# Average-Case Performance Analysis of Online Non-clairvoyant Scheduling of Parallel Tasks with Precedence Constraints

Keqin Li
Department of Computer Science
State University of New York
New Paltz, New York 12561, USA
lik@newpaltz.edu

## Abstract

*We evaluate the average-case performance of three approximation algorithms for online non-clairvoyant scheduling of parallel tasks with precedence constraints. We show that for a class of wide task graphs, when task sizes are uniformly distributed in the range $[1..C]$, the online non-clairvoyant scheduling algorithm LL-SIMPLE has an asymptotic average-case performance bound of*

$$M/(M-(3-(1+1/C)^{C+1})C-1),$$

*where $M$ is the number of processors. For arbitrary probability distributions of task sizes, we present numerical and simulation data to demonstrate the accuracy of our general asymptotic average-case performance bound. We also report extensive experimental results on the average-case performance of online non-clairvoyant scheduling algorithms LL-GREEDY and LS. Algorithm LL-GREEDY has better performance than LL-SIMPLE by using an improved algorithm to schedule independent tasks in the same level. Algorithm LS produces even better schedules due to break of boundaries among levels.*

## 1 Introduction

Assume that we are given a list of $N$ parallel tasks $L = (T_1, T_2, ..., T_N)$ with precedence constraints. Each task $T_i$ is specified by its execution time $\tau_i$ and its size $\pi_i$ (i.e., the number of processors requested by $T_i$). There are $M$ identical processors and any $\pi_i$ processors can be allocated to $T_i$. Among the $N$ tasks, there is a partial order $\prec$ which specifies the precedence constraints, that is, if $T_i \prec T_j$, then task $T_j$ cannot start its execution until task $T_i$ is finished. The problem addressed here is to find a non-preemptive schedule (i.e., once a task starts to execute, it runs without interruption until it completes) of $L$ such that its makespan (i.e., the total execution time of the $N$ tasks) is minimized. The problem can be regarded as a scheduling problem with resource constraints [2, 4] and precedence constraints, where the resource is a set of processors.

The problem is NP-hard, since it includes many classical NP-hard problems as special cases [5, 6, 9]. One practical and effective way to solve this parallel task scheduling problem is to design and analyze approximation algorithms that produce near-optimal solutions. Let $A(L)$ be the makespan of the schedule generated by an algorithm $A$ for $L$, and $\mathrm{OPT}(L)$ be the makespan of an optimal schedule of $L$. The *worst-case performance ratio* $R_A$ for algorithm $A$ is defined as

$$R_A = \inf\{r \geq 1 :\ A(L) \leq r\mathrm{OPT}(L) \text{ for all lists } L\}.$$

If there exists a constant $\alpha$ such that $A(L) \leq \alpha\mathrm{OPT}(L)$ for all $L$, then $R_A \leq \alpha$ and $\alpha$ is called a *worst-case performance bound* for algorithm $A$. When task sizes, execution times, and precedence constraints are random variables, both $A(L)$ and $\mathrm{OPT}(L)$ become random variables. The *average-case performance ratio* for algorithm $A$ is

$$\bar{R}_A^N = \inf\{r \geq 1 :\ \mathbb{E}(A(L)) \leq r\mathbb{E}(\mathrm{OPT}(L)) \text{ for all } L \text{ with } N \text{ tasks}\},$$

where $\mathbb{E}(\cdot)$ stands for the expectation of a random variable. The *asymptotic average-case performance ratio* for algorithm $A$ is $\bar{R}_A^\infty = \lim_{N\to\infty} \bar{R}_A^N$. Of course, $\bar{R}_A^\infty$ depends on the probability distributions of task sizes, execution times, and precedence constraints. If there exists a constant $\beta$ such that $\mathbb{E}(A(L)) \leq \beta\mathbb{E}(\mathrm{OPT}(L))$ for all $L$, as $N \to \infty$, then $\bar{R}_A^\infty \leq \beta$, and $\beta$ is called an *asymptotic average-case performance bound* for algorithm $A$.

In many applications, the execution time of a task is not available until the task is executed and completed. In this paper, we are interested in non-clairvoyant schedul-

1-4244-0910-1/07/$20.00 

ing of precedence constrained parallel tasks in parallel systems with identical processors, where it is assumed that the execution times of the tasks are not known a priori. A *non-clairvoyant* scheduling algorithm only knows the sizes $\pi_1, \pi_2, ..., \pi_N$ of the tasks and the precedence constraints among the tasks, but has no access to information about the execution times $\tau_1, \tau_2, ..., \tau_N$ of the tasks it is to schedule. The execution time of a task is known only when it is completed.

A list of $N$ precedence constrained parallel tasks can be represented by a task graph. In some applications, the $N$ tasks together constitute a single parallel computation. A task graph which represents a parallel computation is expanded gradually and dynamically during the course of the computation. An *online* scheduling algorithm is only given tasks which are ready for execution (that is, whose predecessors are all completed) and is not aware of their successors. For instance, a task graph can be divided into levels (see Section 2 for detailed discussion), and an online algorithm may schedule the tasks level by level (LL). When scheduling tasks in level $l$, an online algorithm does not know any information (sizes, execution times, and precedence constraints) of the tasks in levels $l+1$, $l+2$, ..., since these tasks are not generated or not ready yet.

Therefore, an online non-clairvoyant scheduling algorithm receives tasks at different times without prior knowledge of the future tasks and the execution times of the tasks that are not yet completed. However, the performance of an online non-clairvoyant scheduling algorithm is compared with an optimal offline clairvoyant scheduling algorithm which knows all the information of task sizes, execution times, and precedence constraints in advance.

It is still on open problem on whether there is an approximation algorithm $A$ with finite worst-case or average-case performance ratio for clairvoyant scheduling parallel tasks with precedence constraints. It has been proven in [3] that no non-clairvoyant scheduling algorithm, online or offline, has worst-case performance ratio less than $M$. However, when task sizes do not exceed $qM$, where $1/M \leq q \leq 1$, the simple online non-clairvoyant list scheduling (LS) algorithm can achieve a worst-case performance bound of

$$R_{\rm LS} \leq \frac{(2-q)M}{(1-q)M+1},$$

for non-clairvoyant scheduling of parallel tasks with precedence constraints [7]. The special case of online non-clairvoyant scheduling of precedence constrained parallel tasks with identical execution times was investigated in [1, 10].

In this paper, we analyze the average-case performance of algorithm LL-SIMPLE for online non-clairvoyant scheduling of parallel tasks with precedence constraints. Algorithm LL-SIMPLE is extended from algorithm SIMPLE designed for non-clairvoyant scheduling of independent parallel tasks [8]. Algorithm LL-SIMPLE schedules tasks level by level and schedules independent tasks in the same level by using algorithm SIMPLE. We show that for a class of wide task graphs, algorithm LL-SIMPLE has the same asymptotic average-case performance bound $M/P_M$ of algorithm SIMPLE, where $P_M$ (to be defined in Section 2) can be obtained easily for arbitrary probability distribution of the $\pi_j$'s by using a method developed in [8]. In particular, when task sizes are uniformly distributed in the range $[1..C]$, we have

$$\bar{R}^{\infty}_{\text{LL-SIMPLE}} \leq \frac{M}{M-(3-(1+1/C)^{C+1})C-1}.$$

The above asymptotic average-case performance bound achieves its maximum value when $C = M$, which is approximately $1/(e-2) = 1.3922112$ for large $M$. We present numerical and simulation data to demonstrate the accuracy of the bound $M/P_M$.

We also report extensive experimental results on the average-case performance of online non-clairvoyant scheduling algorithms LL-GREEDY and LS. Algorithm LL-GREEDY has better performance than LL-SIMPLE by using an improved algorithm GREEDY [8] to schedule independent tasks in the same level. Algorithm LS produces even better schedules due to break of boundaries among levels. Although the online non-clairvoyant scheduling algorithms LL-GREEDY and LS cannot achieve finite worst-case performance ratios, algorithm LL-GREEDY can achieve an average-case performance ratio very close to optimal for large wide task graphs, and algorithm LS can achieve an average-case performance ratio very close to optimal even for small to moderate wide task graphs.

## 2 Algorithm LL-SIMPLE

A list $L$ of parallel tasks with precedence constraints $\prec$ can be represented by a task graph which is a directed acyclic graph (dag) $(L, \prec)$, where each task $T_j$ is represented by a node and there is an arc $(T_i, T_j)$ in the task graph if $T_i \prec T_j$. A task graph can be decomposed into levels $L_1, L_2, ..., L_v$, where $v$ is the number of levels and each level $L_l$ is a sublist of $L$, $1 \leq l \leq v$. Tasks with no predecessors (called initial tasks) constitute the first level $L_1$. Generally, a task $T_j$ is in level $L_l$ if the number of nodes on the longest path from some initial task to $T_j$ is $l$. Let $N_l = |L_l|$ be the number of tasks in $L_l$, where $1 \leq l \leq v$. Note that all tasks in the same level are independent of each other, and hence, they can be scheduled by any algorithm for scheduling independent parallel tasks.

The online non-clairvoyant scheduling algorithm LL-SIMPLE schedules tasks in $L$ level by level in the order $L_1, L_2, ..., L_v$. Tasks in $L_{l+1}$ cannot start their execution

until all tasks in $L_l$ are completed. For level $L_l$, we use algorithm SIMPLE to generate its schedule.

Algorithm SIMPLE for scheduling a list $L = (T_1, T_2, ..., T_N)$ of independent parallel tasks works as follows. Let the $N$ tasks be executed in the time interval $[0, \text{SIMPLE}(L)]$. The $N$ tasks $T_1, T_2, ..., T_N$ are divided into $(k+1)$ groups, $G_b = \{T_{i_{b-1}+1}, T_{i_{b-1}+2}, ..., T_{i_b}\}$, $1 \leq b \leq k+1$, where $i_0 = 0$, $i_{k+1} = N$, and the value $k$ and the indices $i_1, i_2, ..., i_k$ are to be defined below. Let $T_{j_1}, T_{j_2}$, ..., $T_{j_k}$, ... be the sequence of tasks finished in a schedule produced by algorithm SIMPLE, where $T_{j_b}$ is completed at time $s_b$, $1 \leq b \leq k$, and at that time tasks in $G_{b+1}$ are scheduled for execution together with those tasks not completed yet. Hence, the time interval $[0, \text{SIMPLE}(L)]$ is divided into $(k+1)$ subintervals $[s_0, s_1)$, $[s_1, s_2)$, $[s_2, s_3)$, ..., $[s_{k-1}, s_k)$, $[s_k, s_{k+1}]$, where $s_0 = 0$ and $s_{k+1} = \text{SIMPLE}(L)$. For $0 \leq b \leq k$, the set of active tasks running during time subinterval $[s_b, s_{b+1}]$ is $G_1 \cup G_2 \cup \cdots \cup G_{b+1} - \{T_{j_1}, T_{j_2}, ..., T_{j_b}\}$.

Initially, at time 0, tasks in $G_1$ are scheduled for execution, where $i_1$ is defined such that the total number of processors used by the $N_1 = i_1$ tasks executed during the time subinterval $[s_0, s_1]$ is $p_1 = \pi_1 + \pi_2 + \cdots + \pi_{i_1} \leq M$, but $p_1 + \pi_{i_1+1} = \pi_1 + \pi_2 + \cdots + \pi_{i_1} + \pi_{i_1+1} > M$. In other words, we schedule as many tasks as possible for simultaneous execution. In general, for $2 \leq b \leq k$, when task $T_{j_{b-1}}$ is completed, tasks in $G_b$ are scheduled for execution, where $i_b$ is defined such that the total number of processors used by the $N_b = N_{b-1} - 1 + i_b - i_{b-1}$ tasks executed during the time subinterval $[s_{b-1}, s_b]$ is $p_b = p_{b-1} - \pi_{j_{b-1}} + \pi_{i_{b-1}+1} + \pi_{i_{b-1}+2} + \cdots + \pi_{i_b} \leq M$, but $p_b + \pi_{i_b+1} = p_{b-1} - \pi_{j_{b-1}} + \pi_{i_{b-1}+1} + \pi_{i_{b-1}+2} + \cdots + \pi_{i_b} + \pi_{i_b+1} > M$. Again, algorithm SIMPLE schedules as many tasks as possible for simultaneous execution. Finally, when task $T_{j_k}$ is completed, tasks in $G_{k+1}$ are scheduled for execution. Hence, $k$ is the smallest value such that when $T_{j_k}$ is completed, all the remaining tasks in $L$ can be scheduled and there may be room to accommodate more tasks. The number of tasks scheduled for execution at time $s_k$ is $N_{k+1} = N_k - 1 + N - i_k$, and the number of processors used at time $s_k$ is $p_{k+1} = p_k - \pi_{j_k} + \pi_{i_k+1} + \pi_{i_k+2} + \cdots + \pi_N$. Note that the number of processors used during the time subinterval $[s_k, \text{SIMPLE}(L)]$ decreases as more and more tasks are finished but there is no more task to be scheduled.

As a special case, there might be several tasks completed at the same time, say, $T_{j_b}, T_{j_{b+1}}, ..., T_{j_{b+r}}$ are finished simultaneously, where $r \geq 1$. According to the description of SIMPLE, we have $r$ null time subintervals $[s_b, s_{b+1})$, $[s_{b+1}, s_{b+2})$, ..., $[s_{b+r-1}, s_{b+r})$, where $s_b = s_{b+1} = s_{b+2} = \cdots = s_{b+r}$.

Let $c_j = \pi_j \tau_j$ be the cost of task $T_j$, where $1 \leq j \leq N$, and

$$C(L) = \sum_{j=1}^{N} c_j = \sum_{j=1}^{N} \pi_j \tau_j$$

be the total cost of the $N$ tasks. Let $t_b = s_b - s_{b-1}$ be the length of the time subinterval $[s_{b-1}, s_b]$, where $1 \leq b \leq k+1$. Since there are $p_b$ processors used during the time subinterval $[s_{b-1}, s_b]$, where $1 \leq b \leq k$, we have

$$C(L) \geq \sum_{b=1}^{k} p_b t_b,$$

by ignoring the cost during $[s_k, \text{SIMPLE}(L)]$.

To conduct probabilistic analysis of the average-case performance of algorithms SIMPLE and LL-SIMPLE, we assume that $\pi_1$, $\pi_2$, ..., $\pi_N$ are i.i.d. discrete random variables with a common probability distribution in the range $[1..M]$, and $\tau_1$, $\tau_2$, ..., $\tau_N$ are i.i.d. continuous random variables. The probability distribution of the $\pi_j$'s is independent of the probability distribution of the $\tau_j$'s. Let $\bar{\pi}$ be the expected task size and $\bar{\tau}$ be the expected task execution time.

It is clear that $N_b$, $p_b$, and $t_b$ are all random variables, and we have

$$\mathbb{E}(C(L)) \geq \sum_{b=1}^{k} \mathbb{E}(p_b t_b).$$

Note that $t_b$ is the minimum remaining execution time of the $N_b$ tasks scheduled at time $s_{b-1}$. Furthermore, the $N_b$'s, $p_b$'s, and $t_b$'s are all correlated random variables. This makes the evaluation of $\mathbb{E}(p_b t_b)$ very difficult. To make our average-case analysis feasible, we make the following assumption.

**Approximation 1.** $p_b$ and $t_b$ are independent of each other for all $1 \leq b \leq k$.

By using the above approximation, we get

$$\mathbb{E}(C(L)) \geq \sum_{b=1}^{k} \mathbb{E}(p_b)\mathbb{E}(t_b).$$

To further evaluate $\mathbb{E}(p_b)$, let us consider the following model. Assume that there is a bag with capacity $M$ and $N$ objects of sizes $\pi_1$, $\pi_2$, ..., $\pi_N$ which are i.i.d. (not necessarily discrete) random variables with a common probability distribution. Objects are packed into the bag as many as possible, i.e., we find index $i$ such that $\pi_1 + \pi_2 + \cdots + \pi_i \leq M$, but $\pi_1 + \pi_2 + \cdots + \pi_i + \pi_{i+1} > M$. The total size of the objects packed into the bag is a random variable. We use $P(N, M)$ to represent its mean. It is not hard to see that at times $s_0$, $s_1$, $s_2$, ..., $s_k$, tasks are scheduled exactly in the same way as objects are packed. In particular, we have $\mathbb{E}(p_b) = P(N - b + 1, M)$, where $1 \leq b \leq k+1$.

The above discussion gives rise to

$$\mathbb{E}(C(L)) \geq \sum_{b=1}^{k} P(N - b + 1, M)\mathbb{E}(t_b).$$

It is clear $P(N, M)$ converges rapidly to its limit $P_M = \lim_{N\to\infty} P(N, M)$. In fact, if the $\pi_j$'s are discrete integer random variables, we have $P_M = P(N, M)$ for all $N \geq M$, since the bag can accommodate at most $M$ objects. When $N$ is large, we make the following assumption.

**Approximation 2.** $P(N-b+1, M) \approx P_M$ for all $1 \leq b \leq k$.

The above approximation yields

$$\mathbb{E}(C(L)) \geq P_M \sum_{b=1}^{k} \mathbb{E}(t_b).$$

Since

$$\text{SIMPLE}(L) = \sum_{b=1}^{k} t_b + t_{k+1},$$

we have

$$\begin{aligned} \mathbb{E}(\text{SIMPLE}(L)) &= \sum_{b=1}^{k} \mathbb{E}(t_b) + \mathbb{E}(t_{k+1}) \\ &\leq \frac{\mathbb{E}(C(L))}{P_M} + \mathbb{E}(t_{k+1}). \end{aligned}$$

We notice that $t_{k+1}$ is a very sophisticated random variable, which is the maximum execution time of $N_{k+1}$ tasks. Among these tasks, $N_k - 1$ of them are partially executed and $N - i_k$ of them are just scheduled for execution, where $N_k$ and $i_k$ themselves are mysterious random variables. For ease of analysis, we make the following assumption which does not affect our asymptotic analysis.

**Approximation 3.** $\mathbb{E}(t_{k+1}) \approx \bar{\tau}$.

It is clear that the level-by-level schedule produced by algorithm LL-SIMPLE yields

$$\mathbb{E}(\text{LL-SIMPLE}(L)) = \sum_{l=1}^{v} \mathbb{E}(\text{SIMPLE}(L_l)).$$

From the above discussion, we know that for all $1 \leq l \leq v$,

$$\mathbb{E}(\text{SIMPLE}(L_l)) \leq \frac{\mathbb{E}(C(L_l))}{P_M} + \bar{\tau}.$$

The above equation implies that

$$\mathbb{E}(\text{LL-SIMPLE}(L)) \leq \frac{\mathbb{E}(C(L))}{P_M} + v\bar{\tau}.$$

Since

$$\text{OPT}(L) \geq \frac{C(L)}{M},$$

we obtain

$$\mathbb{E}(\text{LL-SIMPLE}(L)) \leq \frac{M}{P_M}\mathbb{E}(\text{OPT}(L)) + v\bar{\tau}.$$

Because

$$\mathbb{E}(\text{OPT}(L)) \geq \frac{\mathbb{E}(C(L))}{M},$$

and $\mathbb{E}(C(L)) = N\bar{\pi}\bar{\tau}$, we get

$$\frac{\mathbb{E}(\text{LL-SIMPLE}(L))}{\mathbb{E}(\text{OPT}(L))} \leq \frac{M}{P_M} + \left(\frac{M}{\bar{\pi}}\right)\left(\frac{v}{N}\right).$$

A class of task graphs are called *wide task graphs* if $v/N \to 0$ as $N \to \infty$. For fixed $M$ and $\bar{\pi}$, we have

$$\bar{R}^{\infty}_{\text{LL-SIMPLE}} = \lim_{N\to\infty} \frac{\mathbb{E}(\text{LL-SIMPLE}(L))}{\mathbb{E}(\text{OPT}(L))} \leq \frac{M}{P_M},$$

for wide task graphs. Though the above asymptotic average-case performance bound $M/P_M$ for algorithm LL-SIMPLE is derived by using Approximations 1–3, we will later justify that the bound is highly accurate.

In [8], a recurrence relation was found to calculate $P(N, M)$, the expectation of the total size of the objects (taken from $N$ objects of sizes $\pi_1$, $\pi_2$, ..., $\pi_N$) packed into a bag of capacity $M$, for arbitrary probability distribution of object sizes. Let $r_m$ be the probability that the size of an object is $\pi_j = m$, where $m = 1, 2, 3, ....$ The following recurrence relation characterizes $P(N, M)$:

$$P(N, M) = \begin{cases} \sum_{m=1}^{M} m r_m, & \text{if } N = 1; \\ \sum_{m=1}^{M} r_m(m + P(N-1, M-m)), & \text{if } N > 1. \end{cases}$$

Furthermore, when object sizes have a uniform distribution in the range $[1..C]$, i.e., $r_m = 1/C$ for all $1 \leq m \leq C$, a closed form approximation for $P_M = \lim_{N\to\infty} P(N, M)$ was derived, namely,

$$P_M \approx M - \left(3 - \left(1 + \frac{1}{C}\right)^{C+1}\right)C - 1.$$

We conclude that when task sizes $\pi_1$, $\pi_2$, ..., $\pi_N$ are i.i.d. discrete random variables with a uniform probability distribution in the range $[1..C]$, we get

$$\bar{R}^{\infty}_{\text{LL-SIMPLE}} \leq \frac{M}{M - (3 - (1 + 1/C)^{C+1})C - 1}.$$

When $C$ is large, we have $(1 + 1/C)^C \approx e$, and $P_M \approx M - (3 - e)C + (e - 1)$, and

$$\bar{R}^{\infty}_{\text{LL-SIMPLE}} \leq \frac{M}{M - (3 - e)C + (e - 1)}.$$

This implies that for large $M$, the asymptotic average-case performance bound gets its maximum value approximately $1/(e - 2) = 1.3922112$ when $C = M$.

## 3 Algorithms LL-GREEDY and LS

The online non-clairvoyant scheduling algorithm LL-GREEDY schedules tasks in $L$ level by level in the order $L_1, L_2, ..., L_v$. Tasks in $L_{l+1}$ cannot start their execution until all tasks in $L_l$ are completed. For level $L_l$, we use algorithm GREEDY to generate its schedule [8].

Algorithm GREEDY is an improved version of SIMPLE which can be described in a way similar to that of SIMPLE. Let $T_{j_1}$, $T_{j_2}$, ..., $T_{j_k}$ be the sequence of tasks finished in a schedule produced by algorithm GREEDY. The difference is that at time $s_b$, i.e., when task $T_{j_b}$ is completed, where $1 \leq b \leq k$, each of the remaining tasks not scheduled yet is tested to see whether it can be scheduled for execution, i.e., whether there are enough processors for the task. Thus, $G_{b+1}$ in GREEDY may include more tasks than that of SIMPLE.

It is clear that algorithm GREEDY improves processor utilization. In terms of our object packing model, all the $N$ objects are considered for possible packing into the bag and an object is packed if there is enough room. The expectation $P^*(N, M)$ of the total size of the objects packed into a bag is given by the following recurrence relation:

$$P^*(N,M) = \begin{cases} \sum_{m=1}^{M} m r_m, & \text{if } N = 1; \\ \sum_{m=1}^{M} r_m(m + P^*(N-1, M-m)) & \\ \quad + \left(\sum_{m>M} r_m\right) P^*(N-1, M), & \text{if } N > 1. \end{cases}$$

When $N$ is large, the chance to fully pack the bag is high. Let $P^*_M = \lim_{N\to\infty} P^*(N, M)$. If $r_1 \neq 0$, we have $P^*_1 = 1$. One can then prove by induction on $M$ that $P^*_M = M$ for all $M \geq 1$. This implies that all the $M$ processors are used, especially in the beginning of the schedule.

The well known online non-clairvoyant list scheduling (LS) algorithm further improves performance by breaking the boundaries among levels. The list scheduling algorithm was originally proposed in [5] for scheduling sequential tasks that demand for only one processor, i.e., $\pi_i = 1$, for all $1 \leq i \leq N$. A list schedule is based on an initial ordering of the tasks $L = (T_{j_1}, T_{j_2}, ..., T_{j_n})$, called a priority list. Initially, at time zero, the scheduler instantaneously scans list $L$ from the beginning, searching for tasks that are ready to be executed, i.e., which have no predecessors under $\prec$ still waiting in $L$. The first ready task $T_{j_i}$ in $L$ is removed from $L$ and sent to an idle processor for processing. Such a search is repeated until there is no ready task or there is no more processor available. In general, whenever a processor completes a task, the scheduler immediately scans $L$, looking for the first ready task to be executed. If such a ready task is not found, the processor becomes idle and waits for the next finished task. As running tasks complete, more precedence constraints are removed and more tasks will be ready.

The extension of the method to scheduling parallel tasks is straightforward [7]. When the scheduler finds a ready task $T_{j_i}$, the scheduler checks whether there are at least $\pi_{j_i}$ idle processors. If so, task $T_{j_i}$ is allocated $\pi_{j_i}$ processors and executed nonpreemptively on these processors. Otherwise, the ready task $T_{j_i}$ still needs to wait in $L$ until other running tasks complete. Therefore, initially and later whenever a task is completed, each of the remaining tasks not scheduled yet is examined to see whether it can be scheduled for execution, i.e., whether all its predecessors are completed and there are enough processors for the task.

In the above description, algorithm LS is allowed to scan the entire list of tasks to be consistent with Graham's original algorithm. It should be noticed that during each scan, tasks not ready for execution are simply skipped without further examination of their processor requirements. Therefore, when a list scheduler is only provided with ready tasks for online scheduling, it generates exactly the same schedule, since the LS algorithm does not need any information of tasks not ready for execution.

## 4 Simulation Data

Extensive simulations have been conducted to validate the asymptotic average-case performance bound for algorithm LL-SIMPLE and to demonstrate the average-case performance of the three approximation algorithms for non-clairvoyant scheduling of parallel tasks with precedence constraints. The following task graphs are considered in our experiments.

- *Tree-Structured Computations.* For simplicity, we consider CT$(b, h)$, i.e., complete $b$-ary trees of height $h$. There are $v = h + 1$ levels numbered as 0, 1, 2, ..., $h$, and $N_l = b^l$ for $0 \leq l \leq h$, and $N = (b^{h+1} - 1)/(b - 1)$.

- *Partitioning Algorithms.* A partitioning algorithm PA$(b, h)$ has $v = 2h+1$ levels numbered as 0, 1, 2, ..., $2h$, where $N_l = N_{2h-l} = b^l$, for all $0 \leq l \leq h - 1$, $N_h = b^h$, and $N = (b^{h+1} + b^h - 2)/(b - 1)$.

- *Linear Algebra Task Graphs.* A linear algebra task graph LA$(v)$ with $v$ levels has $N_l = v - l + 1$ for all $1 \leq l \leq v$, and $N = v(v + 1)/2$.

- *Diamond Dags.* A diamond dag DD$(d)$ contains $v = 2d - 1$ levels numbered as 1, 2, ..., $2d - 1$, where $N_l = N_{2d-l} = l$, for all $1 \leq l \leq d-1$, $N_d = d$, and $N = d^2$.

Since each task graph has at least one parameter, we are actually dealing with classes of task graphs. It is easily verified that all the four classes of task graphs are wide task graphs.

Now, we present numerical and simulation data for the above task graphs. We make the following parameter setting in our simulations.

- The number of processors is $M = 128$. (The choice of $M$ does not affect our observations and conclusions.)
- The probability distribution of task sizes is a uniform distributions in the range $[1..C]$, i.e., $r_m = 1/C$ for all $1 \leq m \leq C$, where $C$ is chosen such that $(C+1)/2 = \bar{\pi}$, i.e., $C = 2\bar{\pi} - 1$.
- Task execution times are uniformly distributed in the interval $[0, 10]$ with $\bar{\tau} = 5$.
- For complete trees and partitioning algorithms, we set $b = 2$.

Tables 1–4 display our numerical and simulation data for the five types of task graphs. The asymptotic average-case performance bound $M/P_M$ for algorithm LL-SIMPLE is given as the last line of Tables 1a–4a, where $P_M$ is calculated using the recurrence relation in Section 2 with $N = M$. Each simulation datum is the average of $rep$ (given in each table) schedule lengths generated by algorithm $A$, $A \in \{\text{LL-SIMPLE}, \text{LL-GREEDY}, \text{LS}\}$, divided by a lower bound for $\mathbb{E}(\text{OPT}(L))$, namely, $\mathbb{E}(C(L))/M$, where $\mathbb{E}(C(L))$ is the mean cost $\mathbb{E}(C(L)) = N\bar{\pi}\bar{\tau}$, The 99% confidence interval of our simulation data is given in each table.

It is observed from Tables 1a–4a that as $N$ increases, the performance bound obtained from simulation improves and approaches the analytical bound. When $N$ is large, the performance bound obtained from simulation is slightly larger than but still very close to the analytical bound, primarily due to the under-estimation of $\mathbb{E}(\text{OPT}(L))$ and loss of efficiency caused by boundaries among levels. These simulation data justify the high accuracy of our analytical asymptotic average-case performance bound $M/P_M$ for algorithm LL-SIMPLE in scheduling precedence constrained parallel tasks.

It is also observed from Tables 1b–4b and 1c–4c that algorithm LL-GREEDY has better performance than LL-SIMPLE by using an improved algorithm GREEDY to schedule independent tasks in the same level. Algorithm LS produces even better schedules than algorithm LL-GREEDY due to break of boundaries among levels. Algorithm LL-GREEDY can achieve an average-case performance ratio very close to optimal for large wide task graphs. Algorithm LS can achieve an average-case performance ratio very close to optimal even for small to moderate wide task graphs.

# 5 Conclusions

We have analyzed the average-case performance of algorithm LL-SIMPLE for online non-clairvoyant scheduling of parallel tasks with precedence constraints. Our extensive numerical and simulation data demonstrate that the analytical asymptotic average-case performance bound for algorithm LL-SIMPLE is highly accurate and that algorithms LL-GREEDY and LS can produce high quality schedules, especially for large wide task graphs.

# References


[1] S. Bischof and E. W. Mayr, "On-line scheduling of parallel jobs with runtime restrictions," *Theoretical Computer Science* **268**, 67-90, 2001.

[2] J. Blazewicz, J. K. Lenstra, and A. H. G. Rinnooy Kan, "Scheduling subject to resource constraints: classification and complexity," *Discrete Applied Mathematics* **5**, 11-24, 1983.

[3] A. Feldmann, M.-Y. Kao, J. Sgall, and S.-H. Teng, "Optimal on-line scheduling of parallel jobs with dependencies," *Journal of Combinatorial Optimization* **1**, 393-411, 1998.

[4] M. R. Garey and R. L. Graham, "Bound for multiprocessor scheduling with resource constraints," *SIAM Journal on Computing* **4**, 187-200, 1975.

[5] R. L. Graham, "Bounds on multiprocessing timing anomalies," *SIAM J. Appl. Math.* **2**, 416-429, 1969.

[6] D. S. Johnson, *et al.*, "Worst-case performance bounds for simple one-dimensional packing algorithms," *SIAM Journal on Computing* **3**, 299-325, 1974.

[7] K. Li, "Analysis of the list scheduling algorithm for precedence constrained parallel tasks," *Journal of Combinatorial Optimization* **3**, 73-88, 1999.

[8] K. Li, "An average-case analysis of online non-clairvoyant scheduling of independent parallel tasks," *Journal of Parallel and Distributed Computing* **66**, 617-625, 2006.

[9] J. D. Ullman, "NP-complete scheduling problems," *Journal of Computer and System Science* **10**, 384-393, 1975.

[10] D. Ye and G. Zhang, "Online scheduling of parallel jobs with dependencies on 2-dimensional meshes," *Lecture Notes in Computer Science*, vol. 2906, 329-338, 2003.

Table 1a: Simulation Data for LL-SIMPLE on Complete Trees.
($rep = 200$, 99% confidence interval $= \pm 1.43271\%$)

| $h$ | $\bar{\pi} = 5$ | $\bar{\pi} = 10$ | $\bar{\pi} = 15$ | $\bar{\pi} = 20$ | $\bar{\pi} = 25$ | $\bar{\pi} = 30$ |
|---|---|---|---|---|---|---|
| 6 | 2.68375 | 1.76032 | 1.49632 | 1.39524 | 1.34164 | 1.34402 |
| 7 | 1.94714 | 1.44954 | 1.31151 | 1.26233 | 1.25059 | 1.26836 |
| 8 | 1.53624 | 1.26962 | 1.20721 | 1.20423 | 1.21151 | 1.23381 |
| 9 | 1.30376 | 1.17122 | 1.14915 | 1.16408 | 1.18362 | 1.21508 |
| 10 | 1.17545 | 1.11466 | 1.11734 | 1.13768 | 1.16711 | 1.20497 |
| 11 | 1.10276 | 1.08517 | 1.10216 | 1.12820 | 1.15897 | 1.19647 |
| 12 | 1.06623 | 1.06960 | 1.09198 | 1.12208 | 1.15446 | 1.19298 |
| 13 | 1.04599 | 1.06057 | 1.08636 | 1.11782 | 1.15287 | 1.19130 |
| | 1.02128 | 1.04918 | 1.07865 | 1.10992 | 1.14277 | 1.17845 |

Table 2a: Simulation Data for LL-SIMPLE on Partitioning Algorithms.
($rep = 200$, 99% confidence interval $= \pm 1.23513\%$)

| $h$ | $\bar{\pi} = 5$ | $\bar{\pi} = 10$ | $\bar{\pi} = 15$ | $\bar{\pi} = 20$ | $\bar{\pi} = 25$ | $\bar{\pi} = 30$ |
|---|---|---|---|---|---|---|
| 6 | 3.11308 | 1.94281 | 1.60902 | 1.45977 | 1.39894 | 1.35943 |
| 7 | 2.18663 | 1.54686 | 1.38394 | 1.31629 | 1.29715 | 1.29296 |
| 8 | 1.67087 | 1.33092 | 1.24932 | 1.22782 | 1.22710 | 1.24541 |
| 9 | 1.38301 | 1.20482 | 1.17145 | 1.17496 | 1.19399 | 1.22306 |
| 10 | 1.21776 | 1.13485 | 1.12906 | 1.14629 | 1.17302 | 1.20644 |
| 11 | 1.12904 | 1.09574 | 1.10744 | 1.13152 | 1.16328 | 1.19956 |
| 12 | 1.07894 | 1.07517 | 1.09407 | 1.12359 | 1.15602 | 1.19416 |
| 13 | 1.05191 | 1.06344 | 1.08801 | 1.11941 | 1.15331 | 1.19266 |
| | 1.02128 | 1.04918 | 1.07865 | 1.10992 | 1.14277 | 1.17845 |

Table 1b: Simulation Data for LL-GREEDY on Complete Trees.
($rep = 200$, 99% confidence interval $= \pm 1.62384\%$)

| $h$ | $\bar{\pi} = 5$ | $\bar{\pi} = 10$ | $\bar{\pi} = 15$ | $\bar{\pi} = 20$ | $\bar{\pi} = 25$ | $\bar{\pi} = 30$ |
|---|---|---|---|---|---|---|
| 6 | 2.66068 | 1.72388 | 1.45190 | 1.33743 | 1.27057 | 1.24246 |
| 7 | 1.92790 | 1.40719 | 1.25836 | 1.20189 | 1.16702 | 1.17122 |
| 8 | 1.51620 | 1.23138 | 1.15159 | 1.12388 | 1.10738 | 1.12594 |
| 9 | 1.27892 | 1.13181 | 1.08691 | 1.07729 | 1.06527 | 1.09217 |
| 10 | 1.15408 | 1.07292 | 1.05294 | 1.05010 | 1.04561 | 1.06739 |
| 11 | 1.08348 | 1.04181 | 1.03064 | 1.03130 | 1.03069 | 1.05434 |
| 12 | 1.04505 | 1.02448 | 1.01754 | 1.01993 | 1.01842 | 1.04205 |
| 13 | 1.02358 | 1.01347 | 1.01029 | 1.01191 | 1.01223 | 1.03294 |

Table 2b: Simulation Data for LL-GREEDY on Partitioning Algorithms.
($rep = 200$, 99% confidence interval $= \pm 1.08233\%$)

| $h$ | $\bar{\pi} = 5$ | $\bar{\pi} = 10$ | $\bar{\pi} = 15$ | $\bar{\pi} = 20$ | $\bar{\pi} = 25$ | $\bar{\pi} = 30$ |
|---|---|---|---|---|---|---|
| 6 | 3.08224 | 1.90332 | 1.55633 | 1.39734 | 1.32375 | 1.28893 |
| 7 | 2.16210 | 1.50806 | 1.31779 | 1.24008 | 1.20405 | 1.19899 |
| 8 | 1.64784 | 1.28733 | 1.18547 | 1.14542 | 1.12633 | 1.13730 |
| 9 | 1.35868 | 1.16469 | 1.10952 | 1.09143 | 1.08237 | 1.09911 |
| 10 | 1.19490 | 1.09042 | 1.06378 | 1.05777 | 1.05463 | 1.07593 |
| 11 | 1.10646 | 1.05234 | 1.03639 | 1.03534 | 1.03438 | 1.05933 |
| 12 | 1.05859 | 1.02850 | 1.02185 | 1.02329 | 1.02193 | 1.04664 |
| 13 | 1.03133 | 1.01601 | 1.01365 | 1.01464 | 1.01455 | 1.03627 |

Table 1c: Simulation Data for LS on Complete Trees.
($rep = 200$, 99% confidence interval $= \pm 1.70777\%$)

| $h$ | $\bar{\pi} = 5$ | $\bar{\pi} = 10$ | $\bar{\pi} = 15$ | $\bar{\pi} = 20$ | $\bar{\pi} = 25$ | $\bar{\pi} = 30$ |
|---|---|---|---|---|---|---|
| 6 | 2.02364 | 1.36391 | 1.22982 | 1.16528 | 1.13485 | 1.14323 |
| 7 | 1.42756 | 1.17242 | 1.12064 | 1.08516 | 1.09269 | 1.09260 |
| 8 | 1.21703 | 1.08882 | 1.06126 | 1.05738 | 1.05292 | 1.06660 |
| 9 | 1.11257 | 1.04877 | 1.03660 | 1.03544 | 1.03366 | 1.04632 |
| 10 | 1.05627 | 1.02581 | 1.02287 | 1.02017 | 1.02301 | 1.03194 |
| 11 | 1.02969 | 1.01288 | 1.01041 | 1.01337 | 1.01274 | 1.02162 |
| 12 | 1.01556 | 1.00869 | 1.00565 | 1.00798 | 1.00986 | 1.01429 |
| 13 | 1.00807 | 1.00474 | 1.00339 | 1.00501 | 1.00584 | 1.00914 |

Table 2c: Simulation Data for LS on Partitioning Algorithms.
($rep = 200$, 99% confidence interval $= \pm 1.41154\%$)

| $h$ | $\bar{\pi} = 5$ | $\bar{\pi} = 10$ | $\bar{\pi} = 15$ | $\bar{\pi} = 20$ | $\bar{\pi} = 25$ | $\bar{\pi} = 30$ |
|---|---|---|---|---|---|---|
| 6 | 2.29793 | 1.47990 | 1.28494 | 1.19601 | 1.17318 | 1.15455 |
| 7 | 1.62619 | 1.24501 | 1.14791 | 1.10882 | 1.09633 | 1.09496 |
| 8 | 1.31601 | 1.12894 | 1.07991 | 1.06029 | 1.05345 | 1.05578 |
| 9 | 1.15920 | 1.06567 | 1.04489 | 1.03647 | 1.03590 | 1.04064 |
| 10 | 1.08141 | 1.03214 | 1.02496 | 1.02119 | 1.02153 | 1.02735 |
| 11 | 1.04015 | 1.01798 | 1.01276 | 1.01495 | 1.01393 | 1.01716 |
| 12 | 1.01923 | 1.00913 | 1.00754 | 1.00760 | 1.00906 | 1.01161 |
| 13 | 1.01003 | 1.00471 | 1.00453 | 1.00492 | 1.00539 | 1.00760 |

Table 3a: Simulation Data for LL-SIMPLE on Linear Algebra Task Graphs.
($rep = 50$, 99% confidence interval $= \pm 1.00533\%$)

| $v$ | $\bar{\pi} = 5$ | $\bar{\pi} = 10$ | $\bar{\pi} = 15$ | $\bar{\pi} = 20$ | $\bar{\pi} = 25$ | $\bar{\pi} = 30$ |
|---|---|---|---|---|---|---|
| 50 | 2.13478 | 1.52077 | 1.35943 | 1.30048 | 1.28257 | 1.28829 |
| 100 | 1.55604 | 1.28031 | 1.21590 | 1.20584 | 1.21323 | 1.23449 |
| 150 | 1.37183 | 1.20130 | 1.16763 | 1.17270 | 1.19043 | 1.22071 |
| 200 | 1.28188 | 1.16370 | 1.14732 | 1.15934 | 1.18263 | 1.21476 |
| 250 | 1.23063 | 1.13971 | 1.13534 | 1.15010 | 1.17524 | 1.20823 |
| 300 | 1.19498 | 1.12555 | 1.12383 | 1.14141 | 1.17043 | 1.20597 |
| 350 | 1.17041 | 1.11302 | 1.11777 | 1.13883 | 1.16782 | 1.20273 |
| 400 | 1.15110 | 1.10650 | 1.11434 | 1.13621 | 1.16509 | 1.20084 |
|  | 1.02128 | 1.04918 | 1.07865 | 1.10992 | 1.14277 | 1.17845 |

Table 4a: Simulation Data for LL-SIMPLE on Diamond Dags.
($rep = 50$, 99% confidence interval $= \pm 0.68027\%$)

| $d$ | $\bar{\pi} = 5$ | $\bar{\pi} = 10$ | $\bar{\pi} = 15$ | $\bar{\pi} = 20$ | $\bar{\pi} = 25$ | $\bar{\pi} = 30$ |
|---|---|---|---|---|---|---|
| 50 | 2.13710 | 1.51999 | 1.35453 | 1.30154 | 1.28078 | 1.28755 |
| 100 | 1.55982 | 1.28053 | 1.21408 | 1.20307 | 1.21413 | 1.23973 |
| 150 | 1.37236 | 1.20095 | 1.17058 | 1.17331 | 1.19362 | 1.22067 |
| 200 | 1.28410 | 1.16172 | 1.14600 | 1.15864 | 1.18195 | 1.21276 |
| 250 | 1.23094 | 1.13992 | 1.13251 | 1.14942 | 1.17503 | 1.20860 |
| 300 | 1.19522 | 1.12629 | 1.12450 | 1.14297 | 1.17132 | 1.20552 |
| 350 | 1.17054 | 1.11486 | 1.11856 | 1.13996 | 1.16600 | 1.20335 |
| 400 | 1.15108 | 1.10637 | 1.11326 | 1.13626 | 1.16606 | 1.20142 |
|  | 1.02128 | 1.04918 | 1.07865 | 1.10992 | 1.14277 | 1.17845 |

Table 3b: Simulation Data for LL-GREEDY on Linear Algebra Task Graphs.
($rep = 50$, 99% confidence interval $= \pm 0.95177\%$)

| $v$ | $\bar{\pi} = 5$ | $\bar{\pi} = 10$ | $\bar{\pi} = 15$ | $\bar{\pi} = 20$ | $\bar{\pi} = 25$ | $\bar{\pi} = 30$ |
|---|---|---|---|---|---|---|
| 50 | 2.08580 | 1.45961 | 1.29074 | 1.21872 | 1.19388 | 1.19373 |
| 100 | 1.52154 | 1.23555 | 1.16070 | 1.13469 | 1.12382 | 1.13674 |
| 150 | 1.34401 | 1.15788 | 1.11015 | 1.09957 | 1.09509 | 1.11625 |
| 200 | 1.25672 | 1.12190 | 1.08726 | 1.08137 | 1.07962 | 1.10232 |
| 250 | 1.20502 | 1.09980 | 1.07423 | 1.07180 | 1.06933 | 1.09418 |
| 300 | 1.17076 | 1.08403 | 1.06379 | 1.06292 | 1.06213 | 1.08767 |
| 350 | 1.14605 | 1.07376 | 1.05621 | 1.05757 | 1.05537 | 1.08342 |
| 400 | 1.12831 | 1.06455 | 1.05055 | 1.05334 | 1.05154 | 1.07979 |

Table 4b: Simulation Data for LL-GREEDY on Diamond Dags.
($rep = 50$, 99% confidence interval $= \pm 0.73151\%$)

| $d$ | $\bar{\pi} = 5$ | $\bar{\pi} = 10$ | $\bar{\pi} = 15$ | $\bar{\pi} = 20$ | $\bar{\pi} = 25$ | $\bar{\pi} = 30$ |
|---|---|---|---|---|---|---|
| 50 | 2.10918 | 1.47164 | 1.29543 | 1.23047 | 1.19317 | 1.19377 |
| 100 | 1.52502 | 1.23589 | 1.15804 | 1.13594 | 1.12234 | 1.13961 |
| 150 | 1.34448 | 1.15996 | 1.11264 | 1.10006 | 1.09416 | 1.11811 |
| 200 | 1.25587 | 1.12185 | 1.08772 | 1.08372 | 1.07933 | 1.10391 |
| 250 | 1.20434 | 1.09968 | 1.07440 | 1.06979 | 1.06909 | 1.09523 |
| 300 | 1.17061 | 1.08340 | 1.06391 | 1.06417 | 1.06118 | 1.08989 |
| 350 | 1.14623 | 1.07303 | 1.05692 | 1.05771 | 1.05604 | 1.08429 |
| 400 | 1.12823 | 1.06493 | 1.05122 | 1.05309 | 1.05212 | 1.08009 |

Table 3c: Simulation Data for LS on Linear Algebra Task Graphs.
($rep = 50$, 99% confidence interval $= \pm 1.13900\%$)

| $v$ | $\bar{\pi} = 5$ | $\bar{\pi} = 10$ | $\bar{\pi} = 15$ | $\bar{\pi} = 20$ | $\bar{\pi} = 25$ | $\bar{\pi} = 30$ |
|---|---|---|---|---|---|---|
| 50 | 1.51834 | 1.13785 | 1.07327 | 1.06015 | 1.06364 | 1.07162 |
| 100 | 1.13175 | 1.04123 | 1.02993 | 1.03579 | 1.03937 | 1.04903 |
| 150 | 1.05896 | 1.02260 | 1.02150 | 1.02489 | 1.03138 | 1.03957 |
| 200 | 1.03392 | 1.01472 | 1.01733 | 1.02133 | 1.02606 | 1.03442 |
| 250 | 1.02237 | 1.01119 | 1.01432 | 1.01738 | 1.02304 | 1.03103 |
| 300 | 1.01577 | 1.00979 | 1.01174 | 1.01683 | 1.01996 | 1.02876 |
| 350 | 1.01222 | 1.00865 | 1.01178 | 1.01447 | 1.01934 | 1.02699 |
| 400 | 1.01006 | 1.00706 | 1.01027 | 1.01420 | 1.01840 | 1.02377 |

Table 4c: Simulation Data for LS on Diamond Dags.
($rep = 50$, 99% confidence interval $= \pm 0.73406\%$)

| $d$ | $\bar{\pi} = 5$ | $\bar{\pi} = 10$ | $\bar{\pi} = 15$ | $\bar{\pi} = 20$ | $\bar{\pi} = 25$ | $\bar{\pi} = 30$ |
|---|---|---|---|---|---|---|
| 50 | 1.49805 | 1.12749 | 1.07633 | 1.06616 | 1.06490 | 1.07154 |
| 100 | 1.12508 | 1.04232 | 1.03061 | 1.03342 | 1.04085 | 1.04809 |
| 150 | 1.05898 | 1.02073 | 1.02052 | 1.02522 | 1.03266 | 1.04001 |
| 200 | 1.03282 | 1.01480 | 1.01708 | 1.01992 | 1.02701 | 1.03474 |
| 250 | 1.02220 | 1.01154 | 1.01393 | 1.01894 | 1.02450 | 1.03068 |
| 300 | 1.01642 | 1.01022 | 1.01236 | 1.01654 | 1.02223 | 1.02682 |
| 350 | 1.01235 | 1.00830 | 1.01175 | 1.01519 | 1.02031 | 1.02577 |
| 400 | 1.00959 | 1.00711 | 1.01057 | 1.01457 | 1.01818 | 1.02447 |