International Journal of Pure and Applied Mathematics
Volume 51 No. 2 2009, 157-162

Invited Lecture Delivered at
Fifth International Conference of Applied Mathematics
and Computing (Plovdiv, Bulgaria, August 12–18, 2008)

ON COMMON PROPERTIES OF TWO SCHEDULING PROBLEMS WITH DIFFERENT OBJECTIVE CRITERIA

Nodari Vakhania
Science Faculty
State University of Morelos
Av. Universidad 1001, Cuernavaca, 62210, Morelos, MEXICO
e-mail: nodari@uaem.mx

Abstract: We consider two scheduling problems with release times and due dates. In the first problem our objective is to minimize the total job completion time, whereas in the second problem we aim to minimize the number of late jobs. We show that two problems have inherent common properties which can lead to efficient common solution methods for these problems.

AMS Subject Classification: 68Q25, 90B35
Key Words: scheduling algorithms, time complexity, due-date

1. Introduction

Scheduling problems arise in various practical circumstances. Examples of such problems are job shop problems in industry, scheduling of information and computational processes, traffic scheduling and servicing of cargo trains, ships, airplanes. There are scheduling problems of diverse types and different complexities. Saying generally, one deals with two initial notions: *job* (or *task*) and *machine* (or *processor*). A job is a part of the whole work to be done, a machine is the means for the performance of a job. Each job has its *processing time*, i.e.

Received: August 14, 2008

it needs a prescribed time on a machine, and a machine cannot handle more than one job at a time. The *due date* d_j is the desirable completion time for job j, and the *release time* r_j is the time moment when job j becomes available (it cannot be processed before). In some practical circumstances it is important to complete each job by some fixed given date as otherwise this job looses its importance, and not all the jobs become available simultaneously. The *delivery time* q_j of job j is the time needed to job j for the total completion once its processing is complete on the machine. So q_j is an extra time that requires job j and it needs no processor time. The delivery times have the following obvious practical interpretation: once a job is processed and the product is ready it must be delivered to a customer. Job *preemptions* might be allowed: a job can be interrupted on a machine and later resumed on the same or on a different machine.

The set of *feasible solutions* is defined by the restrictions on how the jobs can be scheduled on the machines. Although the number of such solutions is typically finite, it is not always easy to define a subset of feasible solutions of reasonable size that contains an optimal solution. We aim to arrange an order in which the jobs are handled by the machines to minimize basic due-date related criterions (*objective functions*), which are the minimization of the maximal job lateness, i.e., the difference between the completion time of the job and its due date denoted by L_{max}, and the minimization of the number of late jobs, i.e., jobs completed after their due dates, denoted by $\sum U_j$ (U_j is a 0-1 function taking the value 1 iff the corresponding job is late, otherwise it is 0).

Here we concentrate our attention to the following two scheduling problems with equal processing time p. Problem (1) is that of scheduling n (equal-length) jobs with release times and due dates on m parallel identical processors to minimize the number of late jobs, commonly abbreviated as $P/p_j = p, r_j/\sum U_j$. The complexity status of this problem was yet open, though we believe that it can be solved in a polynomial time. In problem (2) n (equal-length) jobs have release times and delivery times and have to be scheduled also on m parallel identical machines. The *completion time* of job j in S is $c_j^S = t_j^S + p$, where t_j^S is the corresponding starting time, and the *full completion time* of j in S is $\mathcal{C}_j^S = c_j^S + q_j$. Our objective is to find a feasible schedule S that minimizes the maximal full job completion time $|S| = \max_j \mathcal{C}_j$, the so-called *makespan*. This problem is abbreviated as $P/r_j, q_j, p_j = p/C_{\max}$ and can also be reformulated in terms of due-dates with the objective to minimize L_{max}, $P/r_j, p_j = p/L_{\max}$. Indeed, if delivery times are replaced by due dates then the maximal job lateness $L_{\max}$ is to be minimized. Problem (2) can be solved in

time $O(q_{\max}mn \log n + O(m\kappa n))$, where $q_{\max}$ is the maximal job delivery time and $\kappa < n$ is a parameter which is known only after the termination of the algorithm. In practice, κ turns out to be a small positive integer (see Vakhania [1]).

We show that problems (1) and (2) have an inherent common properties that can lead to common efficient solution methods for these problems. We use the so-called *solution tree* $\mathcal{T}$ to enumerate feasible schedules. A node in $\mathcal{T}$ is a feasible solution to our problem. We may reduce the number of the enumerated solutions by restricting the type of the feasible solutions we deal with. In particular, we introduce the so-called *behavior alternatives*. The analysis of these behavior alternatives leads us to a reduced solution space. We show that certain idle time intervals (gaps) are unavoidable in an optimal schedule, which are iteratively "guessed" by our method.

2. Common Properties

Our definitions cover both problems (1) and (2). In case of a variation, we give two alternative cases for problem (1) and problem (2), respectively, in one definition. We will use a fast heuristic algorithm for the creation of a feasible solution. Iteratively, among all released jobs by the current scheduling time t, our heuristic $\mathcal{H}$ schedules a job with the smallest due-date or equivalently, with the largest delivery time on the next adjacent machine.

In problem (1), if a job is late, it can be scheduled arbitrarily late. Hence our task reduces to finding such a schedule in which all jobs are scheduled within their *feasible intervals*, that is, between their respective release times and due dates. From now on, we will be dealing only with such feasible schedules.

We denote by $|S|$ the value of the objective function for schedule S. For problem (2), we call the earliest scheduled in S job o that realizes the value $|S|$ (i.e., $\mathcal{C}_o = |S|$) the *overflow job* in S. Iteratively, we will increase artificially the release times of some non-urgent jobs so that they will be scheduled after more urgent jobs once $\mathcal{H}$ is again applied. As a result, the latter jobs will be restarted earlier in the resultant schedule. In general, we expect that this will decrease the value of our objective function. We call these non-urgent jobs *emerging jobs*. From now on, we let o be an omitted job for problem (1), and the overflow job for problem (2). Then e is an emerging job for o if it is scheduled before time d_o and is less urgent than o, i.e., $d_e > d_o$.

Due to $\mathcal{H}$-heuristic, all jobs in S completely scheduled (started and finished)

within the feasible region of o for problem (1) (between jobs e and o for problem (2)) are no less urgent than o (i.e., they have no greater due-date than o). We shall refer to this set of jobs as the *kernel* jobs. Note that all kernel jobs are scheduled before the earliest scheduled emerging job for o (we call that job the *live emerging job*). Observe that if we (re)move an emerging job then the kernel jobs will be left-shifted (rescheduled at earlier times) releasing in this way space for (re)scheduling job o earlier.

As we have already mentioned, we release the space occupied by an emerging job e by increasing artificially its release time to r_o, so that when $\mathcal{H}$-heuristic is again applied, it will not select job e ahead job o (e may only be scheduled after o). To keep the released by job e space for job o, we would not like another non-urgent job j with $d_j \geq d_o$ to be scheduled (partially) within or before the feasible region of o in S to be scheduled within the space released by job e in the newly generated schedule. Hence, we modify the release time of any such a job similarly as that of e.

We shall refer to the resultant schedule as a *complementary* or *$\mathcal{H}$-schedule* and denote it by $S_e[o]$. We will say that e is *activated* for o in $S_e[o]$. For the notational simplicity, we may omit an argument in $S_e[o]$.

Now we give our first result for problems (1) and (2) showing that the existence of an emerging job is principal for both problems. If for problem (1) there is no omitted job in σ then it is optimal. Otherwise, there may exist or may not an emerging job for such a job o. Note that in the latter case, o cannot be included without omitting some job from S. But such a rearrangement, imposing an additional gap, yields only a forced left-shift for the preceding jobs. It is straightforward to see that the rearrangement cannot be beneficial:

Lemma 1. (Problem (1)) *An omitted job in σ can be passively discarded if there is no emerging job for it.*

We have a similar statement for problem (2) which ensures that an optimal schedule can be found by enumeration of only $\mathcal{H}$-schedules:

Lemma 2. (Problem (2)) *$|S_e|$ cannot be less than $|S|$ if e is not an emerging job.*

Proof. It is obvious that to reduce $|S|$, the number of jobs scheduled before o in S should decrease by at least by one. Hence, a job j, scheduled before the kernel jobs in S should be rescheduled after these jobs. Now $t_j^{S_j} \geq t_o^S$ and $q_j \geq q_o$ easily yields $|S_j| \geq |S|$. □

Let S be an $\mathcal{H}$-schedule for problem (1) with the live emerging job l, j be

a job with the minimal release time in the kernel, and let $\delta_S = c_l^S - r_j$. The following lemma insures that the distance from S to an optimal schedule is not "too much":

Lemma 3. (Problem (2)) *The lower bound on the optimal schedule makespan is* $|S| - \delta_S$.

Proof. $\mathcal{C}_o^S$ is the makespan of S. This magnitude cannot be reduced unless the starting time of the earliest scheduled kernel job in S is decreased. But by the definition of δ_S and $\langle$-heuristic, no kernel job j can start its processing before the time $t_j^S - \delta_S$. Therefore $|S|$ can decrease at most by δ_S and so $|S| - \delta_S$ is resulting bound. □

It is easily seen that there arises a gap (an idle time interval) before job o in $S_e[o]$. This follows from the $r_o > r_e$ which, in turn holds as otherwise the $\mathcal{H}$-heuristic would select job o ahead job e (and this was not the case).

Now we introduce our first behavior alternative for both problems. Suppose, for problem (1), job o is not included in $S_e[o]$, and for problem (2) suppose that o still remains the overflow job in $S_e[o]$. Then we will say that there occurs an *instance of alternative (a)* with job o (IA(a) for short) in $S_e[o]$.

Suppose for problem (1) IA(a) in $S_e[o]$ occurs. We may force to include o but only at the expense of the omission of some j scheduled in the feasible region of o in $S[o]$ (by the definition of $S_e[o]$). Similarly as in Lemma 1, such a possible rearrangement cannot be beneficial and the following lemma holds.

Lemma 4. (Problem (1)) *Job o might be discarded if IA(a) with o occurs.*

Essentially stronger statement holds for problem (2).

Lemma 5. (Problem (2)) $S_e[o]$ *is optimal if there occurs an IA(a) with o in it.*

Proof. The overflow job o is started at its release time as it is preceded by a gap in $S_e[o]$. Hence, there exists no feasible schedule in which its full completion time can be reduced, and therefore $S_e[o]$ is optimal. □

Because of the newly arisen gap(s) in $S_e[o]$, there will occur a forced right-shift for the jobs scheduled after job o in $S_e[o]$. We shall refer to the magnitude of this right-shift as the *disturbance* for the latter jobs.

For problem (1), similarly to job o, not necessarily job e is present in $S_e[o]$. We will say in this case that there occurs an *instance of alternative (b)* with job e in $S_e[o]$. For problem (2) the definition of alternative (b) is essentially the same: We will say that there occurs IA(b) in S_e if e becomes the overflow job

in S_e.

For problem (1), it is not difficult to see that $|S_e[o]| \leq |S|$ if IA(b) in $S_e[o]$ occurs. Indeed, on the one hand, the number of jobs scheduled after time r_e in both S and $S_e[o]$ should be the same as otherwise job e would have been included in $S_e[o]$. At the same time, because of the disturbance caused by the principal gaps these jobs in $S_e[o]$ are rescheduled at an earlier time moments as compared to S. Then clearly no extra job could have been introduced in $S_e[o]$ before time r_e. Hence, the number of jobs scheduled before time r_e in $S_e[o]$ cannot be more than that in S and we obtain the claimed inequality $|S_e[o]| \leq |S|$. Thus we are doomed to loose a job (job e) whenever IA(b) with job e occurs. Since we gain in at most one job (job o), we may gain nothing in total. Instead of job e now we have job o included, whereas the earlier scheduled jobs are forced to be left-shifted because of the newly arisen principal gaps in $S_e[o]$. This obviously cannot give us improved conditions for including more jobs, as compared to S. The reader may readily verify that a similar picture holds for problem (2).

Lemma 6. (Problems (1) and (2)) *The pair* (e, o) *can be closed if IA(b) with* e *occurs in* $S_e[o]$.

Presented lemmas can be used in an obvious manner to reduce the set of feasible schedules necessary for finding an optimal schedule, which, in turn, leads us to an improved efficiency.

Acknowledgements

This work is partially supported by CONACyT grant 48433.

References

[1] N. Vakhania, A better algorithm for sequencing with release and delivery times on identical processors, *Journal of Algorithms*, **48** (2003), 273-293.