INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Precise Evaluation of the Efficiency and the Robustness of Stochastic DAG Schedules

Louis-Claude Canon — Emmanuel Jeannot

UHP, INRIA

N° 6895

April 2009

Thème NUM

ISSN 0249-6399 ISRN INRIA/RR--6895--FR+ENG

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

centre de recherche
NANCY - GRAND EST

Precise Evaluation of the Efficiency and the Robustness of Stochastic DAG Schedules*

Louis-Claude Canon, Emmanuel Jeannot
UHP, INRIA

Thème NUM — Systèmes numériques
Équipes-Projets AlGorille

Rapport de recherche n° 6895 — April 2009 — 19 pages

Abstract: This study is devoted to the evaluation of schedules of parallel applications consisting of a set of stochastic tasks having precedence constraints between them. This is an important issue when the goal is to evaluate the efficiency and the robustness of a schedule. In this case, evaluating the makespan consists to evaluate an arithmetic expression with random variables. Results show that the proposed methods provide a reasonable trade-off between precision and computation time.

Key-words: Stochastic DAG; Random Variable; Max+; Correlation.

* A preliminary version of this paper appeared in *10ème congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF)*, Nancy, France, February 2009, pages 13–24.

Centre de recherche INRIA Nancy – Grand Est
LORIA, Technopôle de Nancy-Brabois, Campus scientifique,
615, rue du Jardin Botanique, BP 101, 54602 Villers-Lès-Nancy
Téléphone : +33 3 83 59 30 00 — Télécopie : +33 3 83 27 83 19

Évaluation Précise de l'Efficacité et la Robustesse d'Ordonnancements de Graphe de Tâches Stochastiques †

Résumé : Cette étude est consacrée à l'évaluation des ordonnancements d'applications parallèles se modélisant par un ensemble de tâches aléatoires soumises à des contraintes de précédences. Cette problématique se rencontre lors de l'évaluation de l'efficacité et de la robustesse d'un ordonnancement. Dans ce cas, évaluer le makespan consiste à évaluer une expression arithmétique portant sur des variables aléatoires. Les méthode proposées présentent un compromis raisonnable entre la précision et le temps de calcul.

Mots-clés : Graphe de Tâches Stochastiques ; Variable Aléatoire ; Max+ ; Corrélation.

† Une version préliminaire de cet article a été publié lors du *10ème congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF)*, Nancy, France, février 2009, pages 13–24.

Contents

1 Introduction

Nowadays systems are full of uncertainties. This is especially the case when executing an application on a parallel distributed system. In this case, the application can be modeled by a DAG (directed acyclic graph) where each task and message are represented by a node and an edge, respectively. Executing this application on the environment requires to schedule the DAG, *i.e.*, assign each task to a processor and define the order of execution. In this context, uncertainties can come from different factors: task costs that cannot be deterministically predicted (exact duration are obtained once executed); system models are too simplistic and do not capture all the phenomena (cache effect, system performance, etc.); costs depend on data inputs (for different data set, duration change).

When communication and task durations cannot be known deterministically, we can model these durations with random variables (RVs). Therefore, the duration of the schedule (the makespan) is also a RV and is defined by a *distribution*. However, evaluating precisely the makespan distribution of one schedule with stochastic costs is difficult (it is shown to be #P-Complete[1] in [12] when distributions are discrete).

This study is mainly motivated by off-line scheduling that consists in preparing a good schedule before an application's execution. This minimizes the cost of dynamic re-scheduling and simplifies the implementation. However, this step need to be the fastest possible and requires evaluation phases. Therefore, in order to design efficient scheduling heuristics, we need an efficient evaluation scheme for characterizing the makespan distribution.

In this work, we assume that the scheduling heuristic is given and we address the problem of approximating the makespan distribution when tasks and communication durations are modeled by a RV. We target two metrics: the efficiency of the schedule (given by the mean of the makespan distribution) and the robustness of the schedule (the ability to absorb some degree of uncertainty in task and communication duration). More precisely, we define the robustness as the system's capacity to give the same output independently of inputs' variations. It is shown in [5] that a good metric for this criterion is the variance of the makespan (the narrower the distribution, the lower the variance and the less uncertainty in the application execution time).

Unfortunately, as shown in the next section, existing approximation methods either favor precision or quickness, and do not present an acceptable trade-off.

The contribution of this paper is to propose a mechanism based on Clark's moment matching approach [7] that considers correlations between each RV and that approximates tightly the true mean and variance of the makespan with an acceptable complexity.

2 Related work

Evaluating the makespan distribution is central to many problematics and exhibits some variations in each concerned field. It was first introduced in [15] as evaluation of PERT (Program Evaluation and Review Technique) networks

[1] intuitively a #P problem consists in counting the number of solutions to a NP problem.

(graphs where vertices correspond to activities with stochastic durations). Estimating the duration probability distribution of such projects involves applying addition and maximum on RVs (random variables). A large number of articles were published later in the field of operations research on that subject [4, 10, 13, 14, 16, 18, 21]. Scientists are also concerned by this issue when representing a parallel application as a stochastic DAG [22, 6]. In this case, duplication can be allowed for performance or reliability reasons [1], and involves the evaluation of minimum operations. The problem is also known as STA (statistical timing analysis) in the field of digital circuit optimization where signal delay distributions among all chips has to be approximated. In such cases, spatial correlations between delays need to be tackled, which increases even more the complexity of the problem.

Since the concerned literature is huge, we focus on 3 kinds of contributions: approaches based on Clark's formulas; bounding methods; and, Monte Carlo simulations. In [7], Clark proposes formulas for computing the first 4 moments of the maximum of 2 possibly dependent Gaussian distributions. In [18], Sculli derives a method for evaluating the makespan by considering only the first 2 moments. This approach is similar to ours, except that all correlations are ignored. In [13], Kamburowski provides bounds on the result by using a similar approach. Mostly focused on spatial correlations, many articles related to STA problem do not present in detail how to deal with correlations due to re-convergent paths. For example, the authors of [17] describe how to apply principal component analysis that addresses directly the existence of spatial correlations.

Martin and Dodin [16, 10] provided an upper bound by transforming a DAG into a series-parallel graph, which can be accurately evaluated in polynomial time. Although lot of work has been proposed to produce tight bounds [14], we prefer to obtain quickly the most accurate result even if no guarantee is available.

Finally, research has been conducted in order to develop Monte Carlo simulations [21, 4]. The idea resides in computing deterministically the makespan by instantiating each RV a large number of times. This allows computing the empirical distribution function, which converges to the true law of the makespan. Although precise, these methods require too much time to be used inside a scheduling heuristic.

3 Model

We assume that the DAG, the target platform and the schedule are given. Based on these inputs (DAG, platform, schedule), we compute a new DAG with the same set of vertices but with new edges that comes from execution ordering constraints defined by the schedule. These edges are called transitive edges [19] and assure the correct execution order of the application. Then, we transform all the communication edges into vertices with appropriate costs for a simpler model. Lastly, the stochastic DAG is obtained by separating vertices into two sets (alongside with the insertion of some necessary pseudo-vertices, as explain later) to discriminate maximum from minimum operations.

In such stochastic DAG, $G = (V, E)$, each vertex of V represents a RV (random variable) and edges represent dependencies between these RVs. Since we allow schedule with duplications, we characterize 2 disjoint subsets of V: V_1,

for maximums, and V_2, for minimums (with $V_1 \cup V_2 \subseteq V$). When a node has several predecessors, there are three cases. If there is no duplicate among the predecessors, it is put in V_1 (it is a regular *join* in the application DAG). If its predecessors (communications put aside) are only duplicates of the same task, this node is put in V_2. Otherwise, when only a strict subset of the predecessors are duplicated tasks, insertions of vertices with zero cost occur: we add a pseudo-vertex of null duration, which is put in V_2, for regrouping the precedence constraints of each duplicated preceding task. A last vertex, belonging to V_1, is connected to these inserted vertices and guarantees that the task begins when at least one duplicate of every predecessors is finished.

To summarize, a single edge between two vertices corresponds to the addition of their RVs. When several edges are connected towards the same node, we have two different operations. A *maximum* of the RVs if the arriving node is in V_1 and a *minimum* of the RVs if the node is in V_2. Intuitively, we use a maximum when all the predecessors need to finish their executions to start the task, and we use a minimum when only one predecessor (a duplicate) is needed to start the task.

In its most basic form, the problem consists to evaluate computationally mathematical expressions formed by a sequence of maximum, minimum and addition operations on possibly dependent RVs. The objective is to characterize the probability density function of the results of such expressions.

Formally, f denotes the function that maps each vertex to a RV and g the function that maps each vertex to the RV representing its ending time.

- $\mathrm{Pred}(v) = \{v_i : (v_i, v) \in E\}$
- $\forall v \in V_1, g(v) = \max_{v_i \in \mathrm{Pred}(v)} g(v_i) + f(v)$
- $\forall v \in V_2, g(v) = \min_{v_i \in \mathrm{Pred}(v)} g(v_i) + f(v)$

For each other vertex ($v \in V \backslash (V_1 \cap V_2)$), which are those having only one predecessor (denoted as v_i), the local RV is added to the expression (namely, $g(v) = g(v_i) + f(v)$).

An example is depicted on Figure 1. First, a DAG is represented in a) and contains 6 vertices with communication costs corresponding to each precedence constraint. From this graph, we generate a schedule (the b) diagram) where task T_3 is duplicated. All the other tasks are assigned to one of the 2 available processors and local communications are assumed to have negligible durations. Since durations are RVs, a precise schedule (with deterministic starting and ending times) cannot be generated, and the schedule only specifies assignments and execution orders. The graph c) is obtained from this schedule by keeping required communication costs (those that take place between distinct processors) and by adding transitive edges (edges specifying the order of execution on one processor). The final graph d) illustrates the transformation realized in order to separate vertices into sets V_1 and V_2. Indeed, some of the predecessors of vertex T_4 in graph c) are duplicated tasks, except T_2. Thus, we create a pseudo vertex T_4' that has a zero cost and is put in V_2 (light gray vertices), while T_4 belongs to V_1 (dark gray vertices). Finally, the objective is to compute the end time of the exit vertex, $g(T_6)$, and the complete expression that need to be evaluated is the following (each f are removed for clarity):

$$T_6 + \max(\quad T_4 + \max(T_2 + T_1, T_4' + \min(T_3 + T_2 + T_1, c_{34} + T_3 + c_{13} + T_1)),$$
$$c_{56} + T_5 + \min(c_{35} + T_3 + T_2 + T_1, T_3 + c_{13} + T_1))$$

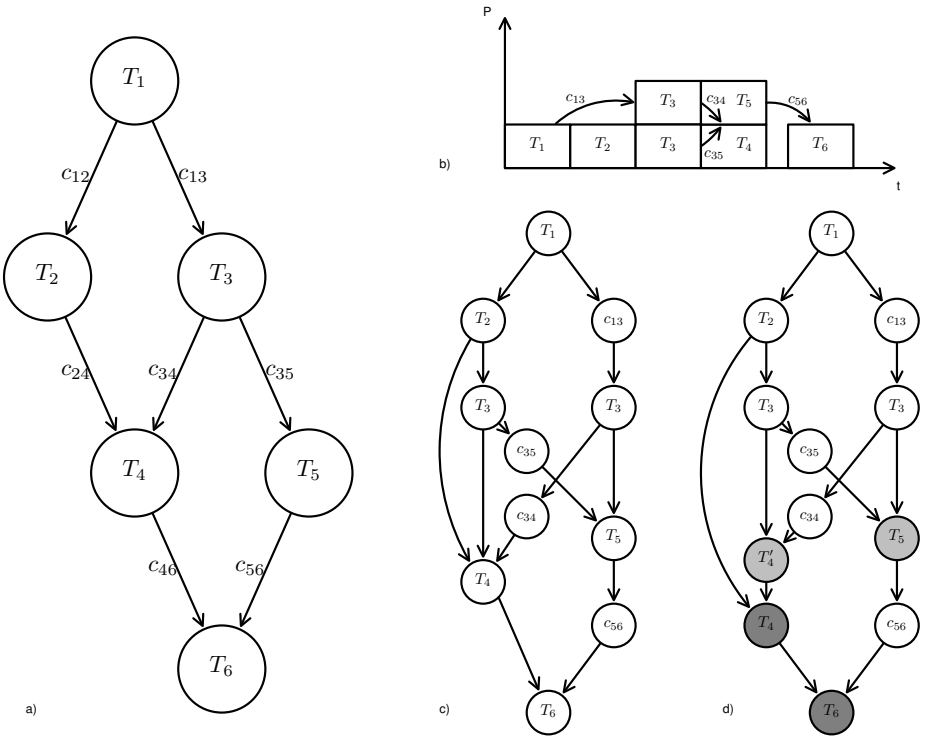


Figure 1: Example of a graph to evaluate (light gray vertices for minimums and dark gray for maximums)

This representation allows modeling many expressions with additions, minimums and maximums. It is easy to see that for the exit node v_{exit} of the DAG, the size of the expanded expression from which $g(v_{\text{exit}})$ results is proportional to the number of paths in the DAG. However, the number of paths in a DAG can be exponential. Therefore, we need a way to precisely approximate the computation of this expression. This is what is proposed in the next section.

4 Cordyn: a dynamic programming based approach

In this section, we present a novel algorithm, called *Cordyn*, that encompasses Clark's moment matching approach [7] into a dynamic programming based approach.

4.1 Assumptions

The main assumption is that the resulting RV (random variable) follows a Gaussian distribution. Indeed, for sufficiently large graph (for which an exact evaluation is intractable), the evaluated expression contains additions of a large number of RVs. If we suppose that maximum and minimum operations do not have a significant influence and that RVs are mostly independents, we can apply the central limit theorem and state that the resulting RV is approximately normally distributed. The following facts tend to worsen this approximation: low graph's depth; highly dependents RVs; not normal RVs; and, maximum and minimum operations applied to similar terms. However, experiments show that this normality hypothesis is often verified [5].

The second assumption is that each intermediate RV can be reduced to its mean and variance. These are the minimal values needed for exact additions. For maximums and minimums, we suppose the resulting RV to be also a Gaussian. Therefore, for each RV, only the pair mean/variance is considered. Once again, the closer initial distributions are to normals, the more this approximation is accurate. Also, when maximums and minimums are applied to RVs with different characteristics, the result is more precise.

Lastly, Clark's moment matching approach is applied to non-normal distributions when doing maximums, and the impact on the overall precision is assumed to be negligible.

The efficiency of this method depends on the correctness of these assumptions and will be discussed later.

4.2 Operation rules

This first method is based on Clark's formulas [7] for determining the first 2 moments of the maximum of 2 Gaussian distributions while taking into account their correlations. We recall these formulas and propose equivalent derivations for the minimum operation.

Let us consider two RVs, ε and η, with respective means μ_ε and μ_η, and variances σ_ε^2 and σ_η^2. Let $\rho_{\varepsilon,\eta}$ be the linear correlation coefficient between ε and η (this will be discussed later). Let $\varphi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$ and $\Phi(x) = \int_{-\infty}^{x} \varphi(t)dt$. Finally, let $\mu_{\max}$ and $\sigma_{\max}^2$ (resp. $\mu_{\min}$ and $\sigma_{\min}^2$, and μ_{sum} and σ_{sum}^2) be the mean and variance of the maximum (resp. minimum and sum) of ε and η. We extend Clark's formulas for the minimum operation as follows:

$$a = \sqrt{\sigma_\varepsilon^2 + \sigma_\eta^2 - 2\sigma_\varepsilon\sigma_\eta\rho_{\varepsilon,\eta}}$$

$$\alpha = \frac{\mu_\varepsilon - \mu_\eta}{a}$$

$$\mu_{\max} = \mu_\varepsilon\Phi(\alpha) + \mu_\eta\Phi(-\alpha) + a\varphi(\alpha) \tag{1}$$

$$\begin{aligned} \sigma_{\max}^2 &= (\mu_\varepsilon^2 + \sigma_\varepsilon^2)\Phi(\alpha) + (\mu_\eta^2 + \sigma_\eta^2)\Phi(-\alpha) \\ &+ (\mu_\varepsilon + \mu_\eta)a\varphi(\alpha) - \mu_{\max}^2 \end{aligned} \tag{2}$$

$$\mu_{\min} = \mu_\varepsilon \Phi(-\alpha) + \mu_\eta \Phi(\alpha) - a\varphi(\alpha) \tag{3}$$

$$\begin{aligned} \sigma^2_{\min} &= (\mu^2_\varepsilon + \sigma^2_\varepsilon)\Phi(-\alpha) + (\mu^2_\eta + \sigma^2_\eta)\Phi(\alpha) \\ &- (\mu_\varepsilon + \mu_\eta)a\varphi(\alpha) - \mu^2_{\min} \end{aligned} \tag{4}$$

$$\mu_{\text{sum}} = \mu_\varepsilon + \mu_\eta \tag{5}$$

$$\sigma^2_{\text{sum}} = \sigma^2_\varepsilon + 2\sigma_\varepsilon\sigma_\eta\rho_{\varepsilon,\eta} + \sigma^2_\eta \tag{6}$$

Additionally, Clark proposed a formula for determining correlations between RVs (extended to the minimum case by us):

$$\rho_{\tau,\max(\varepsilon,\eta)} = \frac{\sigma_\varepsilon\rho_{\tau,\varepsilon}\Phi(\alpha) + \sigma_\eta\rho_{\tau,\eta}\Phi(-\alpha)}{\sigma_{\max}} \tag{7}$$

$$\rho_{\tau,\min(\varepsilon,\eta)} = \frac{\sigma_\varepsilon\rho_{\tau,\varepsilon}\Phi(-\alpha) + \sigma_\eta\rho_{\tau,\eta}\Phi(\alpha)}{\sigma_{\min}} \tag{8}$$

$$\rho_{\tau,\text{sum}(\varepsilon,\eta)} = \frac{\sigma_\varepsilon\rho_{\tau,\varepsilon} + \sigma_\eta\rho_{\tau,\eta}}{\sigma_{\text{sum}}} \tag{9}$$

4.3 Algorithm

The main problem resides in the characterization of correlations between each RV. Since the previous formulation exhibits an overlapping sub-structure, sub-optimal computations occur when determining the correlation coefficient between two RVs with a classic top-down recursion.

Therefore, we use a dynamic programming method with a bottom-up approach. A $2n \times 2n$ symmetric matrix P containing each $\rho_{f(v_i),f(v_j)}$, $\rho_{f(v_i),g(v_j)}$ and $\rho_{g(v_i),g(v_j)}$ is used for memoization.

The DAG has then to be traversed in a topological order (vertices are ordered such that $\forall v_i, v_j \in V, i < j \Rightarrow \{v_j, v_i\} \notin E$) while the matrix P is updated at each step. This avoids to recompute any coefficient. The initial data consists in the correlation coefficients between each RV, $\rho_{f(v_i),f(v_j)}$, which are supposed to be known (this allows tackling spatial correlations for STA). In the case of task or activity scheduling, costs are independent, *i.e.*, $\forall i \neq j, \rho_{f(v_i),f(v_j)} = 0$ and $\rho_{f(v_i),f(v_i)} = 1$.

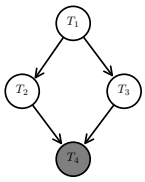
Each step are described in Algorithm 1. When the in-degree of a vertex is strictly greater than 2, maximums or minimums are computed pairwise in an arbitrary order.

In order to determine the complexity of this algorithm, we introduce some notations: let $\deg^-(v)$ be the in-degree of vertex v, $|V| = n$ and $|E| = m$. The most costly steps consist to characterize the correlations between the maximum of several RVs and each previous RVs (at line 6). Determining one correlation coefficient costs $\Theta(\deg^-(v_i))$ operations and is repeated i times for each RV. Thus, the final time complexity is $\Theta(\sum_{i=1}^{n}(i\deg^-(v_i))) = \Theta(nm)$. Moreover, the method need $\Theta(n^2)$ space elements for storing matrix P.

Algorithm 1 Cordyn dynamic programming algorithm

for all $i \in [1..n]$ **do**
  if $v \in V_1$ **then**
    let $\tau = \max_{v_j \in \mathrm{Pred}(v_i)} g(v_j)$
    compute μ_τ and σ_τ with Eq. 1 and 2
    for all $j \in [1..i-1]$ **do**
      compute $\rho_{\tau,g(v_j)}$ with Eq. 7
    end for
  else
    analogously to $v \in V_1$ but with Eq. 3, 4, and 8
  end if
  compute $\mu_{g(v_i)} = \mu_{\mathrm{sum}(\tau,f(v_i))}$ and $\sigma_{g(v_i)} = \sigma_{\mathrm{sum}(\tau,f(v_i))}$ with Eq. 5 and 6
  for all $j \in [1..i-1]$ **do**
    compute $\rho_{g(v_i),g(v_j)}$ with Eq. 9 and update P
  end for
  for all $j \in [1..n]$ **do**
    compute $\rho_{g(v_i),f(v_j)}$ with Eq. 9 and update P
  end for
end for

Figure 2 depicts all the evaluation steps on a small graph where costs follow exponential laws with rate 1. These characteristics present the main limitations of this method.



End times	Sculli		Cordyn		Monte Carlo	
	μ	σ^2	μ	σ^2	μ	σ^2
$g(T_1)$	1	1	1	1	1	1
$g(T_2)$	2	2	2	2	2	2
$g(T_3)$	2	2	2	2	2	2
$g(T_4)$	3.80	2.36	3.56	2.68	3.50	3.25

Figure 2: Example of intermediate values in an unfavorable situation

5 Experimental results

The experimental analysis of this heuristic involves two parts: first, we generate random DAGs and platforms. Good schedules are obtained with the Hul heuristic [6]. Then, we evaluate the makespan with our algorithm (Cordyn), with Sculli's approach [18], and with a method based on Monte Carlo simulations. The main metric that characterizes the precision of the methods under study is the relative error between the theoretical and the estimated variance (more critical than the mean's error). With $1,000,000$ simulations, the Monte Carlo method generates the reference results. Indeed, by keeping the assumption that the makespan is normally distributed, the relative error on the variance can be estimated with a confidence level of 99% to be less than 0.33%. On the overall, roughly 1 thousand schedules are generated.

A first category of DAG is obtained from the *Strassen* algorithm description, which constitutes a concrete application. For being representative, two other categories are obtained through random generation accordingly to Tobita's and Kasahara's methods [20], namely *samepred* (each created node can be connected to any other existing nodes) and *layrpred* (nodes are arranged by layers).

The distributions of the costs in the task graphs follow a Beta distribution where the parameters are such that the probability distribution corresponds to our observations and expectations. For this purpose, we need a well-defined nonzero mode (implying $\alpha > 1$) and more small values than large ones (meaning we should have a right-skewed probability distribution, thus $\beta > \alpha$). Therefore, we select $\alpha = 2$ and $\beta = 5$.

Every parameter used to settle tasks graphs are summarized in Table 1, alongside with some selected values. For each type of graph, we vary the number of tasks, average execution and communication costs, the average number of edges per node, the distributions and the associated uncertainty level (UL, the ratio between the maximum and the minimum of a RV (random variable), or between the 0.999-quantile and the 0.001-quantile when the extrema are infinite). Hence, the larger the UL, the greater each RV's variance. Additionally, we change the seed to obtain different graphs. Finally, we model heterogeneity by using the coefficient-of-variation (COV) that defines a ratio between the mean and the standard deviation of a given value in order to have a relative dispersion metric (see [2] for more details). In this study, we use a Gamma distribution to obtain the values inside each given graph. Some of the parameters are ignored for Strassen graphs: average communication cost (it is already induced by the number of tasks and average execution cost), costs' COV (the coefficient-of-variation associated with these 2 costs is zero) and the average number of edges per node. Besides, numbers of tasks are instead: 23, 163, and 1143.

Parameter	Values
Type of graph	*Strassen samepred layrpred*
Number of tasks (n)	10 30 100 300
Application's seed	*0 1 2 3 4 5 6 7 8 9*
Average execution cost (FLOP)	10 M 100 M 1 G
Average communication cost (B)	10 k 100 k 1 M
Costs' COV	0.001 0.1 0.3 0.5 1 2
Average number of edges per node	1. 3. 5.
Distribution	Beta
UL	1.0001 1.1 1.2 1.5 2 3 5
UL's COV	0.001 0.1 0.3 0.5 1 2

Table 1: Task graph parameters

If not otherwise specified, parameters values are set to $n = 100$, UL = 1.1, and the edge density of the graph (number of edges per node) is 3.

In all figures, measures are depicted with boxplots (five-number summary: the extreme of the lower whisker, the first quartile, the median, the third quartile and the extreme of the upper whisker). The whiskers extend to the most extreme data point which is no more than 1.5 times the interquartile range from the closest quartile. Values of any data points which lie beyond the extremes of

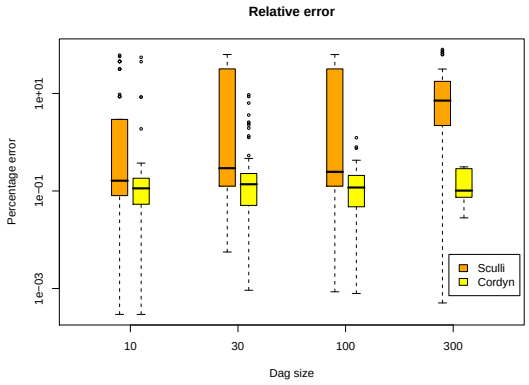


Figure 3: Influence of DAG size (number of RVs) on the precision of the makespan evaluation

the whiskers are also represented (the outliers). We see on Figure 3 that our heuristic has a much better precision than Sculli's approach, particularly for large instances. It can also be noted that the precision does not degrade as the size increases. The explanation lies in the accumulation of the errors done by ignoring correlations in Sculli's approach when DAGs become larger.

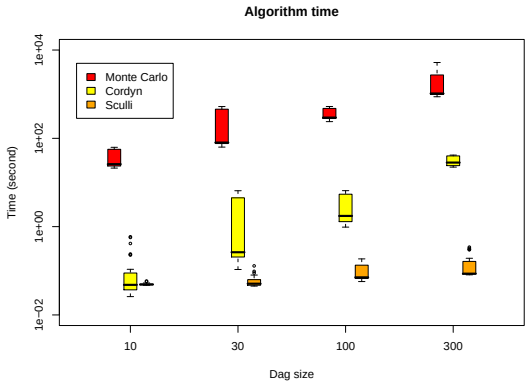


Figure 4: Influence of DAG size (number of RVs) on the time performance

However, as can be seen on Figure 4, the main drawback concerns the speed of this method. Indeed, Sculli's method has a time complexity $\Theta(m)$ and a space complexity $\Theta(n)$ while Cordyn has a time complexity $\Theta(nm)$ and a space complexity $\Theta(n^2)$. This restricts its application to static scheduling of medium-size DAGs (where Sculli's approach is too much imprecise and Monte Carlo method too slow), or evaluations, without time constraints, of schedules of extremely high sizes (where Monte Carlo method would also be too costly).

On Figure 5, we see the influence of UL values. Although precision decreases dramatically with higher UL on Sculli's approach, our method remains acceptable (1%) even for high value. The reason of these degradations comes from the

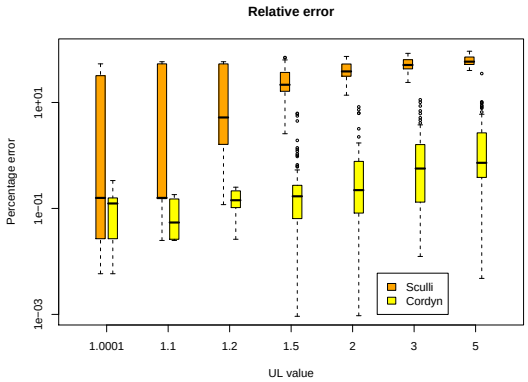


Figure 5: Influence of the uncertainty (UL)

approximation done when considering the maximum of 2 RVs as a Gaussian. The approximation is better when the ratio between the absolute difference of the 2 initial RV's means and their variances is large. When UL increases, this ratio decreases on average making Cordyn less accurate as a consequence.

Other parameters were tested: DAG's type, heterogeneity of the uncertainty (UL's COV), edge density (average number of edges per node), variation of task durations (average execution cost and costs' COV), etc. However, no significant conclusion could be drawn from these tests (Cordyn is always better than Scully's method and results do not depend on the parameters values). As an example, we show the effect of edge density on Figure 6.

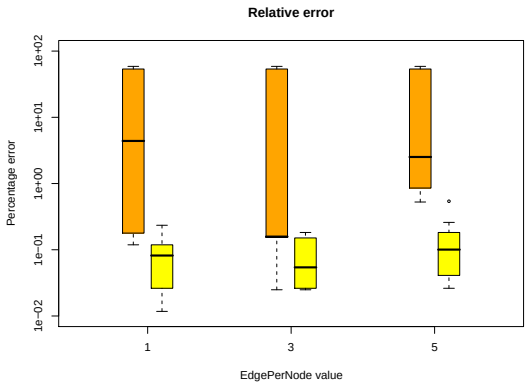


Figure 6: Influence of the edge density (average number of edges per node)

The same scheduling heuristic is applied to each DAG. In this case, a bias on this experimental study is possible. However, the large number of generated DAGs and their diversity minimize this risk. In addition, the Hul heuristic produces compact schedules, which lowers the validity of the assumptions made and makes the problem harder to solve with our approach. Indeed, Hul produces efficient and robust schedules, which are often the most compact possible.

Also, a compact schedule reduces the depth of the obtained stochastic DAG (due to the insertion of transitive edges), introduces more dependencies (for the same reason), and increases the likelihood of having similar RVs on which maximums or minimums are performed. In these circumstances, the possible bias is unfavorable to Cordyn.

6 CorLCA: using Lowest Commun Ancestor requests

Although Cordyn presents a reasonable trade-off in term of precision and speed, its complexity is too high for large graphs. We derive a new heuristic, called *CorLCA*, in this section in order to obtain a really fast approach while producing precise results. This heuristic assumes that tasks costs are independents.

6.1 Correlation tree

The main idea is to simplify the correlations computation part that takes most of the time in Cordyn. To this end, we build a tree while traversing the DAG that is used to denote correlations between any pair of end times. This tree, called the *correlation tree*, has the same nodes – that are added incrementally – than the original DAG and a subset of its edges. Specifically, each node has the same outgoing edges and only one incoming edge. To summarise, the algorithm is based on the following two steps:

- each time a node is traversed, it is added to the correlation tree and only one incoming edge (corresponding to one parent) is selected ;
- each time a maximum is evaluated, the correlation tree is used to determine efficiently the correlation between the operands. This correlation is then used when applying Clark's moment matching approach as in Cordyn (see Section 4).

Selecting one parent for a given node is done by determining which one of the predecessors has the most impact on the begin time of this node. In other words, when performing a maximum (or a minimum), we want to preserve only the relation with the predecessor's RV that influences the most the resulting RV. Thus, the correlation tree represents an approximate structure of the correlations between each pair of RVs.

We describe now the parent selection. The algorithm evaluate the makespan in a single DAG traversal. At each step, if the number of parent of the current node is greater than one, either a maximum or a minimum on the end times is performed (based on the same principle as Cordyn and Sculli's approach). In this case, the incoming edge that is selected in the correlation tree is the one that corresponds to the end time that is the closest to the begin time of the current node (or, the parent whose RV is the closest to the RV resulting from the operation). Indeed, most of the time, only one of the predecessors has a determinant impact on a begin time. We use the Bhattacharyya coefficient [3] that measures the similarity between two probability distribution and derives closed formulas for Gaussian distributions. Let $X_1 = \mathcal{N}(\mu_1, \sigma_1)$ and $X_2 = \mathcal{N}(\mu_2, \sigma_2)$ be two Gaussian distributions with probability density function $p(x)$

and $q(x)$. Then, the Bhattacharyya coefficient between these two RVs, $0 \leq D_B(X_1, X_2) \leq 1$, is defined as:

$$D_B(X_1, X_2) = \int \sqrt{p(x)q(x)}dx = \sqrt{\frac{2\sigma_1\sigma_2}{\sigma_1^2 + \sigma_2^2}} \times e^{-\frac{(\mu_1-\mu_2)^2}{4(\sigma_1^2+\sigma_2^2)}}$$

This coefficient is computed between each predecessor end times and the current node begin time. A Bhattacharyya coefficient equal to one means that the two RVs are identically distributed. Therefore, at each step, we select the parent for which this coefficient if the highest and add the corresponding edge to the correlation tree.

The correlation tree corresponding to the DAG on Figure 1 (last diagram) is depicted on Figure 7. For each node having several predecessors, one edge is selected.

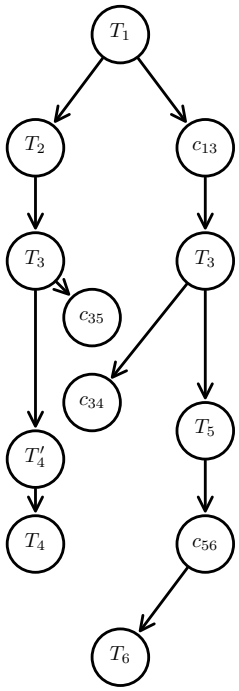


Figure 7: Example of correlation tree corresponding to the DAG of the last diagram of Figure 1

The correlation tree makes it possible to compute quickly any correlation. Indeed, by finding the lowest common ancestor (LCA) of two nodes, it is possible to compute directly the correlation between their RVs. Let X_1 and X_2 be the RVs of two nodes and let X be the RV corresponding the lowest common ancestor of these nodes. Then, under the assumptions that the correlation tree represents correctly the relation between the RVs, it is clear that $X_1 = X + Y$ and $X_2 = X + Z$ where Y (resp. Z) is the sum of the costs of the tasks that are on the path between the common ancestor and the node whose cost is X_1 (resp. X_2). Therefore, Y and Z are independents, because we consider that costs are independents. Therefore, the correlation between X_1 and X_2 is defined as:

$$\rho_{X_1,X_2} = \frac{\sigma_X^2}{\sigma_{X_1}\sigma_{X_2}}$$

6.2 Complexity

The complexity of this approach depends on the method used for computing a LCA query on a dynamic tree, namely a tree in which leaves are added between each query. Let δ be the optimal time complexity for performing leaf insertions and LCA queries on a tree and η be its space complexity. Then, the time complexity of CorLCA is $\Theta(m\delta)$ and its space complexity is $\Theta(n+\eta)$.

Cole and Hariharan present in [8] a method that performs node insertions and LCA queries in constant time assuming that insertions at most double the tree size. This assumption does not hold in our case and data structures have then to be rebuilt periodically. Moreover, they consider internal node insertions and deletions, which does not happen in our algorithm. Gabow describe in [11] an algorithm that performs leaf additions in constant amortized time.

Based on existing works, we conjecture that $\delta = \Theta(1)$ and $\eta = \Theta(n)$, leading to a time (resp. space) complexity of $\Theta(m)$ (resp. $\Theta(n)$) for our approach.

6.3 Experimental comparison

We summarize the precision of CordLCA relatively to Cordyn on Figure 8. The relative difference between both approaches (for both mean and standard deviation) is generated on a subset of the scenario cases described in Section 5. We observe that the relative difference of the means generated by both methods is less than 1% in 96% of cases and the relative difference of the standard deviations is below 1% in 77% of the cases.

For a lower complexity than Cordyn, CorLCA is able to generate comparable results in term of precision. However, the relative difference of standard deviations is greater than 20% in 2% of the cases. Indeed, lower precision is achieved when maximums or minimums are applied to similar terms. In these situations, the Bhattacharyya coefficient is high for more than one RV and selecting one of the parents leads to imprecision. Hence, this heuristic should be used carefully and avoided when several high coefficients are detected.

7 Conclusion

A schedule is said to be robust if it is able to absorb some degree of uncertainty. Evaluating the efficiency and the robustness of the schedule of an application modeled by tasks and communications with stochastic durations is a #P complete problem.

However, it is important to have a precise approximation method in order to evaluate the quality of a given schedule in order to design efficient scheduling heuristics in this context.

We have developed a precise approximation scheme that can be used in any fields (operation research, parallelism or STA) since correlations between any pair of RVs are exploited by Cordyn. Its precision is better than an existing fast method, *i.e.*, Sculli's approach, especially when the degree of uncertainty is high (input RVs have large variance).

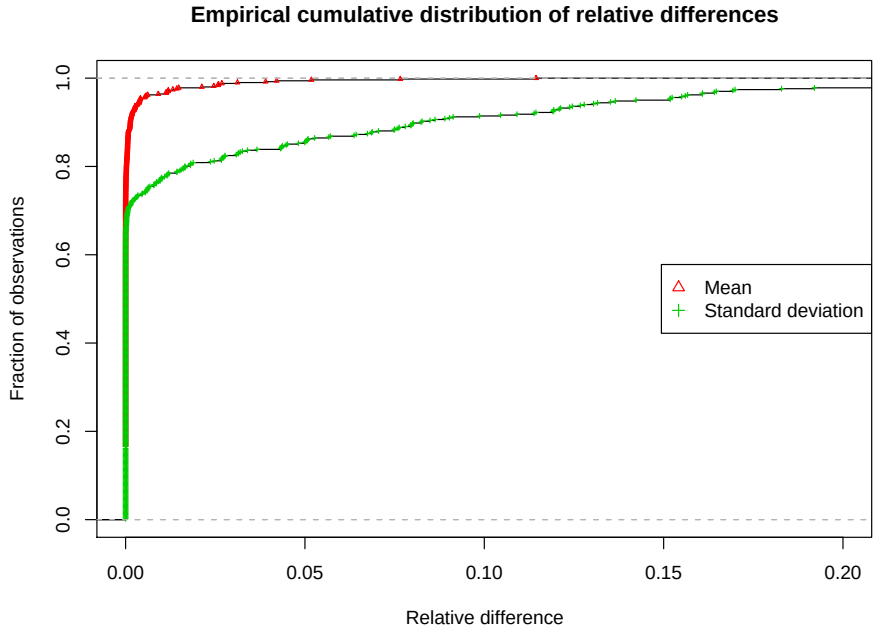


Figure 8: Empirical cumulative distributions of relative differences of means and standard deviations between Cordyn and CorLCA

However, the efficiency can still be improved. Both time and space complexity are not optimal since most of the calculated correlation coefficients are not used. Reducing this complexity raises some pitfalls because it involves to find the best topological order in which the DAG should be traversed (this is related to the directed versions of the sum cut and vertex separation problems [9]) and to characterize efficient data structures.

Concerning this last point, a second heuristic, CorLCA that behaves similarly in term of precision is presented. Its main advantages lies in its speed and lower complexity, though it should be used carefully when two end times are similar.

References

[1] Ishfaq Ahmad and Yu-Kwong Kwok. On Exploiting Task Duplication in Parallel Program Scheduling. *IEEE Transactions on Parallel and Distributed Systems*, 9(9):872–892, 1998.

[2] Shoukat Ali, Howard Jay Siegel, Muthucumaru Maheswaran, Debra Hensgen, and Sahra Ali. Representing Task and Machine Heterogeneities for Heterogeneous Computing Systems. *Tamkang Journal of Science and Engineering,* Special 50th Anniversary Issue, 3(3):195–207, November 2000.

[3] A. Bhattacharyya. On a measure of divergence between two statistical populations defined by their probability distributions. *Bull. Calcutta Math. Soc*, 35(4):99–109, 1943.

[4] John M. Burt and Mark B. Garman. Conditional Monte Carlo: A Simulation Technique for Stochastic Network Analysis. *Management Science*, 18(3):207–217, November 1971.

[5] Louis-Claude Canon and Emmanuel Jeannot. A Comparison of Robustness Metrics for Scheduling DAGs on Heterogeneous Systems. In *Heteropar'07*, pages 568–567, Austin, Texas, September 2007.

[6] Louis-Claude Canon and Emmanuel Jeannot. Scheduling Strategies for the Bicriteria Optimization of the Robustness and Makespan. In *11th International Workshop on Nature Inspired Distributed Computing (NIDISC 2008)*, Miami, Floride, USA, April 2008.

[7] Charles E. Clark. The Greatest of a Finite Set of Random Variables. *Operations Research*, 9(2):145–162, March 1961.

[8] Richard Cole and Ramesh Hariharan. Dynamic LCA Queries on Trees. In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pages 235–244. Society for Industrial and Applied Mathematics Philadelphia, PA, USA, 1999.

[9] Josep Díaz, Jordi Petit, and Maria J. Serna. A Survey of Graph Layout Problems. *ACM Computing Surveys*, 34(3):313–356, 2002.

[10] Bajis Dodin. Bounding the project completion time distribution in PERT networks. *Operations Research*, 33(4):862–881, July 1985.

[11] Harold N. Gabow. Data Structures for Weighted Matching and Nearest Common Ancestors with Linking. In *Proceedings of the first annual ACM-SIAM symposium on Discrete algorithms*, pages 434–443. Society for Industrial and Applied Mathematics Philadelphia, PA, USA, 1990.

[12] Jane N. Hagstrom. Computational complexity of PERT problems. *Networks*, 18(2):139–147, 1988.

[13] Jerzy Kamburowski. Normally Distributed Activity Durations in PERT Networks. *The Journal of the Operational Research Society*, 36(11):1051–11057, November 1985.

[14] George B. Kleindorfer. Bounding Distributions for a Stochastic Acyclic Network. *Operations Research*, 19(7):1586–1601, November 1971.

[15] D. G. Malcolm, J. H. Roseboom, C. E. Clark, and W. Fazar. Application of a Technique for Research and Development Program Evaluation. *Operations Research*, 7(5):646–669, September 1959.

[16] J. J. Martin. Distribution of the Time through a Directed, Acyclic Network. *Operations Research*, 13(1):46–66, January 1965.

[17] Sachin S. Sapatnekar and Hongliang Chang. Statistical Timing Analysis Considering Spatial Correlations using a Single Pert-Like Traversal. In *Computer Aided Design (ICCAD-2003)*, pages 621–625, San Jose, California, USA, November 2003.

[18] D. Sculli. The Completion Time of PERT Networks. *The Journal of the Operational Research Society*, 34(2):155–158, February 1983.

[19] Z. Shi, E. Jeannot, and J. J. Dongarra. Robust Task Scheduling in Non-Deterministic Heterogeneous Systems. In *Proceedings of IEEE International Conference on Cluster Computing*, Barcelona, Spain, October 2006. IEEE.

[20] Takao Tobita and Hironori Kasahara. A standard task graph set for fair evaluation of multiprocessor scheduling algorithms. *Journal of Scheduling*, 5(5):379–394, 2002.

[21] Richard M. van Slyke. Monte Carlo Methods and the PERT Problem. *Operations Research*, 11(5):839–860, September 1963.

[22] N. Yazia-Pekergin and Jean-Marc Vincent. Stochastic Bounds on Execution Times of Parallel Programs. *IEEE Transactions on Software Engineering*, 17(10):1005–1012, October 1991.

Centre de recherche INRIA Nancy – Grand Est
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Centre de recherche INRIA Bordeaux – Sud Ouest : Domaine Universitaire - 351, cours de la Libération - 33405 Talence Cedex
Centre de recherche INRIA Grenoble – Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier
Centre de recherche INRIA Lille – Nord Europe : Parc Scientifique de la Haute Borne - 40, avenue Halley - 59650 Villeneuve d'Ascq
Centre de recherche INRIA Paris – Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex
Centre de recherche INRIA Rennes – Bretagne Atlantique : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex
Centre de recherche INRIA Saclay – Île-de-France : Parc Orsay Université - ZAC des Vignes : 4, rue Jacques Monod - 91893 Orsay Cedex
Centre de recherche INRIA Sophia Antipolis – Méditerranée : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399