

# Worst Case Behavior of List Algorithms for Dynamic Scheduling of Non-Unit Execution Time Tasks with Arbitrary Precedence Constrains

Andrei TCHERNYKH<sup>†</sup>, *Member*, Klaus ECKER<sup>††</sup>, *Nonmember*

**SUMMARY** Performance properties of list scheduling algorithms under various dynamic assumptions are analyzed. The focus is on bounds for scheduling directed acyclic graphs with arbitrary precedence constrains and arbitrary task processing times subject to minimizing the makespan. New performance bounds are derived and compared with known results.

*key words:* list scheduling, worst case analysis, on-line scheduling.

## 1. Introduction

Although list scheduling algorithms have been under investigation for decades [1, 2, 8, 11, 12, 13, 15, 16], many important properties are still unexplored. Performance of list scheduling algorithms was studied for the case of unit (UET) and non unit execution time (NUET) tasks. For the case of UET tasks an analysis under various dynamic assumptions and different levels of knowledge available for scheduling is presented in [18]. Obviously, if arbitrary task processing times are allowed, we cannot get better bounds. The objective of this paper is to show that the bounds obtained for UET tasks remain valid even for arbitrary processing times.

We consider the problem of on-line scheduling tasks with arbitrary integer execution times, and generally assume that, at each point of time, we have knowledge about which tasks are ready to be started and what their execution time is. The performance of algorithms under various dynamic assumptions is analyzed. It is shown that, depending on the available knowledge, Graham's well known bound [9] may be improved.

This work is motivated by the dynamic scheduling of processes that spawn other processes. As a typical situation consider a parallel program represented by a dynamic precedence task graph (dynamic spawn task graph) that is constructed at run time. In such a model the set of tasks is indeed unknown in advance. Each node can be considered as a process (an instruction or a group of instructions) that is able to spawn other processes. In this case the precedence relation between tasks is interpreted as *the spawn*

*precedence*.  $T_i \prec T_j$  means that while task  $T_i$  is processing,

$T_j$  is created, and  $T_j$  can be started only after  $T_i$  is completed. A process can dynamically create child processes via, for instance, the *fork()* system call. This in turn can create further child processes and so on.

To estimate and verify the quality of a schedule obtained during the dynamic execution of the task system an analysis of performance bounds under different assumptions is performed. Additional knowledge such as the sum of processing times of all tasks and the weighted length of the longest chain in the task system could help to verify the worst case behavior and to justify a particular choice of algorithm.

The paper is organized as follows. Section 2 introduces the notation of scheduling problems. In Section 3, we first review results for the deterministic case of scheduling with arbitrary precedence constrains on a set of identical processors subject to minimizing the makespan. Next, the performance bound as a function of the length of the longest task chain in the precedence graph and the total workload is derived. Section 4 summarizes our conclusion.

## 2. Notation

Following the notation given in [2] for deterministic scheduling problems, we denote by  $T = \{T_1, \dots, T_n\}$  the set of  $n$  tasks and by  $p_j$  the task processing time of  $T_j$ . The order in which the tasks can be processed on  $m$  identical processors  $P_1, \dots, P_m$  is restricted by some precedence relation  $\prec$  over the set  $T$ . Furthermore, non-preemptive scheduling is assumed. Generally we deal with *greedy schedules*, i.e. tasks are assigned to processors as early as possible. This strategy does not allow a processor to be idle as long as tasks are available for processing.

We are looking for a schedule that executes a given set of tasks with precedence constraints in minimal possible time (scheduling problem with minimizing the makespan). Following [2] we describe such scheduling problem by the 3-field notation  $P | prec | C_{max}$  where the first component specifies the processor environment, the second the task characteristics, and the third the optimization objective.

Manuscript received January 22, 2007

Manuscript revised November 3, 2007

<sup>†</sup> The author is with CICESE Research Center, Ensenada, Mexico

<sup>††</sup> The author is with Ohio University, Athens Ohio, USA

a) E-mail: chernykh@cicese.mx

DOI: 10.1093/ietfec/e91-a.8.2277

A schedule is defined as a function  $S: \mathbb{R}^{\geq 0} \rightarrow (T \cup \{\varepsilon\})^m$ , where  $\varepsilon$  is the *idle task*, which is used to express the situation that a processor does not process any task, and  $S(t) = (T_{\alpha_1}, \dots, T_{\alpha_m})$  with  $T_{\alpha_j} \in T \cup \{\varepsilon\}$  for  $j = 1, \dots, m$  specifies tasks assigned to processors  $P_1, \dots, P_m$  at time  $t$ . Let  $I$  be a problem instance for  $P | prec | C_{max}$ , and let  $S_{opt}(I)$  be an optimal schedule for  $I$  with makespan  $C_{opt}(I)$ . There are many strategies for scheduling a given set of tasks.

We are dealing here with *list algorithms* where tasks are assigned numerical priorities, the tasks with higher priority are greedily processed first. For a list algorithm  $A$ , let  $S_A(I)$  be the corresponding schedule with makespan  $C_A(I)$ . Let  $r_A(I) = C_A(I) / C_{opt}(I)$  denote the *performance ratio of the algorithm  $A$  for the instance  $I$* . To measure the overall quality of  $A$  we define the *performance of  $A$*  by  $r_A = \sup \{ C_A(I) / C_{opt}(I) \mid I \text{ is a problem instance} \}$ . Let  $C_{list}(I)$  be the longest makespan of a schedule obtained by applying any list algorithm on an instance  $I$ . The worst case behavior of list algorithms denoted by  $r_W := \sup \{ r_A \mid A \text{ is an arbitrary list algorithm} \}$  gives an upper bound on the performance of any list algorithm. By  $C_{list}[C_{opt}]$  we denote the length of an arbitrary greedy [respectively: optimal] schedule.

For analyzing some given schedule  $S$ , the following notation may be useful. At time  $t \geq 0$ , the number of tasks being processed is denoted by  $n_t$ , and  $z_t := m - n_t$  is the number of idle processors (or tasks) at integer time  $t$ . Since processing times are assumed to be integer and preemptions are not allowed, task assignments will only change at integer points of time. Though tasks may arrive at any time, they are considered for scheduling only when a currently processed task is completed, i.e. tasks can be started only at integer times.

An *idle interval* is an interval when a processor does not process a task. Recall that in a greedy schedule, an idle interval can only occur if no tasks are available for processing. Let  $z = \sum_{t=0}^{C_{max}} z_t$  denote the sum of lengths of idle intervals in a schedule within the time span 0 (start time of the schedule) and  $C_{max}$  (end of the schedule). Likewise,  $z_{opt}$ ,  $z_A$ , and  $z_W$  are the sums of lengths of idle intervals in a (hypothetical) optimal schedule, a schedule generated by algorithm  $A$ , and a general greedy schedule, respectively.

Let  $p$  be a sum of processing times of all tasks  $p := \sum_{T_j \in T} p_j$ . This is referred to as the *total workload*.

During the analysis we need to consider chains of tasks. A chain  $L$  is defined as a sequence of tasks  $(T_{\alpha_1}, \dots, T_{\alpha_k})$  where adjacent pairs of tasks fulfil the precedence

constraints:  $T_{\alpha_j} \prec T_{\alpha_{j+1}}$  for  $j = 1, \dots, k-1$ . The (weighted) length of a chain is the sum of processing times of the tasks in the chain.

If  $L$  is a chain in instance  $I$ , then its weighted length  $l$  is defined as the sum of processing times in the chain,  $l = \sum \{ p_j \mid T_j \in L \}$ . Given an instance, we denote by  $L_C$ ,  $l_C$  the longest chain in the precedence graph and its length, respectively.

### 3. Scheduling non-preemptive NUET tasks under Dynamic Assumptions

#### 3.1 General properties of schedules

Recall first the following results from scheduling theory for the problem class  $P | prec | C_{max}$  i.e., for scheduling NUET tasks non-preemptively under static assumptions. The problem is NP-hard. Graham [9] proposed a list scheduling algorithm and proved that the performance bound  $C_{list} / C_{opt}$  is  $2 - 1/m$ . Kunde [14] showed that the performance bound for  $P | out-forest | C_{max}$  is generally no worse than  $2 - 2/(m+1)$ .

Some results are also available for subcases. If the precedence relation is of bounded width  $k$  (i.e. each subset of  $k+1$  tasks has at least one pair of dependent tasks), the problem  $P | width-k | C_{max}$  can be solved in  $O(n^k)$  time, even if processing times are arbitrary [7]. If two possible values for the task processing times are considered, problems  $P2 | prec, p_j = 1 \text{ or } 2 | C_{max}$  and  $P | prec, p_j = 1 \text{ or } k | C_{max}$  are NP-hard [3], while problems  $P2 | tree, p_j = 1 \text{ or } 2 | C_{max}$  and  $P2 | tree, p_j = 1 \text{ or } 3 | C_{max}$  are solvable in time  $O(n \log n)$  [17] and  $O(n^2 \log n)$  [4], respectively. Arbitrary processing times result in strong NP-hardness even for the case of chains scheduled on two processors (problem  $P2 | chains | C_{max}$ ) [5].

#### 3.2. Analysis

Next we mention some simple facts about properties of deterministic schedules (see [6, 10]); that is,

$$z_{list} \leq l_C(m-1), C_{list} \leq (p - l_C)/m + l_C, (p + z_{opt})/m \geq l_C. \quad (1)$$

Note that if a list schedule and an optimal schedule have the same length then they must also have the same sum of idle intervals. On the other hand, it is easy to find instances for which  $C_{list} = C_{opt}(2 - 1/m)$ , where the optimal schedule has no idles. The question is: does there exist an instance for which  $C_{list} = C_{opt}(2 - 1/m)$ , and  $S_{opt}$  still has idle intervals?

**Lemma 1.** Given a greedy and an optimal schedule for the same instance, with respective lengths  $C_{list}$  and  $C_{opt}$ . If  $C_{list} = C_{opt}(2-1/m)$  then  $C_{opt}$  does not have idle intervals.

*Proof.* An upper bound for the sum of idle intervals in  $S_{opt}$  can be easily obtained from  $z_{list} \leq l_c(m-1)$ , i.e. from  $z_{list} \leq C_{opt}(m-1)$ . Because of  $z_{list} - z_{opt} = (C_{list} - C_{opt})m$  we get  $z_{list} = (C_{list} - C_{opt})m + z_{opt} \leq l_c(m-1) \leq C_{opt}(m-1)$  or  $z_{opt} \leq (2m-1)C_{opt} - mC_{list}$ . Note that if  $C_{list} = (2-1/m)C_{opt}$  then  $z_{opt} \leq 0$ , i.e.,  $z_{opt} = 0$ .  $\square$

**Assumption 1:** No specific information is available for analysis.

It is easy to see that with list algorithms we cannot expect a better bound than that of Graham [9]. Consider a task system with  $m(m-1)$  tasks of length  $m-1$  ("long" tasks), and  $m(m-1)$  tasks of length 1 ("short" tasks), where the only precedence constraints are between short tasks. The optimal schedule is obtained if short tasks are processed with high priority (they use one processor for  $m(m-1)$  time units, and distribute long tasks among other  $m-1$  processors. The makespan is  $m(m-1)$ ). In contrast, a list algorithm may first processes long tasks (which keeps all  $m$  processors busy for  $(m-1)^2$  time units), and then the chain of short tasks (which takes additional  $m(m-1)$  time units). The makespan is  $(m-1)(2m-1)$ . Therefore the competitive ratio equals Graham's bound and is known to be the worst possible for list algorithms.

**Assumption 2:** The total workload and the weighted length of the longest chain in the task system are available for analysis.

Now, the performance bound as a function of the length of the longest task chain in the precedence graph and the total workload is derived for  $P|prec|C_{max}$ . It generalizes the result of [18] for  $P|prec, p_j=1|C_{max}$ .

**Theorem 1.** Given a set  $T$  of tasks, the performance of the general list strategy can be estimated by

$$r_w = \min\{r_w', r_w''\}, \text{ where} \quad (2)$$

$$r_w' \leq 1 + \frac{l_c}{p}(m-1) \text{ and } r_w'' \leq 1 + \frac{1}{m} \left( \frac{p}{l_c} - 1 \right)$$

Furthermore,  $r_w'$  is tight in the case of  $l_c \leq p/m$ , and  $r_w''$  is tight in the case of  $l_c > p/m$ .

*Proof.* For any list schedule  $S$ , there is a chain  $L$  with the weighted length  $l$  such that  $z_S \leq (m-1)l$ . Since  $l_c$  is the maximum possible weighted chain length of the task system,  $l_c(m-1)$  is the upper bound for the sum of idle times in any list schedule. For the length  $C_{opt}$  of a schedule with minimum makespan for the same task set, two lower bounds are given as  $C_{opt} \geq p/m$ , and  $C_{opt} \geq l_c$ . Hence using (1) the performance ratio can be estimated as follows:

$$\frac{C_{list}}{C_{opt}} \leq \frac{(p-l_c)/m + l_c}{p/m} = 1 + \frac{l_c}{p}(m-1) \text{ if } p/m \geq l_c \text{ and}$$

$$\frac{C_{list}}{C_{opt}} \leq \frac{(p-l_c)/m + l_c}{l_c} = 1 + \frac{1}{m} \left( \frac{p}{l_c} - 1 \right) \text{ if } p/m < l_c.$$

Since *list* is an arbitrary schedule, the upper bound for the performance ratio is determined by  $r_w$ .

To prove tightness of the bounds, notice first that

$$\begin{aligned} r_w &= 2 - 1/m && \text{in case of } l_c = p/m, \\ r_w' &< r_w'' && \text{for } l_c < p/m, \text{ and} \\ r_w' &> r_w'' && \text{for } l_c > p/m. \end{aligned}$$

Choose the following problem instance:  $p = km + l_c$ , where tasks are organized in  $m + 1$  chains, one of length  $l_c$  [chain  $L_c$ ] and  $m$  chains each of length  $k \in \mathbb{N}$ . An optimal schedule is obtained by assigning the tasks of  $L_c$  to one processor and the tasks of the other chains in a greedy way. The worst greedy schedule is obtained by scheduling first the tasks of the  $m$  chains, thus filling the first  $k$  time slots completely, and then assigning the tasks of  $L_c$ . The length of this schedule is  $k + l_c$ . If we choose  $l_c = p/m = km/(m-1)$ , i.e.  $p = km^2/(m-1)$ , we get  $C_{opt} = l_c$ , and  $r_w = 2 - 1/m$ .

If  $l_c < km/(m-1)$  we get  $C_{opt} = l_c + (km - l_c(m-1))/m$ ; the ratio  $r_w = C_w/C_{opt}$  in this case is  $1 + l_c(m-1)/p = r_w'$ .

If  $l_c > km/(m-1)$  we get  $C_{opt} = l_c$ , and the ratio  $r_w = C_w/C_{opt}$  is  $1 + \frac{1}{m} \left( \frac{p}{l_c} - 1 \right) = r_w''$ .  $\square$

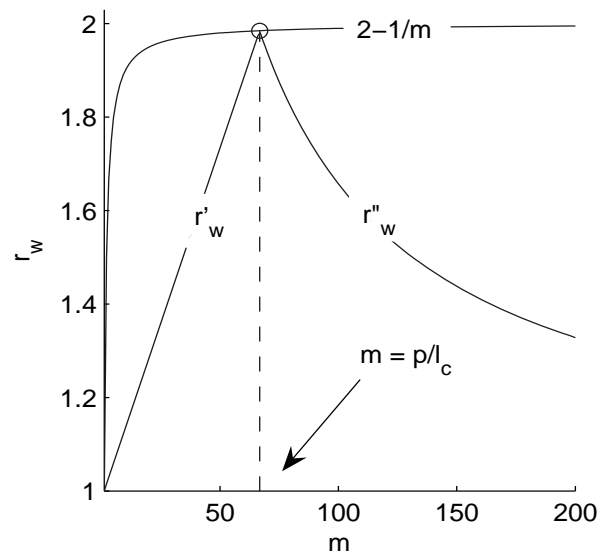


Fig. 1. Worst case ratio  $r_w$ , when keeping the total workload and the length  $l_c$  fixed ( $p = 20000$ ,  $l_c = 300$ ) and varying the number of machines.

Figure 1 shows the worst case performance, when keeping the total workload and the length  $l_c$  fixed, and varying the number of machines. Our new bound reaches the maximum value of Graham's bound in the point  $m = p/l_c$ , and tends to 1 while the number of machines is increased. For the both regions  $m < p/l_c$  and  $m > p/l_c$  the obtained bound is better. This result allows predicting the performance of the dynamic schedule. Furthermore, if an application requires a certain performance guarantee then, for given  $p$  and  $l_c$ , the number of processors that should be used to ensure the required performance can be found.

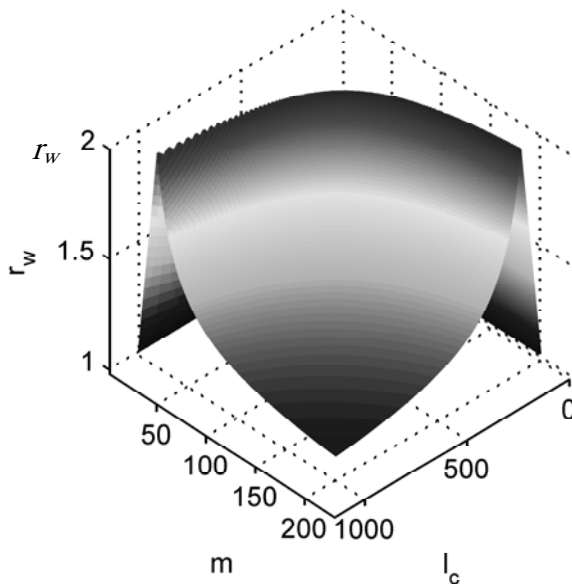


Fig. 2. Worst case ratio  $r_w$ , varying the number  $m$  of machines and the length  $l_c$  of the longest chain, keeping the total workload fixed ( $p = 20000$ ).

Figure 2 shows how the worst case bound depends on the length of the critical path when the number of machines  $m$  is varied. Decreasing the length of the critical path in the problem instance results in the slowdown of tending the bound to 1 with increasing the number of machines.

#### 4. Conclusion

This paper addressed the problem of dynamic non-preemptive list scheduling of  $P|prec|C_{max}$ . We have analyzed the worst case behavior of arbitrary list scheduling strategies and presented new performance guarantees under different assumptions. Additional knowledge of the total workload and the weighted length of the longest chain in the task system allow producing an accurate analysis of performance bounds.

#### References

- [1] T.L. Adam, K.M. Chandy and J.R. Dickson. A comparison of list schedules for parallel processing systems. *Comm. of the ACM*, 17:685-9, 1974.
- [2] J. Błażewicz, K. H. Ecker, E. Pesch, G. Schmidt, J. Węglarz, *Handbook on Scheduling – From Theory to Applications*, International Handbooks on Information Systems, Springer, Berlin, 2007.
- [3] J. Du, J. Y-T. Leung, Scheduling tree-structured tasks with restricted execution times, *Inform. Process. Lett.* 28, 1988, 183-188.
- [4] Du, J., Leung, J.Y-T., Complexity of scheduling parallel tasks systems, *SIAM J. Discrete Math.* 2, 1989, 473-487.
- [5] J. Du, J. Y-T. Leung, G. H. Young, Scheduling chain structured tasks to minimize makespan and mean flow time, *Inform. and Comput.* 92, 1991, 219-236.
- [6] A. Darte, Y. Robert, F. Vivien, *Scheduling and Automatic Parallelization*, Birkhäuser, 2000.
- [7] K.H. Ecker, Scheduling of resource tasks, *European Journal of Operational Research* 115 (1999) 314–327
- [8] R. L. Graham, E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling theory: a survey, *Ann. Discrete Math.* 5, 287-326, 1979.
- [9] R. L. Graham, Bounds for certain multiprocessing anomalies, *Bell System Tech. J.* 45, 1563-1581, 1966.
- [10] R. L. Graham, Bounds on multiprocessor timing anomalies, *SIAM J. Appl. Math.* 17, 263-269, 1969.
- [11] K. GODA, T. YAMADA, S. UENO. A Note on the Complexity of Scheduling for Precedence Constrained Messages in Distributed Systems. *IEICE Trans. Fundamentals*, Vol.E88-A, No.4, pp.1090-1092, 2005
- [12] Berit Johannes. Scheduling parallel jobs to minimize the makespan. *Journal of Scheduling*, Volume 9, Issue 5. Kluwer Academic Publishers, USA. Pages: 433 – 452, 2006
- [13] H. Kasahara and S. Narita, "A Practical Optimal / Approximate Algorithm for Multi-Processor Scheduling Problem", *Trans. of IEICE D*, Vol.67-D, No.7, Jul.1984
- [14] M. Kunde, Nonpreemptive LP-Scheduling on Homogeneous Multiprocessor Systems. *SIAM J. Comput.* 10 (1), 1981.
- [15] R. Li, H.-C. Huang. On-line Scheduling for Jobs with Arbitrary Release Times. *Computing*, Volume 73, Issue 1, Springer-Verlag, USA. pp. 79-97, 2004
- [16] G. Q. Liu, K. L. Poh, M. Xie. Iterative list scheduling for heterogeneous computing. *Journal of Parallel and Distributed Computing*, Volume 65, Issue 5, Academic Press, USA, Pages: 654 – 665. 2005
- [17] K. Nakajima, J. Y-T. Leung, S. L. Hakimi, Optimal two processor scheduling of tree precedence constrained tasks with two execution times, *Performance Evaluation* 1, 1981, 320-330.
- [18] A.Rodriguez, A.Tchernykh, K.Ecker. Algorithms for Dynamic Scheduling of Unit Execution Time Tasks. *European Journal of Operational Research*, Elsevier Science, North-Holland, v.146, 2, p. 403-416, 2003