

The Strength of Priority Algorithms

Yakov Zinder

University of Technology, Sydney, Department of Mathematics, PO Box 123, Broadway, NSW 2007, Australia,
yakov.zinder@uts.edu.au

The paper is concerned with scheduling problems with partially ordered unit execution time tasks and parallel identical machines. For the algorithms, constituting a broad class of algorithms for the maximum lateness problem, the paper introduces the notion of a strength, which characterizes the algorithm's worst-case performance. Using this formal framework, a positive answer is given to the question of the existence of a strongest algorithm, i.e. an algorithm with the best worst-case performance. The idea of this algorithm can be generalized to the maximum lateness problem with release times. The paper shows that this idea in combination with the idea of the Garey-Johnson algorithm leads to an exact algorithm for the maximum cost problem with release times and partially ordered unit execution time tasks.

Keywords: Machine Scheduling, Theoretical Scheduling

1 Scheduling Problems with UET Tasks

The problem of scheduling a partially ordered finite set of unit execution time (UET) tasks on parallel machines with the criterion of maximum lateness is one of the central in scheduling theory. This problem can be stated as follows. A finite set $N = \{1, \dots, n\}$ of tasks (jobs, operations) is to be processed on $m > 1$ identical machines (processors) subject to precedence constraints in the form of an anti-reflexive, anti-symmetric and transitive binary relation α on N . If task i precedes task j in α , i.e. $(i, j) \in \alpha$, then the processing of i must be completed before the processing of j begins. The processing of tasks commences at time $t = 0$. Each machine can process only one task at a time. Once a machine begins processing a task, it continues until completion, i.e. no preemptions are allowed. Each task can be processed by any machine and requires one unit of time. A schedule σ is a mapping which assigns to each task j a positive integer – its completion time $C_j(\sigma)$. Each task has a due time (also often referred to as a due date) d_j . The goal is to find a schedule that minimizes the criterion of maximum lateness

$$L_{max}(\sigma) = \max_{j \in N} [C_j(\sigma) - d_j].$$

All due dates are arbitrary rational numbers.

In the standard three-field notation (see for example [2]), this problem is denoted by $P|prec, p_j = 1|L_{max}$, where P and $prec$ indicate parallel identical machines and the presence of precedence constraints, the term $p_j = 1$ specifies that the processing time p_j of each task j is one unit of time, and L_{max} denotes the criterion of maximum lateness. If all due times are equal to zero, the maximum lateness problem $P|prec, p_j = 1|L_{max}$ becomes the makespan problem $P|prec, p_j = 1|C_{max}$ with the criterion

$$C_{max}(\sigma) = \max_{j \in N} C_j(\sigma).$$

It is well known that $P|prec, p_j = 1|C_{max}$, and therefore $P|prec, p_j = 1|L_{max}$, is NP-hard in the strong sense [12], [9]. Moreover, the $P|prec, p_j = 1|C_{max}$ problem remains NP-hard even if the partially ordered sets of tasks are restricted to the bipartite graphs [15]. The $P|prec, p_j = 1|L_{max}$

problem remains NP-hard even if the partially ordered sets of tasks are restricted to the out-trees [3]. The NP-hardness results boosted the interest to the worst-case performance of various approximation algorithms. The majority of these algorithms were originally developed for some particular cases of the $P|prec, p_j = 1|L_{max}$ problem and are different implementations of the same approach. A class of algorithms, implementing a generalization of this approach, was introduced in [14]. Following [14], these algorithms will be referred to as priority algorithms.

In what follows, we will analyze the worst-case performance of priority algorithms, introducing the notion of a strength, and will give a positive answer to the question of the existence of a strongest priority algorithm (with the best performance guarantee). In answering this question we will consider not only currently known algorithms, but also “yet to be discovered” algorithms, i.e. all possible algorithms in the considered class.

A straightforward generalization of the $P|prec, p_j = 1|L_{max}$ problem is the problem with release times, where the processing of each task j can commence only after the nonnegative integer release time r_j . We will assume that $\min_{j \in N} r_j = 0$. In the three-field notation, this problem is denoted by $P|prec, p_j = 1, r_j|L_{max}$. In the problem with unit communication delays, denoted in the three-field notation by $P|prec, p_j = 1, r_j, c_{ij} = 1|L_{max}$, in any feasible schedule σ , for each task j , there is at most one task i such that $(i, j) \in \alpha$ and $C_i(\sigma) = C_j(\sigma) - 1$ and there is at most one task g such that $(j, g) \in \alpha$ and $C_g(\sigma) = C_j(\sigma) + 1$. It will be shown how the idea implemented in the strongest priority algorithm can be combined with the idea in [6] and [7] in order to obtain exact algorithms for $P|prec, p_j = 1, r_j | \max \varphi_j(C_j(\sigma))$ and $P|prec, p_j = 1, r_j, c_{ij} = 1 | \max \varphi_j(C_j(\sigma))$, where all φ_j are nondecreasing functions.

2 Priority Algorithms

A generalization of the approach, implemented in [3], [5] and [16] for the $P|prec, p_j = 1|L_{max}$ problem, and in [1], [4], [8], [10], [11] for a particular case of the maximum lateness problem, the makespan problem $P|prec, p_j = 1|C_{max}$, was considered in [14]. Following [14], for an arbitrary instance \mathcal{P} of the $P|prec, p_j = 1|L_{max}$ problem, the corresponding partially ordered set of tasks will be denoted by $(N(\mathcal{P}), \alpha(\mathcal{P}))$ and the corresponding sets of due times will be denoted by $\{d_j(\mathcal{P}) : j \in N(\mathcal{P})\}$. A subset $Q \subseteq N(\mathcal{P})$ is $\alpha(\mathcal{P})$ -closed if, for any $j \in Q$ and any $g \in N(\mathcal{P})$, the relation $(j, g) \in \alpha(\mathcal{P})$ implies $g \in Q$. For any partially ordered set (M, δ) and any $Q \subset M$,

$$\delta_Q = \delta \cap (Q \times Q).$$

For any instance \mathcal{P} of the $P|prec, p_j = 1|L_{max}$ problem and any $\alpha(\mathcal{P})$ -closed subset $Q \subseteq N(\mathcal{P})$, \mathcal{P}_Q is the instance of $P|prec, p_j = 1|L_{max}$ with the partially ordered set of tasks $(Q, \alpha(\mathcal{P})_Q)$ and the set of due times $\{d_j(\mathcal{P}_Q) : j \in Q\}$ such that $d_j(\mathcal{P}_Q) = d_j(\mathcal{P})$ for all $j \in Q$.

A priority algorithm for the $P|prec, p_j = 1|L_{max}$ problem is an algorithm that, for any instance \mathcal{P} of this problem, first determines an anti-reflexive, anti-symmetric and transitive relation $\delta(\mathcal{P})$ on $N(\mathcal{P})$, satisfying

$$\delta(\mathcal{P}_Q) \subseteq \delta(\mathcal{P})_Q$$

for any $\alpha(\mathcal{P})$ -closed subset $Q \subseteq N(\mathcal{P})$, and then applies the List Algorithm to a result of topological sorting of the partially ordered set $(N(\mathcal{P}), \delta(\mathcal{P}))$.

Although [14] is concerned with the $P|prec, p_j = 1|L_{max}$ problem, i.e. the problem where all the release times are equal to zero, for the purpose of Section 4 it is convenient to describe the List Algorithm, assuming that each task j can be processed only after the associated nonnegative integer release time r_j . In this case, without loss of generality assume that if $(i, j) \in \alpha$, then $r_i \leq r_j - 1$. The List Algorithm is an iterative procedure assigning the completion times in order

which is determined by a list of tasks. Initially this list contains all tasks. Once a task is assigned its completion time, it is deleted from the current list. Let \mathcal{L} be the current list of tasks. For any task j , let $\mathcal{L}(j)$ be the position of this task in \mathcal{L} . For example, if $\mathcal{L} = (j_1, \dots, j_u)$, then $\mathcal{L}(j_k) = k$. Let $|\mathcal{L}|$ be the number of tasks in the current list. Let $\bar{\sigma}$ be a schedule constructed by the List Algorithm. Schedule $\bar{\sigma}$ will be referred to as a list schedule.

List Algorithm

1. Set $t = \min_{j \in \mathcal{L}} r_j + 1 = 1$, $i = 1$ and $w = 0$.
2. Scan \mathcal{L} from left to right, starting from the task in position i , and find the first task j such that $r_j < t$ and $C_i(\bar{\sigma}) < t$ for all i such that $(i, j) \in \alpha$. If such j does not exist, go to Step 4.
3. Set $C_j(\bar{\sigma}) = t$, $i = \mathcal{L}(j)$, $w = w + 1$, and eliminate j from the list. If $|\mathcal{L}| = 0$, then stop. If $w < m$ and $i \leq |\mathcal{L}|$, then go to Step 2.
4. Set $t = \min[t, \min_{j \in \mathcal{L}} r_j] + 1$, $w = 0$, $i = 1$. Go to Step 2.

The definition of a priority algorithm does not specify how the relation $\delta(\mathcal{P})$ is determined. For example, the definition does not exclude algorithms that establish $\delta(\mathcal{P})$ by enumerating all feasible schedules or some very large subset of the set of all feasible schedules. The relation $\delta(\mathcal{P})$ can be interpreted as an assignment of tasks' priorities, where task j is of higher priority than task g if $(j, g) \in \delta(\mathcal{P})$, and these two tasks are of the same priority if $(j, g) \notin \delta(\mathcal{P})$ and $(g, j) \notin \delta(\mathcal{P})$. The result of topological sorting $\mathcal{L} = (j_1, \dots, j_n)$ has the following property: for any j_i and j_k , the relation $(j_i, j_k) \in \delta(\mathcal{P})$ implies the inequality $i < k$. Therefore, each time when several tasks compete for the assignment of a completion time, the List Algorithm chooses among all these tasks a task with the highest priority.

The analysis of the class of priority algorithms, presented in [14], is based of the notion of a domain, which was introduced in [14] as a key characteristic of a priority algorithm. On the intuitive level, the domain of an algorithm is a family of partially ordered sets of tasks for which this algorithm can solve the $P|prec, p_j = 1|L_{max}$ problem. Rigorous definitions can be found in [14].

3 Strength of Priority Algorithms

Since the general $P|prec, p_j = 1|L_{max}$ problem is NP-hard in the strong sense, various algorithms, developed for this problem, are characterized by their worst-case performance. Commonly the worst-case performance of any algorithm A for the $P|prec, p_j = 1|L_{max}$ problem is described in the form

$$L_{max}^{\mathcal{P}}(\sigma_{\mathcal{P}}^A) \leq \gamma L_{max}^{\mathcal{P}}(\sigma_{\mathcal{P}}^*) + \beta \max_{j \in N(\mathcal{P})} d_j(\mathcal{P}) - \eta, \tag{1}$$

where \mathcal{P} is an arbitrary instance of the $P|prec, p_j = 1|L_{max}$ problem, $\sigma_{\mathcal{P}}^A$ is a schedule, constructed by A for the instance \mathcal{P} , $\sigma_{\mathcal{P}}^*$ is an optimal schedule for this instance,

$$L_{max}^{\mathcal{P}}(\sigma) = \max_{j \in N(\mathcal{P})} \{C_j(\sigma) - d_j(\mathcal{P})\},$$

and γ , β and η are some constants, which remain the same for all instances \mathcal{P} of the $P|prec, p_j = 1|L_{max}$ problem.

Lemma 1 For any priority algorithm A there exists a constant γ such that

$$L_{max}^{\mathcal{P}}(\sigma_{\mathcal{P}}^A) \leq \gamma L_{max}^{\mathcal{P}}(\sigma_{\mathcal{P}}^*) + (\gamma - 1) \max_{j \in N(\mathcal{P})} d_j(\mathcal{P}) \quad (2)$$

for all instances \mathcal{P} of the $P|prec, p_j = 1|L_{max}$ problem.

The series composition of partially ordered sets (N_1, α_1) and (N_2, α_2) is the partially ordered set $(N_1 \cup N_2, \alpha_1 \cup \alpha_2 \cup (N_1 \times N_2))$. Lemma 1 leads to the following definition. For any priority algorithm A , which domain is not empty and is closed with respect of the series composition, its strength, denoted by $\gamma(A)$, is the smallest γ , satisfying the inequality (2) for all instances \mathcal{P} of the $P|prec, p_j = 1|L_{max}$ problem. Consequently, priority algorithm A with strength $\gamma(A)$ is stronger than priority algorithm B with strength $\gamma(B)$ if $\gamma(A) \leq \gamma(B)$.

The above definitions raise the following questions:

- (a) Does there exist the strongest priority algorithm?
- (b) If the strongest priority algorithm exists, what is its strength?
- (c) If the strongest priority algorithm exists, what is its computational complexity?

The following theorem answers these questions.

Theorem 1 The polynomial-time algorithm, presented in [16], is the strongest priority algorithm.

As has been shown in [16], the polynomial-time algorithm, presented in this paper, satisfies the following performance guarantee

$$L_{max}^{\mathcal{P}}(\sigma_{\mathcal{P}}^A) \leq \left(2 - \frac{2}{m}\right) L_{max}^{\mathcal{P}}(\sigma_{\mathcal{P}}^*) + \left(1 - \frac{2}{m}\right) \max_{j \in N(\mathcal{P})} d_j(\mathcal{P}) - r(m), \quad (3)$$

where $\sigma_{\mathcal{P}}^A$ is a schedule constructed by this algorithm, $\sigma_{\mathcal{P}}^*$ is an optimal schedule for the maximum lateness problem, and

$$r(m) = \begin{cases} \frac{m-3}{m} & \text{for } m \text{ odd} \\ \frac{m-2}{m} & \text{for } m \text{ even.} \end{cases} \quad (4)$$

4 Minimization of Maximum Cost

The results of [14] and Theorem 1 characterize the capability of priority algorithms and motivate the exploration of other approaches. One of the well known algorithms, which does not satisfy the above definition of a apriority algorithm, is the classical Garey-Johnson algorithm [6]. Originally, this algorithm was intended for the $P2|prec, p_j = 1, r_j|L_{max}$ problem, i.e. the problem with only two machines, but allows a generalization for the case of an arbitrary number of machines [7].

In order to apply this generalization, which following [7] will be referred to as a GJ-algorithm, the original $P|prec, p_j = 1|L_{max}$ problem should be transformed into the $P|prec, p_j = 1, r_j|L_{max}$ problem. For a partially ordered set (N, α) , k -path from any $j \in N$ to any $g \in N$ is a sequence j_0, \dots, j_k such that $j_0 = j$, $j_k = g$, and for any $0 \leq i \leq k - 1$, $(j_i, j_{i+1}) \in \alpha$. If $(j, g) \in \alpha$, then let $l(j, g)$ be the largest k for which there exists a k -path from j to g . If $(j, g) \notin \alpha$, then let $l(j, g) = 0$. Then $P|prec, p_j = 1|L_{max}$ can be transformed into $P|prec, p_j = 1, r_j|L_{max}$ by introducing for each $j \in N$ the release time

$$r_j = \max_{i \in N} l(i, j).$$

As has been proven in [7] for any instance of the $P|prec, p_j = 1, r_j|L_{max}$ problem

$$L_{max}(\sigma^{GJ}) \leq [2 - \alpha(m)]L_{max}(\sigma^*) + [1 - \alpha(m)] \max_{v \in N} d_v - [1 - \alpha(m)],$$

where σ^{GJ} is a schedule constructed by the GJ-algorithm, σ^* is an optimal schedule, and

$$\alpha(m) = \begin{cases} \frac{2}{m+1} & \text{if } m \text{ is odd} \\ \frac{2}{m} & \text{if } m \text{ is even} \end{cases}.$$

Another algorithm for the $P|prec, p_j = 1, r_j|L_{max}$ problem was presented in [13] and has performance guarantee (3)-(4). This algorithm can be considered as a generalization of [16]. The ideas, presented in [6], [7] and [13], lead to the algorithm below, constructing an exact solution to the $P|prec, p_j = 1, r_j|\max \varphi_j(C_j)$ problem, where all φ_j are nondecreasing functions. Without loss of generality, we again assume that if $(i, j) \in \alpha$, then $r_j \geq r_i + 1$. We also assume that for any list schedule σ , $C_{max}(\sigma) \leq n$. If this condition does not hold, then the problem can be decomposed into two separate problems.

The algorithm below is based on the idea of Δ -modified due dates, which according to [7] (see also [6]) are defined as follows. Let D_1, \dots, D_n be arbitrary nonnegative integers, then for any task i and any two numbers s and h such that

$$r_i \leq s \leq D_i \leq h, \tag{5}$$

$S(i, s, h)$ denotes the set of all tasks j such that $j \neq i$, $D_j \leq h$, and either $(i, j) \in \alpha$ or $r_j \geq s$. Integers D_1, \dots, D_n are consistent if for every task j

$$r_j \leq D_j - 1,$$

and for any task i and any two integers s and h , satisfying (5), either $D_i = s$ and $|S(i, s, h)| = m(h - s)$, or $|S(i, s, h)| < m(h - s)$. For any integer Δ , integers D_1, \dots, D_n are Δ -modified due dates with respect to d_1, \dots, d_n , if they are consistent and $D_i \leq d_i + \Delta$ for all $i \in N$.

For any set of due dates $d = \{d_1, \dots, d_n\}$ let $\Delta(d)$ be the smallest Δ for which there exist Δ -modified due dates with respect to d_1, \dots, d_n . As has been shown in [7], for any schedule σ ,

$$\Delta(d) \leq \max_{j \in N} [C_j(\sigma) - d_j].$$

For any set of consistent due dates $x = \{x_1, \dots, x_n\}$, σ^x will denote a list schedule, corresponding to a list where tasks are arranged in a nondecreasing order of $\{x_1, \dots, x_n\}$. For any $i \in N$, $K(i)$ will denote the set of all tasks j such that $(i, j) \in \alpha$. For any positive integer t the time interval $[t - 1, t]$ will be referred to as a time slot t .

Algorithm

1. Set $V = \max_{j \in N} \varphi_j(r_j + 1)$.
2. For each $j \in N$, find d_j : the largest τ among all integers τ such that $\tau \leq n$ and $\varphi_j(\tau) \leq V$. For the set $d = \{d_1, \dots, d_n\}$, find $\Delta(d)$ and the corresponding set of $\Delta(d)$ -modified due dates, say $z = \{z_1, \dots, z_n\}$. Set $\delta = \Delta(d)$. If $\Delta(d) = 0$, then go to Step 4.

3. Set

$$V = \min_{\{j: d_j + \delta \leq n\}} \varphi_j(d_j + \delta)$$

and go to Step 2.

4. If $L(\sigma^z) = 0$, then stop: σ^z is an optimal schedule. Otherwise, set $D = \{z\}$ and $\delta = n$.

5. If $D = \emptyset$, then go to Step 3. Otherwise, among all sets of consistent due dates $y = \{y_1, \dots, y_n\} \in D$ select a set, say $x = \{x_1, \dots, x_n\}$, with the smallest value of

$$\max_{j \in N} [C_j(\sigma^y) - y_j].$$

Eliminate x from D .

6. Among all tasks $j \in N$, satisfying the equality

$$C_j(\sigma^x) - x_j = \max_{j \in N} [C_j(\sigma^x) - x_j],$$

select one with the smallest $C_j(\sigma^x)$, say task g . Select T : the largest t among all integers $t < C_g(\sigma^x)$ such that the number of tasks with the completion time t is less than m or there exists j with $C_j(\sigma^x) = t$ and $x_j > x_g$.

7. If $r_g < T$, then set

$$U = \{j : T < C_j(\sigma^x) < C_g(\sigma^x) \text{ and } r_j < T\}.$$

Otherwise, set

$$U = \{j : T < C_j(\sigma^x) < C_g(\sigma^x) \text{ and } r_j < T\} \cup \{g\}.$$

Set

$$J = \{j : C_j(\sigma^x) = T \text{ and } K(j) \cap U \neq \emptyset\}.$$

8. Select an arbitrary $j \in J$ and find $\Delta(y)$ and the corresponding set of $\Delta(y)$ -modified due dates, say $w = \{w_1, \dots, w_n\}$, for $y = \{y_1, \dots, y_n\}$ such that

$$y_i = \begin{cases} x_i, & \text{if } i \neq j \\ x_j = T - C_g(\sigma^x) + x_g, & \text{if } i = j \end{cases}$$

If $\Delta(y) > 0$, then set $\delta = \min[\delta, \Delta(y)]$ and to Step 10.

9. If $\max_{j \in N} \varphi_j(C_j(\sigma^w)) = V$, then stop: σ^w is an optimal schedule. Otherwise, set $D = D \cup \{w\}$.

10. If $J \neq \emptyset$, then go to Step 8. Otherwise, go to Step 5 .

The presented approach is also applicable to $P|prec, p_j = 1, r_j, c_{ij} = 1| \max \varphi_j(C_j(\sigma))$. The particular implementation of this approach can vary depending on cost functions φ_j . First computational experiments with an algorithm for the $P|prec, p_j = 1|C_{max}$ problem showed its high effectiveness [17].

References

- [1] B. Braschi and D. Trystram (1994), A new insight into the Coffman-Graham algorithm , *SIAM Journal on Computing* **23**, 662 – 669.
- [2] P. Brucker (2001), *Scheduling algorithms*, third edition, Springer, Berlin.
- [3] P. Brucker, M.R. Garey and D.S. Johnson (1977), Scheduling equal-length tasks under tree-like precedence constraints to minimise maximum lateness , *Math. Oper. Res.* **2**, 275 – 284.
- [4] E.G. Coffman, Jr and R.L. Graham (1972), Optimal scheduling for two-processor systems , *Acta Inform.* **1**, 200 – 213.
- [5] M.R. Garey and D.S. Johnson (1976), Scheduling tasks with nonuniform deadlines on two processors , *Journal of the Association for Computing Machinery* **23**, 461 – 467.
- [6] M. R. Garey and D.S. Johnson (1977), Two-processor scheduling with start-time and deadlines, *SIAM Journal on Computing* **6**, 416 – 426.
- [7] C. Hanen and Y. Zinder, The worst-case analysis of the Garey-Johnson algorithm, *submitted for publication*
- [8] T.C. Hu (1961), Parallel sequencing and assembly line problems , *Operation Research* **9**, 841 – 848.
- [9] J.K. Lenstra and A.H.G. Rinnooy Kan (1978), Complexity of scheduling under precedence constraints , *Oper. Res.* **26**, 22 – 35.
- [10] A. Moukrim (1999), Optimal scheduling on parallel machines for a new order class , *Operation research letters* **24**, 91 – 95.
- [11] C.H. Papadimitriou and M. Yannakakis (1979), Scheduling interval ordered tasks , *SIAM J. Comput.* **8**, 405 – 409.
- [12] J.D. Ullman (1975), NP-complete scheduling problems , *J. Comput. System Sci.* **10**, 384 – 393.
- [13] Y. Zinder, An iterative algorithm for scheduling UET tasks with due dates and release times, *European Journal of Operational Research* **149**, 404 – 416
- [14] Y. Zinder and V.H. Do, Priority algorithms for scheduling UET tasks: domains and dominance, *submitted for publication*
- [15] Y. Zinder and D. Roper (1995), A minimax combinatorial optimisation problem on an acyclic directed graph: polynomial-time algorithms and complexity , In: *Proceedings of the A.C. Aitken Centenary Conference*, Dunedin, 391 – 400.
- [16] Y. Zinder and D. Roper (1998), An iterative algorithm for scheduling unit-time operations with precedence constraints to minimise the maximum lateness, *Annals of Operations Research* **81**, 321 – 340.
- [17] Y. Zinder, G. Singh and R. Weiskircher (2006), A new method of scheduling UET tasks on parallel machines, *Lecture Notes in Engineering and Computer Science*, Hong Kong, 796 – 801