w_j Y_j$ and $Q \mid pmtn \mid \sum w_j Y_j$ as well as for imprecise computation jobs; see also [6,12].

Theorem 34.1

$1 \mid\mid \sum Y_j$ is NP-hard in the ordinary sense.

Proof

We shall reduce the Partition problem (see Chapter 2 for a definition) to $1 \mid\mid \sum Y_j$. Given an instance $A = (a_1, a_2, \ldots, a_n)$ of the Partition problem, we construct an instance of $1 \mid\mid \sum Y_j$ as follows. There are $n+1$ jobs. Job j, $1 \leq j \leq n$, has processing time a_j and due date $B = \frac{1}{2}\sum a_j$, while job $n+1$ has processing time B and due date $2B$. The threshold for $\sum Y_j$ is B.

Since the first n jobs have (common) due date B and job $n+1$ has due date $2B$, by a simple interchange argument, we may assume that the completion time of job $n+1$ in an optimal schedule is $2B$ or later. The total tardy units from the first n jobs is at least B, since their common due date is B and their total processing times is $2B$. Thus, if we have a schedule with $\sum Y_j \leq B$, then there must be no tardy units from job $n+1$ which implies that it completes at time $2B$. This means that the total tardy units from the