Minimizing the number of tardy jobs on a single machine with batch setup times [*]

Günter Rote [†] Gerhard J. Woeginger[‡]

Abstract

This paper investigates a single-machine sequencing problem where the jobs are divided into families, and where a setup time is incurred whenever there is a switch from a job in one family to a job in another family. This setup only depends on the family of the job next to come and hence is sequence independent. The jobs are due-dated, and the objective is to find a sequence of jobs that minimizes the number of tardy jobs.

The special case of this problem where in every family the jobs have at most two different due dates is known to be $\mathcal{NP}$-complete [Bruno & Downey, 1978]. The main result of this paper is a polynomial time algorithm for the remaining open case where in every family all the jobs have the same due date. This case may be formulated as a dual resource allocation problem with a tree-structured constraint system, which can be solved to optimality in polynomial time.

Keywords. sequencing; scheduling; batch setup times; number of tardy jobs.

1 Introduction

This paper deals with the following scheduling problem. There are n jobs $J_1, \ldots, J_n$ that are to be processed without interruption on a single machine. All jobs are available for processing at time zero. The set of jobs is divided into F families; a setup time s_f is associated to each family $f = 1, \ldots, F$. Whenever a job in family f is processed, this incurs the setup time s_f unless another job from the same family is processed immediately before this job. The machine can execute at most one job at a time, and it cannot perform any processing while undergoing a setup. Job J_j ($j = 1, \ldots, n$) has a positive integer processing time p_j, and an integer due date d_j. In a schedule σ, we denote by $C_j(\sigma)$ the completion time of job J_j ($j = 1, \ldots, n$).

[*]This research has been supported by the START program Y43-MAT of the Austrian Ministry of Science, and by the Spezialforschungsbereich *Optimierung und Kontrolle*, Projektbereich *Disckrete Optimierung*.

[†]Institut für Mathematik B, TU Graz, Steyrergasse 30, A-8010 Graz, Austria, e-mail: rote@opt.math.tu-graz.ac.at.

[‡]Institut für Mathematik B, TU Graz, Steyrergasse 30, A-8010 Graz, Austria, e-mail: gwoegi@opt.math.tu-graz.ac.at.

If $C_j(\sigma) > d_j$, then job J_j is *tardy* and we set $U_j = 1$. If $C_j(\sigma) \le d_j$, then job J_j is processed *on-time* and we set $U_j = 0$. The objective is to find a processing order of the jobs that minimizes $\sum_{j=1}^{n} U_j$, i.e. the number of tardy jobs. In the standard scheduling notation (cf. Lawler, Lenstra, Rinnooy Kan & Shmoys [5] and Potts & van Wassenhove [8]), this problem is denoted by $1\,|\,s_f\,|\,\sum U_j$. For related problems and for practical applications involving batch setup times, the interested reader is referred to Monma & Potts [6], Potts & van Wassenhove [8], and Webster & Baker [9].

A special case of $1\,|\,s_f\,|\,\sum U_j$ is the feasibility testing problem, i.e. the problem of deciding whether there is a feasible schedule in which *all* jobs of a given instance are on-time. Bruno & Downey [1] prove that the feasibility testing problem is $\mathcal{NP}$-hard, even if there are only two distinct deadlines per family. An instance of $1\,|\,s_f\,|\,\sum U_j$ where in every family all jobs have the same due date, is said to have *uniform family due dates*. In this paper we will show that the problem with uniform family due dates is solvable in polynomial time. This special case is sufficiently general to contain the problem $1\,|\,|\,\sum U_j$ *without* batch setup times (in $1\,|\,|\,\sum U_j$, every jobs forms its own family and all family setup times are zero). Hence, our result generalizes the well-known polynomial time algorithm of Moore [7].

Our solution approach to $1\,|\,s_f\,|\,\sum U_j$ is as follows: We formulate $1\,|\,s_f\,|\,\sum U_j$ with uniform family due dates as a dual resource allocation problem with tree-structured constraints (cf. Section 3). Since this dual resource allocation problem can be solved in polynomial time by dynamic programming (cf. Section 2), the scheduling problem itself can be solved in polynomial time.

2 A dual resource allocation problem

The resource allocation problem (cf. Ibaraki & Katoh [3]) is a well-known optimization problem with a (possibly) complex objective function under a single, extremely simple constraint. In the dual resource allocation problem (cf. Katoh, Ibaraki & Mine [4] or Section 10.1 in [3]), the roles are exchanged and the objective function is simple whereas the constraint system may be messy. In this section, we investigate the following dual resource allocation problem (DAP).

$$\text{(DAP)} \qquad \begin{array}{ll} \max & \sum_{f=1}^{F} x_f \\ \text{s.t.} & \left\{ \begin{array}{lll} \sum_{f\in S} g_f(x_f) \le c_S & & \text{for all } S \in \mathcal{S} \\ 0 \le x_f \le n_f & & f = 1,\ldots,F \\ x_f \text{ integer} & & f = 1,\ldots,F \end{array} \right. \end{array}$$

For $1 \le f \le F$, the function $g_f\colon [0,n_f] \to \mathbb{R}$ is an arbitrary function which is specified as an ordered list of pairs $(x, g_f(x))$, $x = 0,\ldots,n_f$. The values n_f, $1 \le f \le F$, are positive integers. The set system $\mathcal{S}$ is a system of non-empty sets over $\{1,\ldots,F\}$. For every $S \in \mathcal{S}$, the value c_S is an arbitrary real number.

Moreover, denote $n = \sum_{f=1}^{F} n_f$. Observe that $n \geq F$ holds and that by the specification of the functions g_f, the numbers n_f and n are essentially encoded in unary.

Proposition 2.1 *The dual allocation problem (DAP) is an $\mathcal{NP}$-hard problem.*

Proof. The statement may be proved e.g. via a reduction from INDEPENDENT SET IN GRAPHS (cf. Garey & Johnson [2]): Given a graph $G = (V, E)$, find the maximum number of pairwise non-adjacent vertices. For every vertex $v_f \in V$, introduce a corresponding variable x_f in (DAP) with the interpretation "$x_f = 1$" if v_f belongs to the independent set and "$x_f = 0$" otherwise. Moreover, set $n_f = 1$, $g_f(0) = 0$ and $g_f(1) = 1$. For every edge $e = (v_f, v_h)$ introduce the set $\{f, h\}$ in $\mathcal{S}$ and set $c_{\{f,h\}} = 1$. Then the optimal objective value of (DAP) yields the size of the maximum independent set in G. $\square$

A set system $\mathcal{S}$ is called *tree-structured* if $\emptyset \notin \mathcal{S}$ and for all $S', S'' \in \mathcal{S}$,

$$S' \subseteq S'' \qquad \text{or} \qquad S'' \subseteq S' \qquad \text{or} \qquad S' \cap S'' = \emptyset.$$

With a tree-structured set system $\mathcal{S}$, we associate a directed in-forest $\mathcal{F}(\mathcal{S})$ as follows: For every set $S \in \mathcal{S}$ the forest contains a corresponding vertex; in the following we will not distinguish between a set S and its corresponding vertex. There is a directed edge from a set S' to another set S'' in $\mathcal{F}(\mathcal{S})$ if and only if $S' \subset S''$ and there is no S''' in $\mathcal{S}$ with $S' \neq S''' \neq S'$ and $S' \subset S''' \subset S''$. Clearly, every vertex in $\mathcal{F}(\mathcal{S})$ has out-degree at most one. Adding all singleton sets to $\mathcal{S}$ does not destroy the tree-structured property. But then, the forest $\mathcal{F}(\mathcal{S})$ has F leaves, and the remaining vertices have indegree at least 2. It follows that a tree-structured family $\mathcal{S}$ contains at most $2F - 1$ sets.

Lemma 2.2 *For any instance $I = (n_f, g_f, \mathcal{S}, c_S)$ of (DAP) with tree-structured $\mathcal{S}$, one can construct in $O(n + F^2)$ time another instance $I' = (n_f, g_f, \mathcal{S}', c'_S)$ of (DAP) such that the following conditions are fulfilled.*

(C1) I and I' are equivalent, i.e. they have the same set of optimal solutions and the same optimal objective value.

(C2) The set system $\mathcal{S}'$ in I' is tree-structured; $|\mathcal{S}'| = 2F - 1$ holds; the in-forest $\mathcal{F}(\mathcal{S}')$ associated with $\mathcal{S}'$ is a binary in-tree.

Proof. We construct I' in two steps by adding more sets to $\mathcal{S}$. We set the right-hand sides c'_S of all the corresponding new inequalities to the global upper bound $c^* = \sum_{f=1}^{F} \max\{0, \max_{0 \leq x \leq n_f} g_f(x)\}$, which makes them redundant. Initialize $\mathcal{S}' := \mathcal{S}$ and for all $S \in \mathcal{S}$ set $c'_S = c_S$. If $\{f\} \notin \mathcal{S}'$ for some $f \in \{1, \ldots, F\}$ then add the new set $\{f\}$ to $\mathcal{S}'$. If $\{1, \ldots, F\} \notin \mathcal{S}'$ then add the new set $\{1, \ldots, F\}$ to $\mathcal{S}'$.

In the second step, repeat the following procedure as long as some vertex S in $\mathcal{F}(\mathcal{S}')$ has three or more in-going edges: Let S_{i_1} and S_{i_2} be two arbitrary children of

S in $\mathcal{F}(\mathcal{S}')$; add the set $S' = S_{i_1} \cup S_{i_2}$ to $\mathcal{S}'$. By iterating this procedure, eventually every interior vertex in $\mathcal{F}(\mathcal{S}')$ will have in-degree two and condition (C2) will be fulfilled. This completes the construction of instance I'. It can be verified that I' is equivalent to I and hence, conditions (C1) and (C2) are both fulfilled.

It remains to discuss the time complexity. The first step is easily done in $O(n + F)$ time. In the beginning of the second step, compute the current forest $\mathcal{F}(\mathcal{S}')$ as follows. First construct a simple, undirected, loopless auxiliary graph with vertex set $\mathcal{S}'$: For every $f = 1, \dots, F$ and for every $S', S'' \in \mathcal{S}'$ with $f \in S'$ and $f \in S''$, put an edge between S' and S'' into the auxiliary graph. The auxiliary graph can be constructed in $O(F^2)$ overall time. Then for $S \in \mathcal{S}'$, $S \neq \{1, \dots, F\}$, the unique out-going edge in $\mathcal{F}(\mathcal{S}')$ goes to the set S' where (i) S' is adjacent to S in the auxiliary graph, (ii) $|S'| > |S|$, and (iii) $|S'|$ is smallest possible under these conditions. In this way, the forest $\mathcal{F}(\mathcal{S}')$ for $\mathcal{S}'$ at the beginning of the second step can be computed in $O(F^2)$ time from the auxiliary graph. Getting rid of the vertices with in-degree greater than two can be done by locally manipulating $\mathcal{F}(\mathcal{S}')$; it is routine to implement it in $O(F^2)$ overall time. □

Theorem 2.3 *The special case of the dual allocation problem (DAP) where $\mathcal{S}$ is tree-structured is solvable in $O(n^2)$ time.*

Proof. First we apply Lemma 2.2 to get in $O(n + F^2)$ time an equivalent instance where $\mathcal{F}(\mathcal{S})$ is a binary tree. Let $S_1, \dots, S_{2F-1}$ be an enumeration of the sets in $\mathcal{S}$, such that $S_i \subset S_j$ implies $i \leq j$. For $S \in \mathcal{S}$, let $n(S) = \sum_{f \in S} n_f$.

The remaining argument will be done by dynamic programming. Define a two-dimensional array $A[i, \ell]$ where $1 \leq i \leq 2F - 1$ and $0 \leq \ell \leq n$ with the following meaning: The value $A[i, \ell]$ is the smallest g^* for which there exist values x_f^*, $f \in S_i$, such that

(A1) $\sum_{f \in S} g_f(x_f^*) \leq c_S$ holds for all $S \in \mathcal{S}$, $S \subseteq S_i$.

(A2) $\sum_{f \in S_i} x_f^* = \ell$.

(A3) $\sum_{f \in S_i} g_f(x_f^*) = g^*$.

If no values x_f^* fulfilling (A1) and (A2) exist, then $A[i, \ell] = +\infty$. This happens for example when $\ell > n(S_i)$. Hence from now on, we will only deal with entries $A[i, \ell]$ for which $\ell \leq n(S_i)$. We compute the entries $A[i, \ell]$ in increasing order of i. If $|S_i| = 1$, let $S_i = \{f\}$ and set for $0 \leq \ell \leq n_f$

$$A[i, \ell] = \begin{cases} g_f(\ell) & \text{if } g_f(\ell) \leq c_{S_i} \\ +\infty & \text{otherwise.} \end{cases} \tag{1}$$

If $|S_i| > 1$, let $S_i = S_a \cup S_b$ with $a < b < i$, where S_a and S_b are the two children of S_i in $\mathcal{F}(\mathcal{S})$. Then for $\ell \leq n(S_i)$,

$$A[i, \ell] = \begin{cases} \min \; \{ A[a, k] + A[b, \ell - k] : \; 0 \leq k \leq n(S_a), \; 0 \leq \ell - k \leq n(S_b) \} \\ +\infty \qquad \text{if this minimum is greater than } c_{S_i}. \end{cases} \tag{2}$$

It can be verified that with the above definitions, (A1)–(A3) are always fulfilled for $g^* = A[i, \ell]$. In the end, the optimal objective value of (DAP) equals the maximum ℓ for which $A[2F-1, \ell]$ takes a finite value.

Let us analyze the time needed to compute all values $A[i, \ell]$. Denote by $T(i)$ the total time needed to handle all finite entries $A[j, \ell]$ with $0 \le \ell \le n$ and $S_j \subseteq S_i$. Then for $|S_i| = 1$ with $S_i = \{f\}$, (1) implies that

$$T(i) \;=\; \text{const}_1 \cdot n_f \;=\; \text{const}_1 \cdot n(S_i). \tag{3}$$

If $|S_i| > 1$, let S_a and S_b be the two children of S_i in $\mathcal{F}(S)$. Note that $a < b < i$, that $S_i = S_a \cup S_b$, and that $n(S_i) = n(S_a) + n(S_b)$ holds. We claim that

$$T(i) \;=\; T(a) + T(b) + \text{const}_2 \cdot n(S_a) \cdot n(S_b). \tag{4}$$

This can be seen as follows. The time $T(i)$ consists of the total time for handling all entries $A[j, \ell]$ with $0 \le \ell \le n$ and $S_j \subseteq S_a$ or $S_j \subseteq S_b$, plus the total time for handling all entries $A[i, \ell]$ with $0 \le \ell \le n$. For every α, $0 \le \alpha \le n(S_a)$, and for every β, $0 \le \beta \le n(S_b)$, in (2) there is exactly one step performed with $k = \alpha$ and $\ell - k = \beta$. Hence, the total time for handling the entries $A[i, \ell]$ with $0 \le \ell \le n(S_i)$ is proportional to $n(S_a) \cdot n(S_b)$. Hence, (4) indeed holds. By induction, one proves from (3) and (4) that

$$T(i) \;\le\; \text{const} \cdot n(S_i)^2.$$

Consequently, the total time $T(2F-1)$ needed for computing all entries is $O(n^2)$. Since $F \le n$, the time spent on applying Lemma 2.2 is also $O(n^2)$. Summarizing, this yields the running time claimed in the statement of the theorem.

Finally, we remark that by storing appropriate auxiliary information in the dynamic program and by doing some backtracking, one can also explicitly compute the values x_f in an optimal solution; this increases the running time by only a constant factor. Since these are standard techniques, we do not elaborate on them. □

3 Solution of the scheduling problem

In this section we discuss the scheduling problem $1 \mid s_f \mid \sum U_j$ that has been defined in the introduction. The following observation follows via straightforward job interchange arguments.

Observation 3.1 *For any instance of* $1 \mid s_f \mid \sum U_j$ *with uniform family due dates, there is an optimal schedule of the following form.*

- *(i) For every family, the on-time jobs of that family are processed consecutively; hence, the setup for each family is performed at most once.*
- *(ii) In each family, the on-time jobs are the shortest jobs of the family.* □

For $f = 1, \dots, F$, denote by d_f the due date of the jobs in family f. Without loss of generality assume that $d_1 \leq d_2 \leq \cdots \leq d_F$. Let n_f, $f = 1, \dots, F$, denote the number of jobs in family f, and let $p_{f,1} \leq p_{f,2} \leq \cdots \leq p_{f,n_f}$ denote their processing times. Moreover, define

$$g_f(x) = \begin{cases} 0 & \text{if } x = 0 \\ s_f + \sum_{i=1}^{x} p_{f,i} & \text{if } 1 \leq x \leq n_f. \end{cases}$$

For $f = 1, \dots, F$, introduce $S_f = \{1, \dots, f\}$ and set $c_{S_f} = d_f$. Define $\mathcal{S} = \{S_1, \dots, S_F\}$. Finally, denote by x_f the number of on-time jobs from family f, $f = 1, \dots, F$.

With this choice of parameters, the dual allocation problem (DAP) is equivalent to $1 \mid s_f \mid \sum U_j$ with uniform family due dates. Moreover, $\mathcal{S}$ is tree-structured and hence Theorem 2.3 implies the main result of this paper:

Theorem 3.2 *The special case of $1 \mid s_f \mid \sum U_j$ with uniform family due dates is solvable in $O(n^2)$ time.* □

Acknowledgement. The authors thank Bettina Klinz for several valuable discussions and for providing pointers to the literature on allocation problems.

References

[1] J. Bruno and P. Downey, Complexity of task sequencing with deadlines, set-up times, and changeover costs, *SIAM Journal on Computing* **7**, 1978, 393–404.

[2] M.R. Garey and D.S. Johnson, *Computers and Intractability, A Guide to the Theory of NP-Completeness*, Freeman, San Francisco, 1979.

[3] T. Ibaraki and N. Katoh, *Resource Allocation Problems: Algorithmic Approaches*, MIT Press, Boston, 1988.

[4] N. Katoh, T. Ibaraki, and H. Mine, Algorithms for a variant of the resource allocation problem, *Journal of the Operations Research Society of Japan* **22**, 1979, 287–299.

[5] E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, Sequencing and scheduling: Algorithms and complexity, in: S.C. Graves, A.H.G. Rinnooy Kan, and P.H. Zipkin (eds.) *Logistics of Production and Inventory*, Handbooks in Operations Research and Management Science 4, North-Holland, Amsterdam, 1993, 445–522.

[6] C.L. Monma and C.N. Potts, On the complexity of scheduling with batch set-up times, *Operations Research* **37**, 1989, 798–804.

[7] J. Moore, An n job, one machine sequencing algorithm for minimizing the number of late jobs, *Management Science* **15**, 1968, 102–109.

[8] C.N. Potts and L.N. van Wassenhove, Integrating scheduling with batching and lotsizing: a review of algorithms and complexity, *Journal of the Operational Research Society* **43**, 1992, 395–406.

[9] S. Webster and K.R. Baker, Scheduling groups of jobs on a single machine, *Operations Research* **43**, 1995, 692–703.

Received January, 1998