Computing with REAL-TIME SYSTEMS

Volume 2

Proceedings of the
Second European Seminar
University of Erlangen – Nürnberg

Edited by
I.C. PYLE
AERE Harwell
and P. ELZER
University of Erlangen

TRANSCRIPTA BOOKS

30 Craven Street, Strand, London WC2

Process scheduling by output considerations

G. McC. HAWORTH
University of Cambridge, England

Introduction

In multi-tasking systems when it is not possible to guarantee completion of all activities by specified times, the scheduling problem is not straightforward. Examples of this situation in real-time programming include the occurrence of alarm conditions and the buffering of output to peripherals in on-line facilities. The latter case is studied here with the hope of indicating one solution to the general problem.

Three parameters are associated with each job J_i $(i = 1,\ldots,n)$. These are the processing time estimate p_i, the due-date d_i, and a positive weighting w_i. Three problem areas are distinguished:

1. The unweighted deterministic case (UDP) where p_i is the known processing time and $w_i = 1$ for each J_i.
2. The weighted deterministic case (WDP) where we have general weights w_i and known processing times p_i.
3. The stochastic case (SP), where the processing time of J_i is a random variable with an exponential distribution and mean p_i. General weights cause no difficulty in this case, but we are particularly interested in sub-optimal schedules as the optimal schedule cannot be derived entirely in algebraic terms.

Given a schedule S, J_i will be completed at time $c_i(S)$: the cost which we seek to minimise is

$$T = \text{Expectation}\left\{\sum_{i=1}^{n} w_i \times \max[0,\ c_i(S) - d_i]\right\}$$

and jobs are indexed so that

$$i < j \Rightarrow p_i w_i^{-1} < p_j w_j^{-1} \quad or$$

$$(p_i w_i^{-1} = p_j w_j^{-1},\ \ d_i \leqslant d_j)$$

The full derivation of the algorithms, which are original, can be found in [4].

The unweighted deterministic problem

The cost in the deterministic cases is minimised by sequencing the J_i without pre-emption or idle-time [1, p. 24]. When $w_i = 1$, the iterative algorithm on p. 76 computes the best sequence. This method is more convenient than the branch-and-bound approach of [2] but springs from the same source, namely

Th. UD1

$d_1 \leqslant \max(d_2, p_2) \Longleftrightarrow J_1$ precedes J_2 when $n = 2$ (Fig. 1)

Th. UD2

$i < j$, $d_i \leqslant \max(d_j, p_j) \Rightarrow J_i$ precedes J_j (for any 'n')

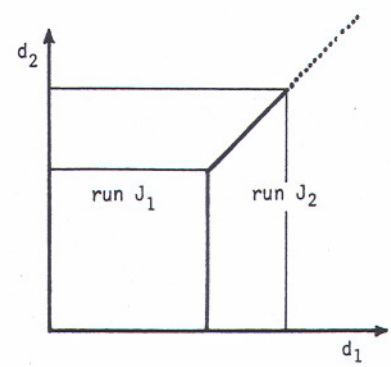
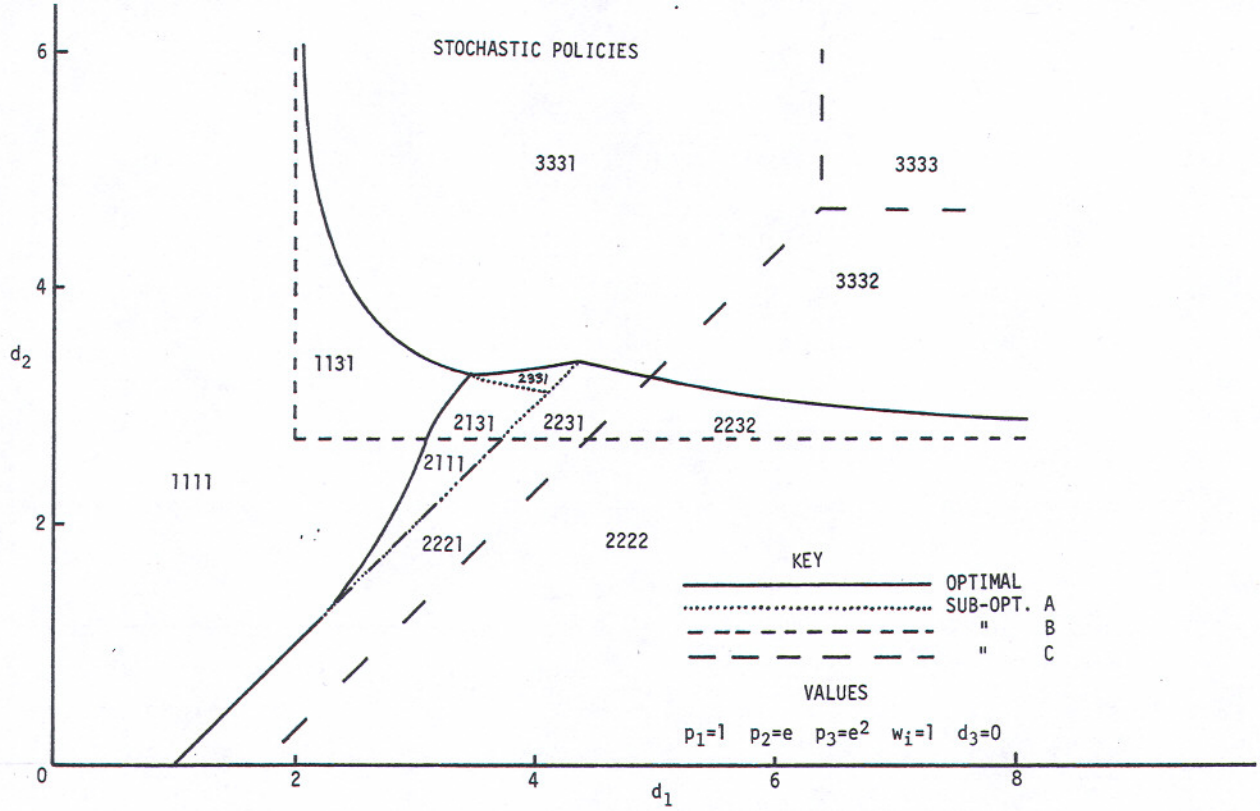


Fig. 1

An 'eligible' job is one which is not excluded from the next position by Theorem UD2. The algorithm forms the LEJ (longest eligible job) sequence by executing the longest eligible job at each stage. The LEJ sequence is certainly as good as the EDD (earliest due-date) and LST (least slack-time) sequences, and connot be improved by interchanging jobs adjacent in the sequence.

Algorithm for the UDP

```
global  proc, due, best sequence [1:n] ;
comment  'proc' and 'due' hold the processing time and due dates
         respectively of the jobs 1,....,n;

 {comment  this block computes the best sequence [1:n]  from
           proc, due [1:n]  for the unweighted deterministic problem;
  local  tard, done, seq [1:n] , best cost, deferred job,
         start of tail, in doubt;
  form LEJ sequence (1,n);  best sequence := seq;  best cost := cost;
  checktail;
  while  in doubt  do  {try to improve best sequence;  checktail}}

form LEJ sequence (start, limit)  ≙
   {comment  this fills places  start,...,n  with job indices ≤ limit;
    local  i, 1, pr, place;   pr := 0;
    if start ≠ 1 then for i := 1 to start-1 do pr := pr + proc[seq[i]];
    for place := start to n do
       {comment  this puts the longest eligible job in place;
        i := 1; while done[i] > 0 do i := i+1;  1 := i+1;
        while 1 ≤ limit and proc[1] < due[i] - pr do
           {if done[1] ≤ 0  and  due[1] < due[i] then i := 1; 1 := 1+1};
        seq[place]:= i; pr := pr + proc[i];
        tard[i] := max (0, pr-due i );
        if done[i] = -1  then  limit := n;  done[i] := 1}}


cost  ≙  Σ (i=1 to n)  tard[i]

check tail  ≙
   {comment  this looks for the shortest tail sequence that might
             not be in the optimal order;
    local  place, prefer;
    in doubt := false;  place := n;  prefer := n+1;
    while  not in doubt  and  place > 0  do
       {done[seq[place]] := 0;
        if  seq[place] > prefer  then
           {in doubt := true;  deferred job := seq[place];
            start of tail := place}
        else  if  tard[seq[place]] > proc[seq[place]]  then
                 {done[seq[place]] := -1;
                  if  seq[place] < prefer then prefer := seq[place]}
        place := place - 1}}


try to improve best sequence  ≙
   {comment  this defers the first job in the doubtful tail sequence;
    form LEJ sequence (start of tail, deferred job -1);
    if cost < best cost  then {best cost := cost; best sequence := seq}}
```



If $W = \{i \mid c_i(\text{LEJ}) - p_i > d_i, \exists j > i$ s.t. J_j precedes J_i in LEJ $\}$

then the minimal cost is not less than

$$T(\text{LEJ}) - \sum_{i \in W} [c_i(\text{LEJ}) - p_i - d_i].$$

Hence, $W = \phi$ is a sufficient but not necessary condition for the LEJ sequence to be optimal, and it is usually satisfied.

If $c_i(\text{LEJ}) - p_i \leq d_i$ for all i, then LEJ coincides with EDD and is optimal. Thus the EDD rule, which is known to be best if each J_i can be completed by d_i, is seen to have far wider optimal properties.

The algorithm has been run on collections of 100 job-sets for various 'n', with p_i, d_i random in $[0, 1]$ and $[0, \frac{1}{2}n]$ respectively. For $n = 4$, $W = \phi$ 95 times (out of 100) and LEJ was sub-opt. 3 times.

For $n = 8$, $W = \phi$ 77 times and LEJ was sub-optimal 7 times.

For $n = 16$, $W = \phi$ 28 times, LEJ was sub-optimal 26 times, and on average, the algorithm performed only two iterations after the initial LEJ.

The weighted deterministic problem

The addition of general weights w_i complicates the problem considerably, and no method matching the efficiency of the UDP algorithm is possible. One method [5] has been proposed, but it is unwieldy for our purposes and has been shown to be sub-optimal by a three-job case study [1, pp. 46-8].

Corresponding to Th. UD1, we have, when $d_1, d_2 \leq p_1 + p_2$

Th. WD1 $\quad d_1 \leq \max[p_2 + (1 - w_1^{-1}w_2)p_1,$

$$w_1^{-1}w_2 d_2 + w_1^{-1}(w_1 - w_2)(p_1 + p_2)]$$

$\Leftrightarrow$ J_1 precedes J_2 when $n = 2$ (see Figs. 2, 3)

but this does not lead to the analogous form of Th. UD2.

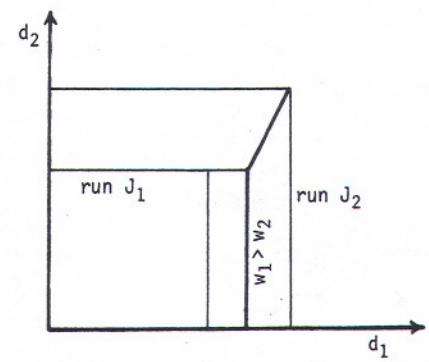


Fig. 2

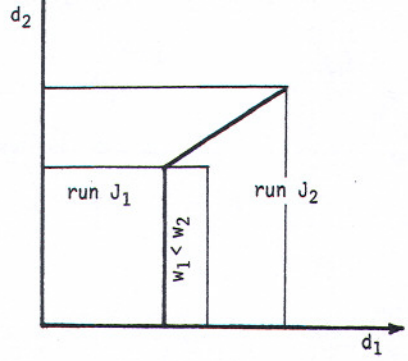


Fig. 3

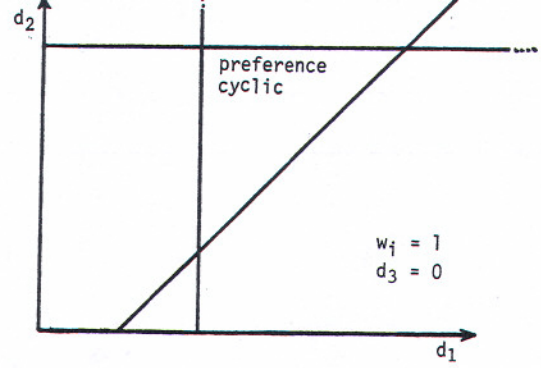


Fig. 4

The current goal is to find an algorithm which will reduce to the UDP algorithm of p.76 when we set the general weights to 1. In this direction we have:

Th. WD2 $\quad i<j<k$: $d_i \leqslant p_j + (1-w_i^{-1}w_j)p_i$

$$\Rightarrow \quad d_i \leqslant p_k + (1-w_i^{-1}w_k)p_i$$

analogous to $d_i < p_j \Rightarrow d_i < p_k$

It is a remarkable fact that general weights are more easily included in a stochastic study than in the deterministic study, and we can for the moment only ape the LEJ sequence, leaving the question of suitable optimality criteria and an iterative procedure open.

```
global  p, d, w, seq[1:n];
local   done[1:n], i, ℓ, place, pr ;
pr := 0;  for i := 1 to n do done[i] := false;
for place := 1 to n do
   {i := 1;  while done[i]  do i := i+1;  ℓ := i+1;
    while ℓ ≤ n  and wᵢpℓ + (wᵢ-wℓ)pᵢ < wᵢ(dᵢ-pr)  do
       {if not done[ℓ] and  wℓ(dℓ-pr)+(wᵢ-wℓ)(pᵢ+pℓ) < wᵢ(dᵢ-pr)
            then i := ℓ;
        ℓ :=ℓ+1  }
    seq[place] := i; pr := pr + proc[i];  done[i] := true  }}
```

The dynamic programming principle of optimality gives the equation

$$-\dot{C}(S,t) = e_S(t) + \min_{i\in S} p_i^{-1}[C(S_i,t) - C(S,t)]$$

where $S \subset \{1,\ldots,n\}$; $S_i = S - \{i\}$; $Z = \{i \mid i\in S,\ d_i \leqslant t\}$; $e_S(t) = \sum_{i\in Z} w_i$ and $C(S,t)$ is the expected additional future cost at time t, given that we follow the optimal policy and have to schedule the jobs J_i, $i \in S$.

It is useful to know that the form of the cost function is

$$C = l(t) + \sum_{i=1}^{n} a_i(t).\exp(t/p_i)$$

where the coefficients $l(t)$, $a_i(t)$ are constant in half-open intervals $[c,d)$. There is only a finite number of these intervals.

The stochastic problem

We add conditions of uncertainty to the problem in the most convenient way by assuming that the processing-time of J_i is a random variable exponentially distributed with known mean p_i. The probability of J_i terminating does not then depend on the time for which it has already run, and so this simplification leads to a Markov system. More general probability distributions and cost functions are treated in [3] for the case $n = 2$.

The result for $n=2$ is

Th.S1

$$d_1 \leqslant d_2 + p_1 \log_e(w_1p_2)/(w_2p_1) \iff \text{run } J_1 \quad (J_1 < J_2)$$

Note that with three jobs, the situation $J_1 < J_2$, $J_2 < J_3$ and $J_3 < J_1$ may arise! (See Fig. 4.)

The best-decision areas for $w_i = 1$, $d_3 = 0$ and $p_i = \exp(i-1)$ are depicted in the graph on p.77. It is conjectured that the decision at time $t \leq d_3$ is the decision at $t = d_3$ for $n = 3$, in which case the graph is sufficient representation of the solution.

Since numerical techniques are involved in finding the decision areas, we turn to more convenient sub-optimal policies for implementation.

Sub-optimal stochastic policies

The graph on p.77 compares the optimal decision rule with three sub-optimal rules. The quartet '$ijkl$' in each region means "choice 'i' with optimal rule, choice 'j' with rule A, 'k' with B and 'l' with C".

A. If we insist that when we switch from J_j to J_i using this rule, J_j is the best choice in the absence of J_i, and vice versa, then the boundaries can be calculated algebraically. Where they differ from those of the optimal policy and those of policy B below, they are marked by dotted lines. Figure 5 indicates how the order of preference of the jobs at any one time changes in a simple case.

B. With $i < j$, define $J_i > J_j$ as

$$d_i - d_j > p_i \cdot \log_e[(w_i p_j)/(w_j p_i)]$$

$$Z = \{j \mid J_i > J_j \ \forall\ i<j\} \ \exists\ k \in Z \quad \text{s.t.} \quad k \geq j \quad \forall j \in Z$$

The decision of rule B is to run J_k, and this is analogous to the LEJ sequence in the deterministic case.

C. $p_i + d_i$ acts as an index when all $w_i = 1$. With rule C, we run J_k where $p_k + d_k \leq p_i + d_i \ \forall\ i$.

D. Also when the $w_i = 1$, we might apply the deterministic rule. This is not shown on the graph as it simply divides the area depicted between J_1 and J_2.

Rule B is the best for the purposes of scheduling in real-time systems as it is applicable with general weights, quick to calculate and in close agreement with the optimal decision rule. In many applications, including the one studied in the next section, the other modelling assumptions being made will make the difference between rule B and the optimal rule insignificant.

One application: an on-line facility

The primary purpose of processor scheduling in an on-line system is to maintain the illusion for each user that the computer is dedicated to him. Most of the scheduling algorithms proposed and implemented are based on resource considerations only such as time quanta, time already spent on a process, or – if the process is not represented in core – storage requirements.

The illusion mentioned above will be achieved if all output channels are kept busy, and the proposal of this section is that we can come closer to that goal by monitoring the output performance of each process.

The on-line system modelled here is one in which 'n' users U_i at separate consoles have each submitted a task J_i. The number 'n' varies with time, but we make no attempt to anticipate the arrival of new tasks: later arrivals pre-empt the currently running process if necessary. An amount $d_i(t)$ of output has been created by J_i but not received by U_i, and this backlog, measured by the time it will take to clear, gives us a due-date for J_i. A further time $d_i(t)$ may elapse before J_i produces more output to maintain the continuity of the data stream. p_i is an estimate of the time required by J_i to produce a unit of output, and a variety of simple heuristics are available to decide this parameter. The weight w_i might represent an amalgamation of factors such as the status of U_i, the time already received by J_i and the output already received by U_i.

The stochastic model with sub-optimal policy 'B' seems appropriate given the approximations above. Since $\underline{p}$, $\underline{w}$, $\underline{d}$ and 'n' are all functions of time, the algorithm must be quantised in some way to maintain CPU utilisation and control housekeeping overheads.

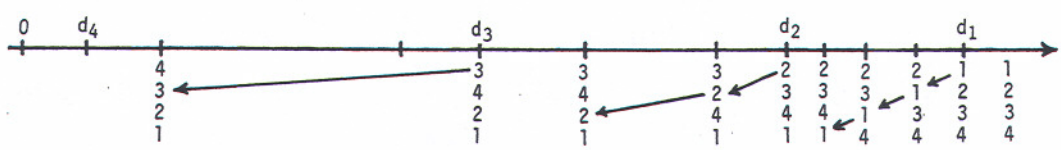


Fig. 5 Illustrating how the preference order can change with sub-optimal strategy A in the stochastic case. The calculations are done in backwards time.

Acknowledgement

I should like to thank Professor J.F. Coales and Dr J.D. Roberts for supporting this contribution to the Seminar, and P. Nash for the stimulus of a continuing dialogue on the ramifications of the stochastic problem.

1. CONWAY, R.W., MAXWELL, W.L., and MILLER, L.W., 'Theory of scheduling', Addison-Wesley Publishing Corporation (1967).
2. EMMONS, H., 'One machine sequencing to minimise certain functions of job tardiness', Operations Research, V17, pp. 701-715 (1969).
3. GITTINS, J.C., 'Optimal resource allocation in chemical research', Advances in Applied Probability, V1, pp. 238-270 (1969).
4. HAWORTH, G.McC., 'Tardiness scheduling and computer systems', Cambridge University Engineering Department Technical Report CUED/B-Control/TR22 (1972).
5. SCHILD, A., and FREDMAN, I.J., 'Scheduling tasks with deadlines and linear loss functions', Management Science, V7, pp. 280-285 (1961).

Discussion

Q. Is the algorithm for the Weighted Deterministic Problem linear?

A. No, the execution time is proportional to $N \log N$. (Similarly for the Unweighted Deterministic Problem.)

Q. What happens when a new job is added?

A. Its priority has to be decided, and then the algorithm used to determine the schedule for the new set of jobs.

Q. How does your method compare with a pure priority scheme?

A. It works as though each job has a gradually increasing priority.