Scheduling linearly deteriorating jobs on parallel machines: a simulated annealing approach

KHALIL S. HINDI and SAMSON MHLANGA

Keywords scheduling, linearly deteriorating jobs, steepest descent, simulated annealing

Abstract. Scheduling deteriorating jobs on parallel machines is an NP-hard problem, for which heuristics would be the first solution option. Two variants of linearly deteriorating jobs are considered. The first is that with simple linear deterioration, i.e. where there is a deterioration rate only, which is meaningful only if the jobs are assumed to be available at a positive time \( t_0 \). In the second variant, there is a basic processing time and a deterioration rate and all jobs are available at time \( t = 0 \). In both cases, we seek to minimize the makespan.

Starting from simple heuristics, both steepest descent search and simulated annealing are designed and implemented to arrive at optimal or near-optimal solutions.

Computational results for randomly generated problem instances with different job/machine combinations are presented.

1. Introduction

Usually, scheduling problems involve jobs with constant, independent processing times. However, situations arise where processing times are not constant but increasing over time, i.e. deteriorating, and, therefore, interdependent. The associated scheduling problems occur, e.g. in maintenance scheduling and cleaning assignments, and also in contexts where machines are deteriorating, so that jobs processed later require longer processing times.

Browne and Yechiali (1990) introduced the problem of scheduling deteriorating jobs on a single machine, where \( p_i(t) = a_i + \alpha_i t \) is the processing time for job \( i \) when processing begins at \( t_i \), \( a_i \) is the basic processing time, and \( 0 \leq \alpha_i \leq 1 \) is the deterioration rate. Mosheiov (1991) investigated minimizing the total flow time when all
jobs have the same basic processing time and showed that the optimal sequence is V-shaped. He also studied the case where the basic processing time is zero, considering some of the most commonly used performance measures (Mosheiov 1994). In a third work, Mosheiov (1995) considered scheduling jobs with step-deterioration on a single machine, as well as on a number of identical machines, with the objective of minimizing the makespan.

Recently, Hsieh and Bricker (1997) studied the multi-processor version of the problem, and developed several heuristics for both the case where the basic processing time is zero and the case where it is not. In the present work, we also address the multi-processor version of the problem and develop both a steepest descent search (DS) and a simulated annealing search (SA).

2. Scheduling of deteriorating jobs

Two cases are considered: the first is the case of simple linear deterioration, where there is a deterioration rate only, which is meaningful only if the jobs are assumed to be available at a positive time \( t_0 \). In the second case, there is a basic processing time and a deterioration rate and all jobs are available at time \( t = 0 \). In both cases, we seek to minimize the makespan.

2.1. Simple linear deterioration

Consider first scheduling \( n \) jobs on a single machine, where \( p_i(t) = a_i t \) and the jobs are available at time \( t_0 \). It is easy to see that for any job sequence, the completion time of any job \( i \) is

\[
C_i = t_0 \prod_{k=1}^{i} (1 + \alpha_k),
\]

from which, the makespan is clearly independent of job sequence and is equal to:

\[
C_n = t_0 \prod_{k=1}^{n} (1 + \alpha_k).
\]

Now consider the multi-machine case, with \( m \) identical machines. Analogy with the ‘longest processing time (LPT) first’ heuristic suggests the following simple heuristic, which may be called a ‘longest deterioration time (LDT) first’ heuristic (Hsieh and Bricker 1997, Mhlanga 1997): order the jobs in the non-increasing order of deterioration rate, then take each job in turn and assign it to the machine that has the shortest makespan for the partial schedule assembled so far.

Hsieh and Bricker (1997) show that this heuristic has the asymptotic optimality property; i.e. the makespan achieved by it tends to equal the optimal makespan as \( n \to \infty \), and should therefore give good results, especially when \( n \gg m \). They also show that

\[
LB = \left[ \prod_{i=1}^{n} (1 + \alpha_i) \right]^{1/m}
\]

is a lower bound on the optimal value of the makespan.

Clearly, because the order of jobs on each machine is of no importance, the task is to partition the set of \( n \) jobs into \( m \) partitions and assign each subset to a single machine, such that the longest completion time is minimized. It is this idea that we use in the local-search-based algorithms developed in this work.

2.2. Jobs with basic processing time

Consider first scheduling \( n \) jobs, all available for processing at time \( t = 0 \), on a single machine. Each job has a basic job-specific processing time \( a_i \) (the random time to complete job \( i \) if it is processed first). If the processing of a job is delayed, the initial requirement deteriorates such that the processing time grows linearly with the delay:

\[
p_i(t) = a_i + \alpha_i t,
\]

where \( \alpha_i \) is the deterioration rate. It is also assumed that deterioration stops as soon as processing starts.

If the \( n \) jobs are to be processed non-preemptively on a single machine and we do not allow the machine to stay idle if there are jobs waiting, then we need to consider only permutations of the index set \( I = \{1, 2, \ldots, n\} \).

Now consider a particular permutation (sequence) \( \pi \). Let \( Y_i \) be the actual processing time of job \( i \) and \( C_i = \sum_{k=1}^{i} Y_i \) be the completion time of job \( i \). It is then easy to see that

\[
C_i - (1 + \alpha_i)C_{i-1} = a_i,
\]

which exhibits the solution

\[
C_i = \sum_{k=1}^{i} a_k \prod_{r=k+1}^{i} (1 + \alpha_r).
\]

Because the corresponding makespan is \( C_n \), it follows that:

\[
C_n = \sum_{k=1}^{n} a_k \prod_{r=k+1}^{n} (1 + \alpha_r).
\]

Rau (1971) had shown that the sum

\[
\sum_{k=1}^{n} \mu_k \prod_{r=k+1}^{n} \gamma_r
\]

is minimized when calculated over the permutation ordered by non-decreasing values of \( \mu_k/(\gamma_k - 1) \). Using
this result, Browne and Yechiali (1990) conclude that minimizing the makespan is clearly achieved by scheduling the jobs by non-decreasing values of $a_i/\alpha_i$.

This conclusion naturally leads in the case of scheduling the jobs on several identical machines to the following simple heuristic: order the jobs by non-decreasing values of $a_i/\alpha_i$, then take each job in turn and assign it to the machine that has the shortest makespan for the partial schedule assembled so far.

However, in this case also the task is, once again, to partition the set of $n$ jobs into $m$ subsets and assign each to a single machine such that the longest completion time is minimized, knowing a priori that each subset must be placed on its respective machine in the proper order of non-decreasing values of $a_i/\alpha_i$. This idea is used in the local-search-based algorithms developed in this work.

3. Steepest descent search (SD)

SD is started with a partition of the $n$ jobs into $m$ partitions. This partition could either be random or that arrived at by a suitable heuristic; e.g. the LDT heuristic for the simple deterioration case. At each step, the best partition in the neighbourhood of the current one, i.e. the partition that leads to the largest decrease in the makespan is identified and adopted. The search is terminated when the current partition cannot be improved upon.

The neighbourhood is defined as the set of partitions that can be achieved by a single move from the current partition. A single move could be a transfer of one job from one subset to another or a swap of two jobs belonging to two different subsets. However, because only a move involving the jobs assigned to the machine with the longest completion time, i.e. the machine associated with the makespan, will affect the makespan, the neighbourhood is defined dynamically by these moves. Thus, at each step, the following is carried out.

Step 1. Calculate the makespan and identify the associated machine and the jobs assigned to it.

Step 2. Evaluate all the moves (transfers and swaps) involving the jobs on the makespan machine and identify the one that would lead to the biggest improvement of the makespan. If there is no such move, stop, the current solution is a local optimum.

Step 3. Carry out the move identified in step 2 and go to step 1.

It is worth noting that the calculation of the makespan in step 1 and the worth of moves in step 2 is carried out incrementally, rather than from scratch.

4. Simulated annealing implementation

Proposed initially by Kirkpatrick et al. (1983), simulated annealing (SA) has been applied successfully to a large number of different combinatorial optimization problems. It is based on an analogy between the process of finding an optimal solution of a combinatorial optimization problem and the process of annealing of a solid to its minimum energy state in statistical physics.

SA employs a randomized move acceptance criterion in order to escape poor quality local minima. Whereas local search descent methods, like the steepest descent algorithm presented in section 3, do not accept non-improving moves at any iteration, SA does with certain probabilities. These probabilities are determined by a control parameter $T$, called temperature, which tends to zero according to a deterministic cooling schedule. For the problem in hand, the monotonic cooling schedule of Lundy and Mees (1986) was adopted.

The steps of the algorithm designed for the problem in hand are as follows.

Step 1. Adopt the solution given by the steepest descent algorithm as an initial solution $S$.

Step 2. Calculate the initial temperature (Aarts and Van Laarhoven 1985)

$$T_s = \Delta^+ \cdot \left[ \ln \left( \frac{m^+}{(1-x)\cdot(m-m^+)} \right) \right]^{-1},$$

where $m^+$ is the number of cost increase moves found during the steepest descent search, $\Delta^+$ is the average cost increase over these moves and $0 < x < 1$ is the acceptance ratio. In the current scheme $x$ was set to 0.95.

Step 3. Select a solution $S' \in N(S)$. This is carried out by identifying the best move involving the jobs on the current makespan machine.

Step 4. Let $\Delta = C(S') - C$. If $\Delta < 0$ or $e^{(-\Delta/T_k)} \geq \theta$, where $\theta$ is a uniform random parameter $0 < \theta < 1$, then accept the new solution and set $S = S'$, otherwise, retain $S$.

Step 5. Decrement the temperature:

$$T_{k+1} = \frac{T_k}{1 + \beta \cdot T_k},$$

where $\beta \ll 1/U$, and $U$ is an upper bound on the absolute value of moves. In the current scheme $\beta$ was set to 0.0001/$U$, with $U$ being the largest absolute move value found during the steepest descent.

Step 6. Stop if the stopping criterion is met; otherwise go to step 3. For the case of simple linear deterioration, the stopping criterion chosen is proximity to the lower bound or reaching the final tempera-
ture, while for the case with basic processing time, the criterion chosen was reaching the final temperature. Proximity to the lower bound was assessed by calculating the gap:

\[ g = \frac{\text{Solution cost} - \text{Lower bound}}{\text{Lower bound}}. \]

If \( g \leq 0.5\% \), the solution is deemed to be close enough to the optimum. The final temperature \( T_f \) was set to \( 0.001 \times T_s \).

5. Computational experience

The effectiveness of the developed algorithms was assessed through extensive computational experimentation on 16 problem classes, each with a different jobs/machines combination. Twenty problem instances of each class were created, with deterioration rates randomly varying in the interval \((0, 1)\), while the basic processing times for the relevant case were chosen from a normal distribution, with a mean of 50 and a standard deviation of 10. All algorithms were coded in Turbo Pascal and run on a Pentium 133 MHz PC.

5.1. Simple linear deterioration

For each problem instance, the heuristic is applied first. If the gap with respect to the lower bound is greater than 0.5\% then the search is continued using the steepest descent method. If the local optimum arrived at in this way still has a gap greater than 0.5\%, the search for a better solution is continued by simulated annealing.

Table 1 shows for each problem class, the number of problem instances solved at the first two stages, as well as the number of the remaining instances that were tackled by SA. The average computing times for SD and SA are shown, but for the heuristic these were so small as to be negligible.

The results show that as the number of jobs increases the problem becomes easier; the problem instances from the classes with 100 and 500 jobs are almost all solved in the first two stages, without recourse to SA. It is also clear that for the same number of jobs, problem difficulty is affected by number of machines; the higher the more difficult. For the more difficult problem instances, SA is clearly superior to the heuristic and to SD. The results also show that solution times are very small, even for those problem instances where SA proved to be necessary. The efficacy and efficiency of both SD and SA are attributable to the judicious choice of moves.

5.2. Jobs with basic processing time

In the absence of a computed lower bound, every problem instance was tackled by a three-stage solution process: the heuristic is applied first to provide an initial solution, followed by SD then SA. The results obtained show that SA clearly outperforms SD. However, the question arises as to whether SA would still be superior to a random search that employs SD. To resolve this question, SD was run 20 times, each time starting from a random partition of jobs, and the best result obtained

| Problem class | Heuristic | SD | Av. comp. time (s) | SA | Av. gap | Av. comp. time (s) |
|---------------|-----------|----|-------------------|----|---------|-------------------|
| 10/2          | 2         | 8  | 0.00              | 10 | 0.50    | 0.01              |
| 10/3          | 0         | 1  | 0.00              | 19 | 1.47    | 0.69              |
| 20/6          | 0         | 0  | —                 | 20 | 0.88    | 1.35              |
| 50/4          | 1         | 18 | 0.02              | 2  | 0.45    | 0.00              |
| 50/6          | 0         | 18 | 0.04              | 4  | 0.49    | 0.00              |
| 50/7          | 0         | 16 | 0.04              | 6  | 0.63    | 0.04              |
| 50/8          | 0         | 14 | 0.03              | 5  | 0.56    | 0.09              |
| 50/9          | 0         | 15 | 0.07              | 15 | 0.50    | 0.05              |
| 50/10         | 0         | 5  | 0.03              |    |         |                   |
| 100/7         | 0         | 20 | 0.20              |    |         |                   |
| 100/8         | 0         | 20 | 0.26              |    |         |                   |
| 100/9         | 0         | 19 | 0.21              | 1  | 0.84    | 0.11              |
| 100/10        | 0         | 19 | 0.19              | 1  | 0.26    | 0.05              |
| 500/8         | 2         | 18 | 8.08              |    |         |                   |
| 500/9         | 0         | 20 | 9.32              |    |         |                   |
| 500/10        | 4         | 16 | 7.64              |    |         |                   |
was then compared with SA. The results of this experiment, shown in Table 2, show that SA is still superior, albeit with a small margin. Computing times for both SD and SA were similar to those for the simple deterioration case.

### 6. Conclusion

Scheduling problems usually involve jobs with constant, independent processing times. However, situations arise where processing times are not constant but increasing over time, i.e., deteriorating, and, therefore, interdependent.

In this work, scheduling jobs with linear deterioration on parallel, identical machines has been considered. A steepest descent search scheme and a simulated annealing search scheme for solving the resulting NP-hard problem have been designed and implemented. The efficacy and efficiency of these schemes, as shown by the results of extensive computational experimentation on a large set of randomly generated problem instances, is attributable to the effective move strategy designed for them.

Search schemes similar to those developed in this work should be of use in designing solution algorithms for other scheduling on parallel machines problems, involving deterioration models other than those considered.

### References

AARTS, E. H. L., and VAN LAARHOVEN, P. J. M., 1985, Statistical cooling: a general approach to combinatorial optimization problems. *Philips Journal of Research*, 40, 193–226.

BROWNE, S., and YECHIALI, U., 1990, Scheduling deteriorating jobs on a single processor. *Operations Research*, 38, 495–498.

HSIEH, Y.-C., and BRICKER, D. L., 1997, Scheduling linearly deteriorating jobs on multiple machines. *Computers and Industrial Engineering*, 32, 727–734.

KIRKPATRICK, S., GELATT, C. D. Jr., and Vecchi, M. P., 1983, Optimisation by simulated annealing. *Science*, 220, 671–680.

LUNDY, M., and MEE, A., 1986, Convergence of an annealing algorithm. *Mathematical Programming* 34, 111–124.

MHLANGA, S., 1997, Scheduling linearly deteriorating jobs, MSc dissertation. Department of Systems Engineering, Brunel University.

MOSHEIOV, G., 1991, V-shaped policies for scheduling deteriorating jobs. *Operations Research*, 39, 979–991.

MOSHEIOV, G., 1994, Scheduling jobs under simple linear deterioration. *Computers and Operations Research*, 21, 653–659.

MOSHEIOV, G., 1995, Scheduling jobs with step-deterioration: minimizing makespan on a single- and multi-machine. *Computers and Industrial Engineering*, 28, 869–879.

RAU, G., 1971, Minimising a function of permutations of n integers. *Operations Research*, 19, 237–240.