Two-Machine No-Wait Flow Shop Scheduling Problem with Precedence Constraints

Saied Samie and Behrooz Karimi

Department of Industrial Engineering, Amirkabir University of Technology, Tehran, Iran
saeedsamiee@cic.aut.ac.ir
b.karimi@aut.ac.ir

Summary. In this paper, we consider two-machine permutation flow shop scheduling problem to minimize the makespan in which some of the jobs have to be processed with no-wait in process. We also consider precedence constraints which impose some jobs to be processed before or after some others. We propose a constructive algorithm to find a feasible solution. We also develop a Tabu Search algorithm to solve this problem. Our computational analysis indicates its good performance.

1 Introduction

In the classical two-machine flow shop scheduling problem, we are given a set of n jobs that have to be processed on two machines, A and B. The processing times of all jobs on two machines are given. Each machine processes one job at a time. Each job is processed first on machine A and then on machine B. No preemption is allowed. For a feasible schedule S the maximum completion time of all jobs on two machines is called the makespan and is denoted by $C_{max}(S)$. The resulting problem of minimizing the makespan is traditionally denoted by $F2||C_{max}$ and is known solvable in O(n *log* n) time.

In an important variation of $F2||C_{max}$, known as the flow shop with no-wait in process, it is assumed that every job starts on machine B exactly at the time of its completion on machine A. This problem, denoted by $F2|$*no-wait*$|C_{max}$, reduces to a special case of the traveling salesman problem (TSP) which can be solved in O(n *log* n) time [1, 2].

We consider a version of the two machine flow shop in which there are two types of jobs: regular and no-wait jobs [4]. Each regular job is processed as in problem $F2||C_{max}$, i.e., it starts on machine B no earlier than it is completed on machine A. Each no-wait job is processed as in problem $F2|$*no-wait*$|C_{max}$, i.e., it starts on machine B immediately after it is completed on machine A.

Extending standard scheduling notation, we denote the problem under consideration by $F2|$*reg+no-wait*$|C_{max}$. Let $N_R = \{J_1, \ldots, J_{n_R}\}$ be the set of regular jobs, $|N_R| = n_R$ and $N_{NW} = \{I_1, \ldots, I_{n_{NW}}\}$ be the set of no-wait jobs, $|N_{NW}| = n_{NW}$. We have $n = n_R + n_{NW}$ jobs to schedule. If either the number of regular jobs or the

number of the no-wait jobs is bounded by a constant q, the problem is denoted either by $F2|reg{+}no\text{-}wait, n_R \leq q|C_{max}$ or $F2|reg{+}no\text{-}wait, n_{NW} \leq q|C_{max}$, respectively.

For each of the basic problems $F2||C_{max}$ and $F2|no\text{-}wait|C_{max}$ there exists an optimal solution that is a permutation schedule in which each machine processes the jobs according to the same sequence. In general, this is not true for this extended model. In this model, the global optimal schedule may be achieved by allowing machines process the jobs in different sequences. But we restrict the search for an optimal solution to the class of permutation schedules, in which case we write $F2|reg{+}no\text{-}wait,\ prmu|C_{max}$ to denote the corresponding problem.

Problem $F2|reg{+}no\text{-}wait,\ prmu|C_{max}$ is NP-hard in the strong sense, provided that both n_R and n_{NW} are variable, i.e., part of the problem input. Also, it has been proved that the problem $F2|reg + no\text{-}wait, n_R \leq q, prmu|C_{max}$ is polynomially solvable and the problem $F2|reg + no\text{-}wait, n_{NW} \leq q, prmu|C_{max}$ is NP-hard in the ordinary sense even for q=1 [4].

Feasible schedules are further restricted by job precedence constraints given by partial order $\prec$, where $J_i \prec J_k$ means that job J_i must be processed before job J_k. Because of this new constraint, we shall denote the problem as $F2|reg + no\text{-}wait + prec, prmu|C_{max}$.

2 Generating a Feasible Schedule

We are given a set of precedence constraints in the form of $J_i \prec J_k$ which imposes job J_i to be processed before J_k in the permutation. To generate a feasible schedule, we first sort all jobs in an arbitrary order of their indices. This permutation may not be feasible because some constraint may not be satisfied. To change this unfeasible schedule to a feasible one, we use the following algorithm. This algorithm considers all constraints, one after the other and changes the position of jobs to satisfy the constraint. For example, if the constraint $J_i \prec J_k$ is being considered and the job J_i is placed in a position after job J_k, the algorithm moves the job J_i to be placed just before job J_k. But during this action, some other constraints that are considered earlier may not be longer satisfied. So the algorithm finds all jobs (J_l) between J_k and J_i that have precedence constraint with J_i in the form $J_l \prec J_i$ and moves them with job J_i to keep relating constraints satisfied. The algorithm is as below:

Step 1. Sort all jobs in an arbitrary order of their indices.

Step 2. Consider the first constraint. If this constraint is satisfied, go to step 6 and if not, call first job of constraint as *Pre* and the second job as *Suc* and go to next step.

Step 3. Find all jobs that are placed after *Suc* and meanwhile should be processed before *Pre* and put them in set K. If any job is added to set K, go to next step and if not, just add the *Pre* to set K and then go to step 5.

Step 4. Consider individually all the jobs which are added recently to the set K and find the jobs that are placed after *Suc* and meanwhile should be processed before the considered job. If there exists any new job, add it to the set K. Repeat this step for the last new added jobs in set K. If the set K remains unchanged, add the *Pre* to set K and go to the next step.