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

Louis-Claude Canon,¹ and Emmanuel Jeannot²

¹ Nancy Univerité – LORIA, Campus Scientifique, 54506 Vandœuvre-lès-Nancy Cedex France. louis-claude.canon@loria.fr
² INRIA Nancy Grand Est, Campus Scientifique, 54506 Vandœuvre-lès-Nancy Cedex France. emmanuel.jeannot@loria.fr

Abstract. Our 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 our proposed method provides a reasonable trade-off between precision and computation time.

Mots-Clefs. Stochastic DAG; Random Variable; Max+; Correlation.

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-Complete³ in [8] 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.

³ Intuitively a #P problem consists in counting the number of solutions to a NP problem.
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 [4] 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 [6] 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 [11] as evaluation of PERT (Program Evaluation and Review Technique) networks (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 [3,7,9,10,12,14,17]. Scientists are also concerned by this issue when representing a parallel application as a stochastic DAG [18,5]. 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 [6], Clark proposes formulas for computing the first 4 moments of the maximum of 2 possibly dependent Gaussians. In [14], 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 [9], 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 [13] describe how to apply principal component analysis that addresses directly the existence of spatial correlations.

Martin and Dodin [12,7] 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 [10], 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 [17,3]. 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 [15] 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. Our 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.

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

For each other vertex ($v \in V \setminus (V_1 \cap V_2)$), 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 our 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):

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

This representation allows modeling many expressions with additions, minimums and maximums. It is easy to see that for the exit node $v_{\text{exit}}$ 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 Method

4.1 Assumptions

Our starting 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 [4].

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 our method depends on the correctness of these assumptions and will be discussed later.

4.2 Operation rules

Our proposed method is based on Clark’s formulas [6] for determining the first 2 moments of the maximum of 2 Gaussians while taking into account their correlations. We recall these formulas and propose equivalent derivations for the minimum operation.

Let us consider two RVs, $\varepsilon$ and $\eta$, with respective means $\mu_\varepsilon$ and $\mu_\eta$, and variances $\sigma^2_\varepsilon$ and $\sigma^2_\eta$. Let $\rho_{\varepsilon,\eta}$ be the linear correlation coefficient between $\varepsilon$ and $\eta$ (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 $\mu_{\text{max}}$ and $\sigma^2_{\text{max}}$ (resp. $\mu_{\text{min}}$ and $\sigma^2_{\text{min}}$, and $\mu_{\text{sum}}$ and $\sigma^2_{\text{sum}}$) be the mean and variance of the maximum (resp. minimum and sum) of $\varepsilon$ and $\eta$. We extend Clark’s formulas for the minimum operation as follows:

Fig. 1. Example of a graph to evaluate (light gray vertices for minimums and dark gray for maximums)
\[ a = \sqrt{\sigma_z^2 + \sigma_{\eta}^2 - 2\sigma_z \sigma_{\eta} \rho_{z,\eta}} \]

\[ \alpha = \frac{\mu_z - \mu_\eta}{a} \]

\[ \mu_{\text{max}} = \mu_z \Phi(\alpha) + \mu_\eta \Phi(-\alpha) + a\varphi(\alpha) \] \hspace{1cm} (1)

\[ \sigma_{\text{max}}^2 = (\mu_z^2 + \sigma_z^2)\Phi(\alpha) + (\mu_\eta^2 + \sigma_\eta^2)\Phi(-\alpha) \]
\[ + (\mu_z + \mu_\eta) a\varphi(\alpha) - \mu_{\text{max}}^2 \] \hspace{1cm} (2)

\[ \mu_{\text{min}} = \mu_z \Phi(-\alpha) + \mu_\eta \Phi(\alpha) - a\varphi(\alpha) \] \hspace{1cm} (3)

\[ \sigma_{\text{min}}^2 = (\mu_z^2 + \sigma_z^2)\Phi(-\alpha) + (\mu_\eta^2 + \sigma_\eta^2)\Phi(\alpha) \]
\[ - (\mu_z + \mu_\eta) a\varphi(\alpha) - \mu_{\text{min}}^2 \] \hspace{1cm} (4)

\[ \mu_{\text{sum}} = \mu_z + \mu_\eta \] \hspace{1cm} (5)

\[ \sigma_{\text{sum}}^2 = \sigma_z^2 + 2\sigma_z \sigma_\eta \rho_{z,\eta} + \sigma_\eta^2 \] \hspace{1cm} (6)

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

\[ \rho_{\tau,\text{max}}(z,\eta) = \frac{\sigma_z \rho_{\tau,z}\Phi(\alpha) + \sigma_\eta \rho_{\tau,\eta}\Phi(-\alpha)}{\sigma_{\text{max}}} \] \hspace{1cm} (7)

\[ \rho_{\tau,\text{min}}(z,\eta) = \frac{\sigma_z \rho_{\tau,z}\Phi(-\alpha) + \sigma_\eta \rho_{\tau,\eta}\Phi(\alpha)}{\sigma_{\text{min}}} \] \hspace{1cm} (8)

\[ \rho_{\tau,\text{sum}}(z,\eta) = \frac{\sigma_z \rho_{\tau,z} + \sigma_\eta \rho_{\tau,\eta}}{\sigma_{\text{sum}}} \] \hspace{1cm} (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.

Algorithm 1 CORDYN dynamic programming algorithm
1: for all $i \in [1..n]$ do
2: if $v \in V_1$ then
3: let $\tau = \max_{v_j \in \text{Pred}(v_i)} g(v_j)$
4: compute $\mu_\tau$ and $\sigma_\tau$ with Eq. 1 and 2
5: for all $j \in [1..i-1]$ do
6: compute $\rho_{\tau,g(v_j)}$ with Eq. 7
7: end for
8: else
9: analogously to $v \in V_1$ but with Eq. 3, 4, and 8
10: end if
11: compute $\mu_{g(v_i)} = \mu_{\text{sum}(\tau,f(v_i))}$ and $\sigma_{g(v_i)} = \sigma_{\text{sum}(\tau,f(v_i))}$ with Eq. 5 and 6
12: for all $j \in [1..i-1]$ do
13: compute $\rho_{g(v_i),g(v_j)}$ with Eq. 9 and update $P$
14: end for
15: for all $j \in [1..n]$ do
16: compute $\rho_{g(v_i),f(v_j)}$ with Eq. 9 and update $P$
17: end for
18: end for

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 $O(\deg^{-}(v))$ operations and is repeated $i$ times for each RV. Thus, the final time complexity is $O(\sum_{i=1}^{n}(\deg^{-}(v_i))) = O(mn)$. Moreover, the method need $O(n^2)$ space elements for storing matrix $P$.

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 our method.

5 Experimental results

The experimental analysis of our heuristic involves two parts: first, we generate random DAGs and platforms. Good schedules are obtained with the Hul heuristic [5]. Then, we evaluate the
makespan with our algorithm (denoted as Cordyn), with Sculli’s approach [14], and with a method based on Monte Carlo simulations. Our 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 our reference results. Indeed, by keeping our 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 [16], namely samepred (each created node can be connected to any other existing nodes) and laypred (nodes are arranged by layers).

The distributions of the costs in the task graphs follow either a Beta, an exponential, or a normal distribution. The selected Beta distribution 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 our case, 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.

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

<table>
<thead>
<tr>
<th>End times</th>
<th>Sculli ( \mu )</th>
<th>Sculli ( \sigma )</th>
<th>Cordyn ( \mu )</th>
<th>Cordyn ( \sigma )</th>
<th>Monte Carlo ( \mu )</th>
<th>Monte Carlo ( \sigma )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( g(T_1) )</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( g(T_2) )</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>( g(T_3) )</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>( g(T_4) )</td>
<td>3.80</td>
<td>2.36</td>
<td>3.56</td>
<td>2.68</td>
<td>3.50</td>
<td>3.25</td>
</tr>
</tbody>
</table>

Fig. 2. Example of intermediate values in an unfavorable situation
Evaluation of Stochastic DAG Schedules

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type of graph</td>
<td>Strassen samepred layrpred</td>
</tr>
<tr>
<td>Number of tasks ((n))</td>
<td>10 100 1000</td>
</tr>
<tr>
<td>Application’s seed</td>
<td>0 1 2 3 4 5 6 7 8 9</td>
</tr>
<tr>
<td>Average execution cost (FLOP)</td>
<td>10M 100M 1G</td>
</tr>
<tr>
<td>Average communication cost (B)</td>
<td>10k 100k 1M</td>
</tr>
<tr>
<td>Costs’ COV</td>
<td>0.001 0.1 0.3 0.5 1 2</td>
</tr>
<tr>
<td>Average number of edges per node</td>
<td>1. 3. 5.</td>
</tr>
<tr>
<td>Distribution</td>
<td>Beta Exponential Gaussian</td>
</tr>
<tr>
<td>UL</td>
<td>1.0001 1.1 1.2 1.5 2 3 5</td>
</tr>
<tr>
<td>UL’s COV</td>
<td>0.001 0.1 0.3 0.5 1 2</td>
</tr>
</tbody>
</table>

Table 1. Task graph parameters

![Relative error](image1)

Fig. 3. Influence of DAG size (number of RVs) on the precision of the makespan evaluation

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 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.

However, as can be seen on Figure 4, the main drawback concerns the speed of our method.
Indeed, Sculli’s method has a time complexity \(O(n)\) and a space complexity \(O(n + m)\) while
ours has a time complexity \(O(nm)\) and a space complexity \(O(n^2)\). 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).
Fig. 4. Influence of DAG size (number of RVs) on the time performance

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 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 our method 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.

The same scheduling heuristic is applied to each DAG. In this case, a bias on our 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 our assumptions 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 our method.

6 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 our algorithm. 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).

However, the efficiency could still be improved. Both time and space complexity are not optimal since most of the calculated correlation coefficients are not used or only once. It is part of the future work to reduce this complexity. This raises some pitfalls since it involves to find the best topological order in which the DAG should be traversed and to characterize efficient data structures.

References

<table>
<thead>
<tr>
<th>EdgePerNode value</th>
<th>Relative error</th>
<th>Percentage error</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$10^{-2}$</td>
<td>$10^{0}$</td>
</tr>
<tr>
<td>3</td>
<td>$10^{-1}$</td>
<td>$10^{0}$</td>
</tr>
<tr>
<td>5</td>
<td>$10^{0}$</td>
<td>$10^{1}$</td>
</tr>
</tbody>
</table>

**Fig. 6.** Influence of the edge density (average number of edges per node)