Evaluation and Optimization of the Robustness of DAG Schedules in Heterogeneous Environments

Louis-Claude Canon and Emmanuel Jeannot

Abstract—A schedule is said to be robust if it is able to absorb some degree of uncertainty in task or communication durations while maintaining a stable solution. This intuitive notion of robustness has led to a lot of different metrics and almost no heuristics. In this paper, we perform an experimental study of these different metrics and show how they are correlated to each other. Additionally, we proposed different strategies for minimizing the makespan while maximizing the robustness: from an evolutionary metaheuristic (best solutions but longer computation time) to more simple heuristics making approximations (medium quality solutions but fast computation time). We compare these different approaches experimentally and show that we are able to find different approximations of the Pareto front for this bicriteria problem.

Index Terms—DAG, stochastic scheduling, robustness, makespan

1 Introduction

The problem of scheduling an application modeled by a task graph with the objective to minimize its execution time (makespan) is a well-studied problem [12], [22]. For instance, in the context of heterogeneous computing, different heuristics have been proposed in the literature to minimize the makespan such as HEFT [31], CPOP [31], etc. However, there are a lot of other possible objectives than minimizing the makespan. Among these objectives the *robustness* has recently received a lot of attention [1], [5], [9], [13], [28], [29]. A schedule is said to be robust if it is able to absorb some degree of uncertainty in the task or communication durations while maintaining a stable solution. The reason why robustness is becoming an important objective is the recent focus on large systems that can be dynamic and where uncertainty in terms of workload or resource usage can be very important. For instance, the duration of the tasks that compose the application and the communications between these tasks are subject to some uncertainties (due to the unpredictability of the behavior of the application and its sensitiveness to the input data). It is important to note that the robustness alone is not a metric but it gives an idea of the stability of the solution with regards to another performance metric such as schedule length, load balance of an application, queue waiting time of batch scheduler, etc. Moreover, a brief look at the literature shows that despite the fact that robustness is a very intuitive notion there is no consensus on a single metric. Conversely, almost each paper uses its own metric depending on the studied problem and the general context of the work. Furthermore, there does not exist a comparison between these different metrics. Hence, it is not possible to decide which metric to use when designing a heuristic.

LORIA, INRIA, CNRS, Nancy University

The contribution of this paper is to provide scheduling heuristics that optimize both robustness and makespan. To achieve this goal, we have proceeded in two parts. In a first part, we focus on comparing different metrics of robustness in the context of scheduling task graph on heterogeneous systems: we model an application as a set of tasks having precedence constraints. The performance metric we use is the makespan (the completion time of the application) and therefore, we look at the robustness of the makespan when tasks and communication may have variations (stochastically modeled by random variables) in their duration. Moreover, we try to see to which extend optimizing the makespan can help in optimizing the robustness. In other words, we try to answer the following question: *are short schedules more robust that long ones?* Therefore, the contribution of this first part is the following: we provide a comprehensive study of different robustness metrics in the case of task graph scheduling. We study how they are correlated to each other and whether robustness and makespan are conflicting objectives or not.

In a second part, based on the study of the first part and our understanding of robustness, we address our main problem: scheduling an application with the objective of minimizing the makespan and maximizing the robustness. The context of this second study is the same as the previous one: a heterogeneous environment where machines have different speeds and capabilities for which the robustness of a schedule is even harder to achieve. In this case, designing a scheduling heuristic is a complex task. One reason is that evaluating the makespan and the robustness in presence of stochas-

tic variations is a #P-Complete problem[1]. Moreover, minimizing the makespan is an NP-hard problem [22]. Another reason is that in this problem the robustness and the execution time are not equivalent objectives and therefore it requires a bicriteria approach to find all the Pareto-optimal solutions (solutions that are non-dominated) while these number of solutions can be very large and NP-hard to find. Based on that remark, we propose a suite of heuristics with two goals in mind. First, help the user to choose the trade-off between the computation time and the quality of the solution. Second, help the user to choose the trade-off between robustness and makespan. For the first trade-off (quality vs. computation time) we propose a genetic algorithm and very fast heuristics. For the second trade-off (robustness vs. makespan) each of the proposed heuristic is multi-objective and targets the approximation of the Pareto front (the set of Pareto-optimal solutions). Moreover, we present theoretical facts in order to prove that our multi-objective evolutionary algorithm (MOEA), directly inspired from NSGA-II [10] converges to set of Pareto-optimal solutions. Hence, it gives us a good approximation of the set of Pareto-optimal solutions.

The remaining of the paper is organized as follows. In Section 2 we present the problem and the notations used in this paper. The first part of this paper is devoted to the analysis of the robustness. Several works dealing with robustness are detailed in Section 3. The robustness metrics we use are described in Section 4. In Section 5 we present the experimental setup we used for testing and comparing the different metrics and heuristics. Comparison of the metrics are shown and discussed in Section 6. The second part of this paper is devoted to the design of bicriteria (makespan and robustness) scheduling strategies. We propose a MOEA and prove its convergence in Section 7. Our heuristics are presented in Section 8. In Section 9 we provide the experimental evaluation of the proposed heuristics. Finally, conclusion and future works are given in Section 10.

2 Problem description

2.1 Models of Application, Platform and Execution

We consider an application modeled by a stochastic task graph. A stochastic task graph is a directed acyclic graph (DAG) where nodes represent tasks and edges dependencies between these tasks. Each node and edge is valuated to represent respectively the execution cost (number of instructions) and the communication cost (number of bytes to be transmitted). To model the uncertainty, *i.e.* the fact that these costs are not deterministic, we use discrete random variables (RV) to draw these costs. Each RV gives the probability that an execution or communication cost is in a given interval. The only assumption made about these RV is that the first two moments (mean and variance) are finite.

The target platform is composed of a set of heterogeneous resources where each machine can communicate to any other machine. Each resource has different capacities in terms of network and communication speed.

Scheduling consists in allocating tasks to the processors while respecting the precedence constraints (given by the edges of the DAG), and the resource constraints (no processor executes two tasks at the same time). In this work, we consider only *eager schedule*, this means that each task, once allocated to a processor, starts as soon as possible in the same order than given by the schedule : There is no arbitrary delay (or slack) specifically added in the schedule. Note that most of the scheduling heuristics (list, clustering, etc.) produce eager schedules.

We use the *related model* [23] to simulate the task executions on the resources: each CPU i is given a value τ_i, the time to execute one instruction. This means that if the cost of a task T_x drawn from its random variable is c_x, the execution time of this task on processor i is $c_x\tau_i$. To compute the communication time between two tasks we proceed similarly. For instance, if we assume that task x is allocated on processor i and task y is allocated on processor j, then the communication time is given by: $l_{i,j} + d_{x,y} \times S_{i,j}$, where $d_{x,y}$ is the communication volume (drawn by the random variable) between T_x and T_y, $l_{i,j}$ is the latency between processors i and j, and $S_{i,j}$ is the time to send one data element between these two processors.

Since we use random variables to compute communication and task execution times, the makespan (completion time of the schedule) of each execution of the tasks on the resources can be different: given a schedule, it is possible to have a very large number of *realizations*[2] and hence a very large number of makespans. This poses two problems.

First, a schedule usually specifies when a task must start. Due to the stochastic model we use, it is not possible to ensure the start time of each task. This is why we use only eager schedules to address this problem: tasks are dynamically executed using an eager strategy where they start as soon as possible on their allocated processor while respecting the order given by the schedule.

The second problem is that we need to compute the distribution of the makespan in order to optimize our criteria. However, computing the distribution of the makespan is extremely difficult. Hagstrom [18] has shown it is a #P-complete problem when using discrete random variables. Therefore, having an accurate evaluation of this distribution is extremely costly. We detail how to compute the makespan distribution in the next section

1. intuitively a #P problem consists in counting the number of solutions of an NP-complete problem.

2. a realization is computed by instantiating every computation and communication durations according to the random variables.

2.2 Evaluating the Makespan Distribution

Given a schedule S we call f_S the makespan probability density function (PDF). With f_S, one can compute the probability that the makespan M is within two bounds $[x_1, x_2]$ (noted $(\Pr[x_1 \leq M \leq x_2])$ and is given by $\int_{x_1}^{x_2} f_S(x)dx$. We also use the cumulative distribution function (CDF) of the makespan F_S. F_S is the integral of the probability density function f_S. Therefore, $F_S(x)$ gives the probability that the makespan of schedule S is lower than x (noted $\Pr[M \leq x]$).

The PDF of the makespan comes directly from the distribution of the task duration and communication time. Computing the makespan distribution is a well-known problem that has mainly been studied in the context of PERT graph [24], which is a model very close to the one used here. Therefore, previous results can be used to our context.

Computing analytically the PDF or the CDF of the makespan is a difficult problem in the general case. However, sometimes, it becomes tractable, though computationally intensive. This is the case for task graphs with independent tasks and in which the paths do not diverge (an in-tree for instance). In this case, the distributions of each end times are independent and only two operations need to be considered (see [24] for the details). The first case is when a node x is the ancestor of another node y. The resulting distribution characterizing the end time of y is computed by adding the distributions of the end time of x and the duration of y. The sum of two distributions is computed by doing the convolution of the two probability density distributions and can be calculated numerically using Fast Fourier Transform (FFT)The other case is when two distributions are independent and join to another one. In this case we need to compute the maximum of the two distributions. The maximum of two independent distributions is done by multiplying their CDF.Here again, it can efficiently be calculated by finding the derivative of the probability density and integrating the result.

In the general case, however, DAGs have a structure such that end time distributions are dependent. Then, computing the probability distribution of the makespan becomes intractable: in the general case, it is #P-complete. Several authors have proposed solutions to approximate the distribution of the makespan for this case. Among these methods, two methods are of interest for our case.

A first method (called CorLCA in [7]) is based on the central limit theorem which states that the sum of random variables tends to be normally distributed. Every random variable is then simplified to its mean and standard deviation. They are indeed the only parameters needed to characterize any normal distribution. In this case sums are easy to compute by adding means and standard deviation and Clark's moment matching approach [8] is used for computing maximums (which are again approximated as normals), where correlations between tasks end time distributions are obtained through lowest common ancestor (LCA) queries. The makespan is then obtained without any numerical method. Such a method has a good trade-off between computational speed and accuracy of the computed distribution. Thus, it can be use when fast evaluations are required (*e.g.*, inside a greedy algorithm).

The second method is the Monte-Carlo one (see [32]), which consists in computing a great number of realizations, in order to simulate the makespan a sufficient number of times to obtain a good approximation of the distribution. Moreover, it is possible to limit the inaccuracy of the obtained distribution by setting bounds that depend on the number of realizations. It constitutes our reference methods when precise results are required (*e.g.*, for comparing different metrics).

2.3 The problem

Our goal is to provide scheduling strategies that optimize both schedule length and robustness.

Once we have a distribution of the makespan, we have to define which metrics have to be optimized. Concerning the schedule length, we use the *average makespan* of the distribution. Indeed, minimizing this metrics means that on the average we minimize the overall execution time. The problem of defining robustness has been widely studied in the literature. There exists several metrics for describing the robustness of the schedule with regards to the makespan. The next two sections are devoted to the problem of defining the robustness and comparing the different possible metrics and we show that a good metric is the *makespan standard deviation*. More precisely, we justify the fact that the first two moments are sufficient to characterize the makespan distribution.

Finally, let us sum up the problem. We are given a stochastic graph and a heterogeneous environment. The goal is to schedule the tasks on the processors such that the average makespan and the standard deviation of the makespan are both minimized. Since minimizing the makespan is known to be a NP-hard problem [22] and computing the makespan distribution is #P-complete, this problem is then in the class[3] NP-hard$^{\text{\#P-complete}}$.

Moreover, in the general case, both criteria are not completely dependent and several Pareto-optimal solutions exist. Hence, it requires bicriteria scheduling strategies.

3 Related Work

We start here by covering previous work on robustness metrics. How to measure robustness is a subject that has not yet led to a wide accepted metric. Several works propose different ways to measure this objective Ali, Maciejewski, Siegel and J.-K. Kim [1] do a good

3. a problem in class A^B is solvable by an algorithm in class A with an oracle for solving a problem in class B [4].

job in defining how to measure robustness: 1) defining the performance features that need to be robust, 2) identify the parameter that impacts the robustness 3) identify how a modification of these parameters impact on the performance features 4) identify the smallest collective variation of the parameters that make the performance features to violate acceptable variation. With this methodology the authors define a metric called the *robustness radius* that is the smallest variation of the parameters that make the performance features to exceed tolerable variation. In our case (scheduling tasks on heterogeneous system), the performance feature is the makespan, the parameters that impacts the robustness are the duration of each task and each communication. Hence, a schedule is said more robust than another one if it requires a greater change of durations to exceed some given bounds. The problem of that definition is that it is hard to take into account the fact that some change in task or communication duration are more likely to occur than others. Moreover, computing this metric requires a lot of effort and depends on the studied system.

In order to simplify the computation of the robustness, England, Weissman and Sadagopan [13] propose to use the Kolmogorov-Smirnov (KS) distance between the CDF of the performance metric under normal operating condition and the CDF of the same performance metric when perturbations occur. The idea is that if the KS distance is large (close to 1) then the two distributions are different, and thus, that a perturbation has a large impact on the behavior of the studied system. However, in many cases, the performance metric under normal operating condition has only one value (think for instance of the arrival time of the train at a station). In this case, the distribution is a Dirac delta function and the CDF is a *step* function. Moreover, if this value is computed using the minimum of each intermediate event, the KS distance is always 1 whatever the way you organize the system. This means that this metric is not well adapted to the case where the performance metric has only one possible value, which is the case for the scheduling problem studied here.

In [28] a subset of the authors of [1] proposes a new metric called the probabilistic metric. It is defined as the probability that the performance metric is confined within a given interval. They evaluate this metric against the robustness radius (called the deterministic robustness in the paper) and show that the probabilistic metric is preferable to the deterministic metric in the case of independent tasks scheduling.

Other definitions of the robustness are available in the literature. Bölöni and Marinescu [5] propose to use the slack as a robustness metric. The slack of a task represents a time window within which the task can be delayed without affecting the makespan. The same authors suggest also to use the entropy of the performance metric distribution to compare schedules with the same makespan: given two schedules with the same makespan they conjecture that the one with the smallest entropy is the most robust. In [29] the authors study another definition of slack and show that it is equivalent to the definition given in [5]. They propose two new robustness metrics for the scheduling problem. One is based on the average delay between the expected makespan and different realizations of the schedule under perturbation and the other is the ratio of realization that are late compared to the expected makespan. Moreover the authors show that minimizing the makespan is a contradictory objective with the problem of optimizing the robustness.

This brief look at the literature on robustness metrics shows that there is no consensus. This exemplifies the need for a comparison and a systematic study of different metrics in order to determine how these metrics are correlated to each other.

On the other side, few works had been done on robust scheduling. Most existing proactive methods are based on the insertion of slack or safety lead-time [9]. In [16], Gerasoulis *et al.* have studied the stability of the Dominant Sequence Clustering (DSC) algorithm [17] when task and communication durations are subject to bounded uncertainties. In Chapter 4 of [27], the sensitivity analysis of convex clustering is performed in the case of disturbance of computation durations. However, these two works use less general models than our stochastic graph one. Kim *et al.* [20] propose a scheme for digital integrated circuit sizing problems. In their approach, every statistic duration of the problem is reduced to a deterministic value and the problem is then solved. Their way to deal with uncertainty is to obtain this deterministic value by adding the mean and a multiple of the standard deviation of the stochastic duration.

Other works consider making use of possibility theory [11] that is more adapted to model imprecision. Fargier, Fortemps and Dubois [14] apply fuzzy logic to PERT as the evaluation of the makespan possibility distribution[4] is simpler than the evaluation of the probability distribution. They proposed to use pessimistic decision rules to obtain robust scheduling, however it cannot give the same insight than using usual stochastic evaluation and misses thus some stochastic aspects.

4 Robustness Metrics

As there is no consensus on a good metric definition, we compare most metrics proposed in the literature to each other[5]. For our problem, the robustness we measure is the stability of the makespan for any realization of a same schedule. Given a task graph and a target environment, we schedule the tasks and compute the makespan distribution. Let f be the PDF of the makespan of the schedule, F the CDF of the makespan of the same schedule and $E(M)$ the expected makespan (the average

4. a possibility degree can be viewed as an upper bound of a probability, hence, a possibility distribution encodes a family of probability distributions.

5. we do not compare all the metrics because some are not suited to the scheduling problem such as the robustness radius [1]

value of the makespan). Based on these definitions we define the following robustness metrics.

- **Makespan standard deviation**. Intuitively the standard deviation of the makespan distribution tells how narrow this distribution is. The narrower the distribution, the smaller the standard deviation is. This metric is related to the robustness because when you are given two schedules the one for which the standard deviation is the smaller is the one for which realizations are more likely to have a makespan close to the average value. Mathematically we have:

$$\sigma_M = \sqrt{E(M^2) - E(M)^2}$$

- **Makespan differential entropy**. Measuring the differential entropy of a distribution to assess the uncertainty of that distribution is proposed by Bölöni and Marinescu [5]. If there is less uncertainty there is more chance than two realizations give a close result and hence that the schedule is robust.

$$h(M) = -\int_{-\infty}^{+\infty} f(x) \log f(x) dx$$

- **Mean slack**. In the same paper ([5]) Bölöni and Marinescu proposed to use the slack as a metric of robustness. The slack of a task is defined as the amount of time this task can be delayed without delaying the schedule. This is computed using the spare time of the processors after the execution of this task:

$$t_{\text{spare}}(i,j) = t_{s_j} - t_{e_i} - c_{i,j}$$

  Where (i,j) is an edge in the input DAG, t_{s_j} is the start time of task j, t_{e_i} is the end time of task i and $c_{i,j}$ is the communication time between i and j. The slack of a task is the minimum of all the spare times on any path to an end task (the shortest path to an end-task in the graph of spare time). The slack of a schedule is the sum of the slack of all the tasks of the graph. The intuition is that the more slack in a schedule the less sensitive to change in durations is this schedule. In our case, we have random variables that define task and communication durations. Hence, we compute the expected value of the slack in the same way we do it for the makespan.
- **Probabilistic metric**. This metric is defined by Shestak, Smith, Siegel and Maciejewski [28] and gives the probability that the makespan is within two bounds. If this probability is high, this means that the makespan of a given realization is likely to be close to the average makespan and hence that the robustness is high. We propose two variants of this metric. An absolute probabilistic metric that measures the probability of the makespan to be within $[E(M) - \delta, E(M) + \delta]$ where $E(M)$ is the average makespan and δ a positive constant given by the user. We also propose the relative metric that measure the probability of the makespan to be within $[E(M) \times \frac{1}{\gamma}, E(M) \times \gamma]$, where γ is a real number greater than 1. Formally, The absolute probabilistic metric is defined as:

$$A(\delta) = \Pr\left[E(M) - \delta \leq M \leq E(M) + \delta\right]$$

  and the relative probabilistic metric is defined as:

$$R(\gamma) = \Pr\left[\frac{E(M)}{\gamma} \leq M \leq \gamma \times E(M)\right]$$

- **Lateness likelihood**. A schedule is said late if its makespan exceeds a given target such as the average makespan. The lateness likelihood, or miss ratio, is defined in [29] as the probability to be late. If this metric is large this means that the makespan tends to be often late and then that the robustness is low. It is defined as:

$$L = \Pr[M > E(M)]$$

- **Makespan 0.99-quantile**. Given a schedule, this metric measures the worst makespan it is possible to have in 99% of cases. This is a hybrid criterion between the efficiency and the robustness.

5 Experimental Setup

Our study is mostly empirical and takes place in a stochastic context. Moreover, the task graphs generation, the heterogeneous platform, the stochastic simulation require each a set of parameters. Therefore, considerable attention has been given to validate the methodology, which involves the selection of relevant instances for the problem, the metrics evaluation and the exploitation of the results.

Task graphs are generated from the *Strassen* algorithm description and randomly in two ways accordingly to Tobita and Kasahara [30], namely *samepred* (each created node can be connected to any other existing nodes) and *layrpred* (nodes are arranged by layers). Every parameter used to settle tasks graphs are shown in Table 1 alongside with the selected values. Each value in bracket is corresponding to a single experiment while the values outside are the default ones. More precisely, we do the cartesian product of each value that is not bracketed (leading to 3 basic settings one for each type of graph). And for each of these settings we change the default value by a bracketed one. Since we have 50 bracketed values we end-up with $50 \times 3 = 150$ experiments scenarios. For each kind of graph, we vary the number of tasks (TaskNumber), the execution and communication average costs (M_ExeCost and M_CommCost), the average number of edges per node (Avg Edge/Node), the distributions and the associated uncertainty level (UL, the ratio between the maximum and the minimum of a RV or between the 0.999-quantile and the 0.001-quantile when the previous values are inexistent). Additionally, we change the seed to obtain different graphs (SeedApp).

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 parameter in order to have a relative dispersion metric (see [2] for more details). In our case, we apply a Gamma distribution to obtain the values inside each given graph. Some parameters are susceptible to change between two different tasks (such as the communication or the computation cost). In this case, their means are prefixed by "M_" and their coefficients of variation are prefixed by "CV_". This last coefficient helps to compute the standard deviation of the given parameter. For instance, in the table we see that M_CommCost is set to 100,000 bytes and CV_Cost is 0.5. This means that on the average the number of bytes to be transmitted is 100,000 and the standard deviation is 50,000 (M_CommCost×CV_Cost). Some parameters are ignored for Strassen graphs: the communication cost (it is already induced by the number of tasks and by the execution cost), the coefficient of variation associated with these 2 costs (CV_Cost is then zero) and the average number of edges per node. Besides, the number of tasks is instead: 23, 163 and 1143. The distributions of the costs in the task graphs follow either a Beta, an exponential, or a normal discretized distribution. The Beta distribution parameters are such that the probability distribution corresponds to our observations and expectations. To 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 and thus $\beta > \alpha$). Therefore, we select $\alpha = 2$ and $\beta = 5$.

Parameter	default { experiment }
GraphType	samepred layrpred strassen
TaskNumber	1000 { 10 100 1000 }
SeedApp	0 { 1 2 3 4 5 6 7 8 9 }
M_ExeCost (FLOP)	100 M { 10 M 1 G }
M_CommCost (B)	100 k { 10 k 1 M }
CV_Cost	0.5 { 0.001 0.1 0.3 1 2 }
Avg Edge/Node	3. { 1. 5. }
Distribution	BETA { EXP NORMAL }
M_UL	1.1 { 1.0001 1.2 1.5 2 3 5 }
CV_UL	0.3 { 0.001 0.1 0.5 1 2 }

TABLE 1
Task graph parameters

The parameters for the platform generation are selected in order to correspond to realistic situations. Hence, we modify the number of processors (ProcessorNumber), the bandwidth and the latency of the links (LinkBandwidth and LinkLatency) and the power of the processors (MachPower). The values are represented in Table 2. CV_Plat is the coefficient of variation associated with MachPower and LinkLatency.

On the scheduling side, random schedules are created by repeating iteratively the following three phases: 1) choose randomly a task among the ready ones, 2) assign it to a randomly selected processor and schedule it eagerly, 3) update the list of ready tasks.

Parameter	default { experiment }
ProcessorNumber	50 { 25 100 }
SeedPlat	0 { 1 2 }
M_MachPower (FLOPS)	2.5 G
M_LinkLatency (ms)	0.1
CV_Plat	0.5 { 0.001 0.1 0.3 1 2 }
M_LinkBandwidth (B/s)	50 M
CV_LinkBandwidth	1 { 0.001 0.1 0.3 0.5 2 }

TABLE 2
Platform parameters

The evaluation of the makespan distribution (needed for most of the metrics presented in Section 4) is realized with a MC (Monte Carlo) method. For each RV (random variable), we use the Mersenne Twister random number generator [25]. We have to define the number of MC realizations, T, required to achieve a meaningful precision. If we assume that the makespan distribution is Gaussian, the Cochran's theorem states that the variance estimator follows a χ^2 distribution with $T - 1$ degrees of freedom. Thus, the number of degrees required to obtain a tight confidence interval gives directly the number of MC realizations. In our case, 20,000 realizations are needed in order to have less than 5% of precision with a confidence level of 99% for both criteria (750,000 realizations would be necessary to have a precision less than 1%, which is too much time-consuming). With these 20,000 realizations we obtain an empirical cumulative distribution function (ECDF).

Many of the metrics proposed in Section 4 are calculated from the makespan distribution and most are straightforward to compute from the ECDF. However, the differential entropy presents some issues. An obvious lower bound is $-\infty$ and an upper bound is proposed by DeStefano, Lu and Learned-Miller in [21]. This last method considers the confidence interval of each point of the ECDF and uses a string-tightening algorithm to interpolate the CDF that maximizes the differential entropy. Although this method is close to the true value for large number of realizations, it is highly time-consuming. Therefore, we chose a method that assumes that each point of our ECDF corresponds precily to the true CDF value. This approximates closely the upper bound described in [21].

For the *probabilistic metric*, we have chosen $\delta = 0.1$ and $\gamma = 1.005$ in order to have values well distributed on the interval $[0; 1]$ (for different ULs, execution or communication costs than the one we used here, these values should be adapted).

To study the problem structure and in addition to robustness metrics, we measure the following additional information:

- **Slack standard deviation**. The measure of the slack is similar to the measure of the makespan since the slack is also a RV. We compute thus its standard deviation.
- **Anderson-Darling statistic**. This test is one of the

best ECDF omnibus tests for normality. It allows to test the normality of the makespan. The returned statistic can be thought of as a distance to a normal.

On the overall we have 150 cases with different graphs type, number of nodes, target platform, uncertainty level, etc. For each generated cases, we build 5,000 random schedules. Each metric is then compared to each other visually and with the statistical Spearman correlation coefficient.

6 Empirical comparison of metrics

6.1 Experimental Results

Among all the graphs we have generated, we have selected three relevant ones that are typical of the general behavior (see Figure 1 and 2). On these figures, 9 metrics are compared to every other ones, leading to a matrix of 81 scattered elements. For clarifying the figures, the *makespan 0.99-quantile* is not present because it is completely equivalent respectively to the *makespan mean*. On the diagonal of the matrix, the name of each metric is given. On the lower part, we plotted the values of each metric for the random schedules. For instance, in Fig. 1 left, we plot the value of the *makespan mean* against the *makespan differential entropy* on the first column and third row (the makespan is plotted on the x-axis and the entropy on the y-axis). In order to ease the reading of the plots, we inverted three metrics such that optimization consists in minimizing these metrics (hence, good results should be plotted in the lower left corner of the corresponding plot). These metrics are the *slack mean*, because our initial assumption is that a robust schedule has high slack, and the two *probabilistic metrics*, since we want to maximize the probability to be in an interval. The inversion is done by multiplying by -1 the measured value that was obtained (for the slack mean) or by subtracting the measured value to 1 (for the probabilistic metrics cases). Other metrics are not inverted because optimizing them consisted already to minimize them (such as the makespan mean). Additionally, cubic smoothing spline fitting were performed on each plot, in order to visualize the correlation. Finally, we have added the metric value for the schedule found by HEFT (red square) to show what happen for a schedule with a very good makespan. The upper part of the matrix contains the value of the Spearman coefficients associated with each plot corresponding to the metrics. The higher the correlation, the closer to 1 is the absolute value of the Spearman coefficient. The minus sign for correlation means that the metrics are negatively correlated (if one metric increases, the other decreases). For instance we see in Fig. 1 left that the makespan mean and the slack mean are negatively correlated by a value of -0.84.

Since the Spearman coefficients show how the metrics are correlated to each other, they are a good way to sum-up our contribution. Hence, we have plotted in Figure 3 the matrix with the Spearman coefficients of 150 different cases. In this figure we have plotted the average value on the upper part of the matrix and the standard deviation on the lower part. We see, for instance, that the slack standard deviation and the makespan entropy metric are highly positively correlated (average Spearman coefficient of 1.00) with a very low standard deviation (0.01).

6.2 Discussion

We see immediately the correlation between a number of robustness metrics, which are the makespan standard deviation, the differential entropy and the absolute probabilistic metric. These relations are common to every generated graph, whatsoever the size, the UL (uncertainty level) or the type of graph. Indeed, the low standard deviations of the Spearman coefficients indicate that the degrees of correlation are almost always the same.

The relations between standard deviation, entropy and absolute probabilistic metrics suggest that the probability density shape remains similar for every schedule. Our explanation is based on the use of the central-limit theorem which states that the sum of random variables having finite mean and variance (as in our case) is approximately normally distributed. Despite the fact that the makespan is obtained by performing a number of operations mixing sums and maximums, the result distribution is close to a Gaussian (however, there is a few cases where the makespan distribution fails the normality test). This hypothesis explains the correlation between these metrics.

Furthermore, the relative probabilistic metric is also correlated to the other ones in the case of *layrpred* graph as can be seen in Fig. 1 left. As shown in Figure 3, the average Spearman coefficient is 0.53 with a standard deviation of 0.29 when compared to the makespan standard deviation. Hence, high correlations for *layrpred* graphs are compensated by the lower correlations on other graphs. Further investigations have shown that this metric is equivalent to the makespan coefficient of variation (CoV) which is the ratio between the standard deviation and makespan mean. Indeed the average Spearman coefficient between the CoV and the relative probabilistic metric is 1.00 with a standard deviation of 0.01. It is thus a redundant metric.

A second observation can be made on the relation between the makespan and the slack. On average, they are conflicting objectives in the sense that optimizing one produces a poor value for the other metric. Intuitively, a schedule having a good makespan has not much unused processor time and a schedule with a lot of slack (or spare time) is inefficient.

It is worth noting, too, that the makespan mean and standard deviation are correlated. Although not excellent and differing to some degree for each graph, there is a noticeable relationship in the general case. Table 3 summarizes the correlation coefficients over every experiment scenario (a value close to 1 implies a high linear relationship between the criteria) for the *rand* schedules.

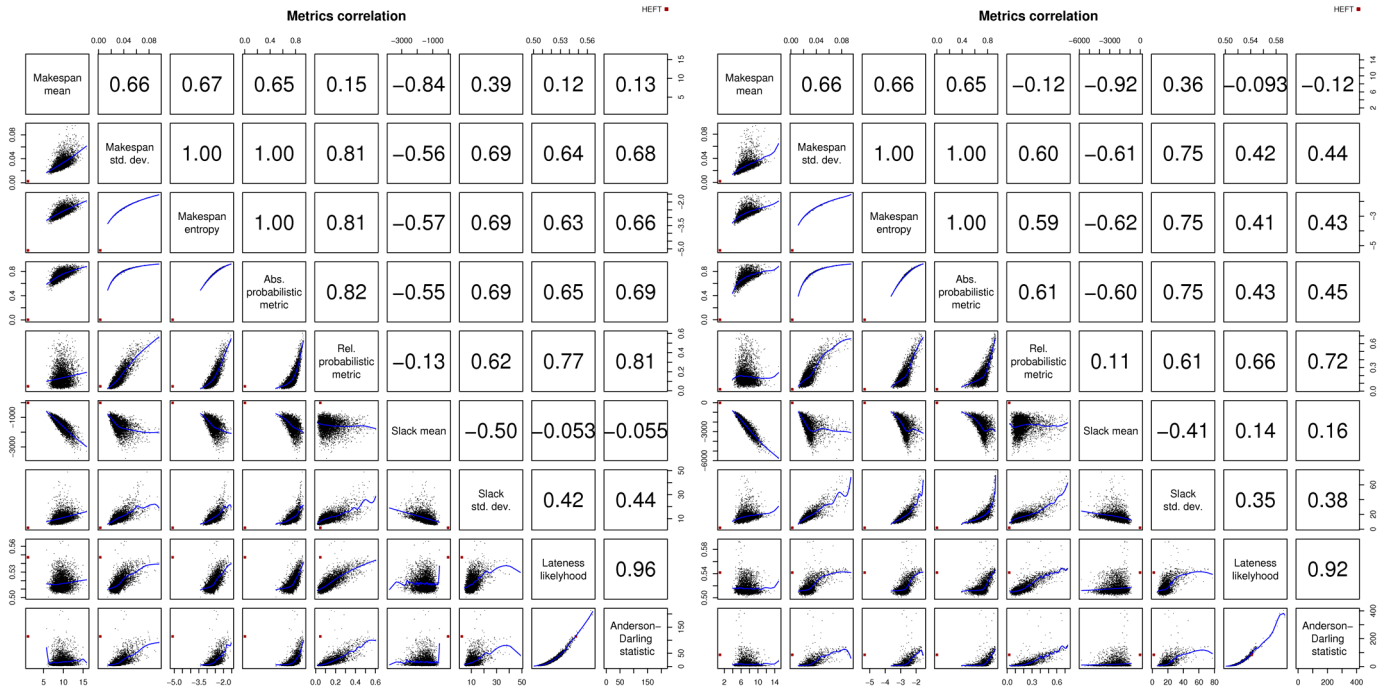

Fig. 1. Metrics correlation for the *layrpred* (left) and *samepred* (right) graphs with default values. Lower part of the matrix: plot for 5,000 random schedules. Upper part of the matrix: value of the Spearman coefficients.

Tukey's five number summary corresponds to minimum, the first quartile, the median, the last quartile and the maximum of the set of measures. We see that 25% of the correlation coefficients of Strassen graphs are lower than 0.78 and 75% are higher. We observe that makespan of Strassen graphs schedules have highly correlated means and standard deviations.

Graph	Min	25%	Med	75%	Max
LAYRPRED	0.27	0.65	0.70	0.79	0.94
SAMEPRED	0.29	0.66	0.70	0.78	0.95
STRASSEN	0.30	0.78	0.79	0.79	0.97

TABLE 3
Tukey's five number summary of correlation coefficients by graph kind

To explain this correlation, we describe one phenomenon that arises when we are evaluating the makespan probability distribution. The variance of a random variable resulting from the sum of two others is the sum of the first two variances. If we do not considerate the implications of the maximum operator, a direct consequence is that the more tasks on the critical path, the more significant is the standard deviation, and hence, the final standard deviation is high. As we modeled the standard deviation to be proportional to the mean of task duration, schedules with low makespan, hence having less tasks or shorter tasks on the critical path, have relatively less standard deviation than schedules with large makespans. The imperfection of the correlation must be due to the maximum operations.

One surprising result is the low correlation that exists between the slack mean and other metrics, and specially the nature of this correlation. Maximizing the slack is indeed a conflicting objective with the robustness. This contradicts the intuition that the more a schedule has slack, the more it is able to absorb uncertainty. Additionally, some previous work also proposed this metric for robustness. Hence, we present some arguments that confirm this result. Fig. 4 exhibits four examples of schedule for a join task graph of $N + 1$ identical tasks having independent and identically distributed (i.i.d.) random variables. Each schedule represents different possibilities with the two objectives being the slack and the makespan standard deviation. The non-robust schedules (c and d) – in the sense of uncertainty absorption – can be interpreted as follow: almost any late task has a repercussion on the overall makespan. The schedule b) has a good robustness because only the three tasks on the critical path have an incidence on the makespan if one of those is late. The schedule a) is more subtle, because it relies on the characteristics of the maximum of two independent random variables being similar. In this case, the resulting mean is greater than the original mean and more importantly, the final standard deviation is lower than at least the maximum of the two originals. A consequence is that the maximum of an infinite number of i.i.d. random variables is equal to a Dirac delta function (which is completely robust) whose value is the maximum possible value of these random variables. Then, the more tasks we are waiting for, the more we are sure that one is late, and the more the schedule is robust because we have more certainty on the expected maximum. With these four examples,

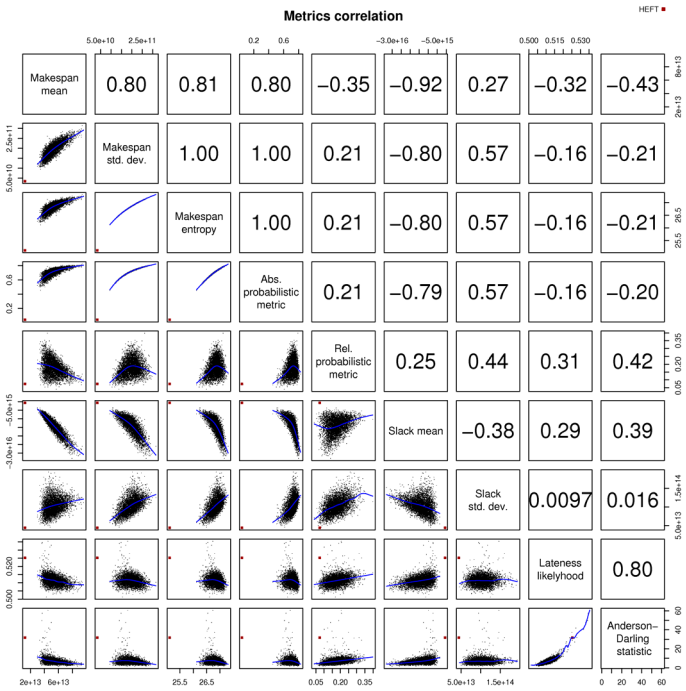

Fig. 2. Metrics correlation for the *strassen* graph with a platform of 25 processors. Lower part of the matrix: plot for the Spearman coefficients for 150 different experiments 5,000 random schedules. Upper part of the matrix: value of the Spearman coefficients.

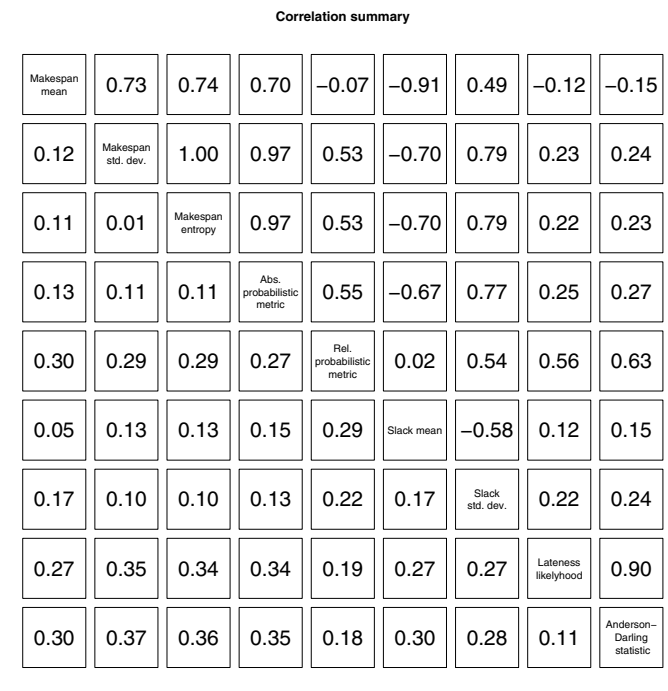

Fig. 3. Average (top) and standard deviation (bottom) of the Spearman coefficients for 150 different experiments

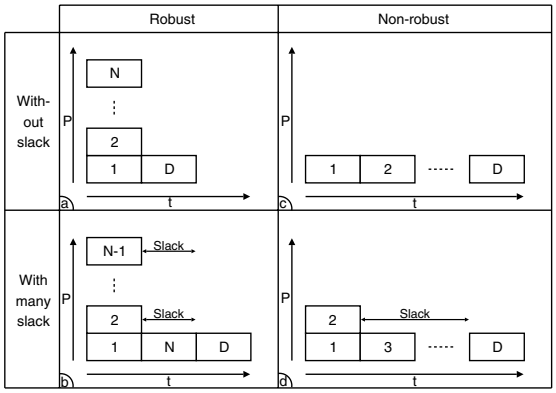

Fig. 4. Four schedules on P processors for different robustness and slackness, considering i.i.d. random variables and a join graph with $N+1$ tasks

we see that the slack is not necessarily related to the robustness. Moreover we see, as we already explained why, that the slack and the makespan are conflicting objectives and schedules with good makespan are often more robust.

The motivation behind the measure of the slack standard deviation is to provide more complete information about the slack distribution. In Figure 3, the mean and standard deviation of the slack have a non-negligible correlation coefficient. Although less evident, the reason is the same as for the makespan. A high correlation exists between the slack standard deviation and other robustness metrics as the makespan standard deviation. This is because the computation of the slack consists of aggregating time intervals between task end and start times, whose standard deviation are strongly related to the makespan standard deviation.

Since the *makespan 0.99-quantile* is completely equivalent (perfect linear correlation) with the makespan mean, it is not represented to keep the figures clear. This equivalence is due to the fact that the variation of the makespan is not sufficient to observe a significant difference between any quantiles of the makespan distribution.

The correlation between the *lateness likelihood* and the Anderson-Darling statistic is explained as follows. The more a distribution is normal the more its skewness is close to 0, the more the Anderson-Darling statistic is close to 0, the more the mean is close to the median and the more the lateness likelihood is close to its minimum value (0.5). Hence, the lateness likelihood is mostly an other way to measure normality of a distribution.

Part of the above study is based on the hypothesis that the obtained makespan distributions are normal (follow a Gaussian distribution). To validate this hypothesis, we have conducted normality tests on our schedules to study at which point the normality assumption holds. The histogram of every Anderson-Darling normality statistics for the makespan distribution of random schedules is depicted in Figure 5. Half of the distributions have a statistic lower than 12.0 (the same as a Student t distribution with 11.5 degrees of freedom, which is

visually quite similar to a normal, see Figure 6), and 90% of these have less than 79.5 and are more closer to a normal than a Weibull with parameter $\lambda = 1$ and $k = 2.17$ (Weibull are considered similar to normal for $k = 3.4$). Although it is not perfect, it is sufficient to validate our normality assumption (in the approximation scheme, for the confidence intervals and for the reduction of the robustness metrics to the standard deviation). We also notice that the choice of the distribution influences strongly the normality of the makespan distribution (it is the worst when cost distributions are exponentials).

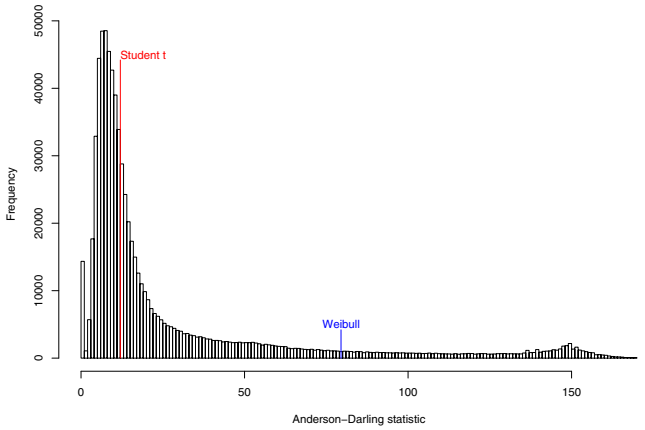

Fig. 5. Histogram of Anderson-Darling statistics for the makespan distribution of random schedules

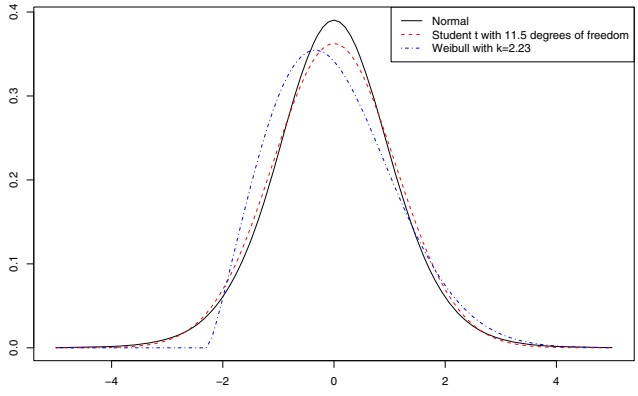

Fig. 6. Three probability density functions with identical mean and standard deviation

Finally, concerning the HEFT schedules (red square in Figure 1 and 2), we see that the obtained results corroborate our observations. When there is a high correlation between two metrics, the HEFT point is in the continuation of the fitting curve (as for the *entropy* vs. *abs. prob. metric* in Fig 2), whereas when the correlation is close to zero, the point is more or less put randomly in the plot (as for the *slack std. dev.* vs. *lateness likelihood*).

6.3 Conclusion

Based on the above observations we draw the following conclusions. First, standard deviation, entropy and absolute probabilistic metrics are highly correlated. Second and counter-intuitively, the slack is not a good way to measure the robustness (as shown in Fig. 4 there exists robust schedules with low slacks and non-robust schedules with large slacks).

Moreover, the standard deviation is very easy to compute when the makespan distribution is approximated by a normal distribution. This approximation is justified by the normality tests that were conducted. Hence, among all the studied metrics, the standard deviation is the most representative and easy-to-compute metric for robustness.

The idea behind the relative probabilistic metric is to tolerate greater uncertainties for long schedules than for short ones. If this notion of robustness is preferred, computing the mean and standard deviation are sufficient because the relative probabilistic metric can be replaced by the CoV (ratio between the standard deviation and the mean).

7 A PROVABLY CONVERGENT MOEA

In the previous sections we have shown that the *makespan standard deviation* is a representative robustness metric as it is highly correlated to many other metrics. Moreover, whatever the method used to compute the makespan distribution, this metric is the easiest to compute. In this second part of the article, we use our understanding of robustness to design bicriteria scheduling strategies that targets optimization of the schedule length and the robustness at the same time.

In a bicriteria problem we need to find the Pareto front, which is the set of non-dominated solutions of the search space (called Pareto-optimal solutions). A solution is said to be non-dominated with respect to a set of solutions if any other solution of this set is worst for at least one criteria. Then, more than one solution can be optimal because we are dealing with a partially ordered set of solutions (two solutions are incomparable if one is better than the other for the first criterion and worse for the second).

In this section, we describe the MOEA (multi-objective evolutionary algorithm) implementation we use and guarantee its convergence.

7.1 MOEA implementation

Several successful modern MOEA exists such as NSGA-II [10] and IBEA [34], just to mention a few. NSGA-II is the reference metaheuristic in the area and performs similarly to IBEA for combinatorial problems. Thus, our work is based on the NSGA-II algorithm as it is implemented in the ParadisEO framework [6].

While the MOEA takes care of the multi-objective aspect and selection phase, we use the chromosome

representation and the crossover and mutation schemes described by Wang et al. in [33]. Algorithm 1 describes our MOEA, which is a modified version of NSGA-II. Lines 1 and 2 we combine parents $P(t)$ and offspring $Q(t)$ to obtain the current population $R(t)$. This current population is decomposed in a set of front $\mathcal{F}$ such that any solution in front $\mathcal{F}_{i+1}$ is dominated by at least one solution in the front $\mathcal{F}_i$. Lines 3 to 8, we update the set S. This set contains the non-dominated solutions and is an added feature compared to the original version of NSGA-II. Initially, S is empty. In these lines we ensure that a Pareto-optimal solution is inserted in S and stays there until a new solution that dominates it is inserted. It can be seen as an elitism mechanism. Lines 9 to 13 is a randomized way to choose the set of parents $P(t+1)$ that will be used to generate the new offspring population $Q(t+1)$. As we do not want the population to grow indefinitely, we have to sample the current population. We use a parameter l (set to 1 in our implementation) to denote the importance of the first fronts. The larger l, the greater the probability of an element from the first fronts to be chosen as a member of $P(t+1)$ (as being done by NSGA-II). Conversely, when $l = 0$ each front has the same importance. The crowding-distance is used to compare two solutions of a given front $\mathcal{F}_i$ and determine the ones who are in the least dense portion of the solution space (or the ones who are the farthest from their neighbors) and hence should be kept for the next step. On a given front, solutions with largest crowding distance are considered first and have a greater probability to be chosen next (line 13). The probability $p_{i,j} > 0$ is the probability that solution j of front i is selected. As this probability is always strictly positive, any solution can stay in the next generation. According to this probability, the sampling is performed line 14. Finally the next generation $Q(t+1)$ is computed line 15 : the `make_new_pop` function has the following characteristic: non-deterministic selection and probability of crossover strictly lower than one (any solution can be left for the next step).

Concerning the evaluation of a solution (the fitness functions that computes the considered metrics: mean and standard deviation), we proceed as follows. We recall that, as stated in Section 2, the evaluation of the makespan distribution is #P-complete and thus an accurate evaluation is too much time-consuming. Achieving a correct precision with Monte Carlo (MC) evaluation requires a lot of computation time. We have thus opted for the last approximation scheme described in [7]: we assume that all the distributions are Gaussian and we compute the final makespan distribution by determining correlation between reconvergent paths. This allows fast evaluation with a correct precision in most cases.

Many sophisticated methods exist to deal with such fitness approximation in order to improve the effectiveness of the MOEA (see the survey of Jin and Branke [19]). However, we have implemented a simple archive mechanism where the three bests non-dominated fronts are always kept. Hence, we end up with more optimized solutions with respect to the fitness approximation and this increases the probability of obtaining optimal ones.

7.2 Convergence conditions

Rudolph defines the conditions for a multi-objective EA to converge (Definition 3 of [26]). Let A be the set of all valid solutions. Let f be the objective function, while $\mathcal{F}$ is the set of objective values ($f : A \mapsto \mathcal{F}$). Let A^* be the set of Pareto-optimal solutions. We define $A(t)$, the population at step t of the EA. Finally, let $\delta_X(Y)$ be the number of elements in set Y but not in set X. The EA is said to converge with probability 1 to a set of Pareto-optimal solutions if $\delta_{A^*}(A(t)) \to 0$ as $t \to \infty$

We give sufficient conditions for our MOEA to converge under the above definition. In order to achieve this goal, we extend the existing theory to local mutation operators having some properties we detail below.

To prove that our EA converges, we use the following proposition (that shows that all the solutions on the S set tends to be composed of Pareto-optimal solutions only, as t grows):

Proposition 1: Let $P(t)$ be the parent population of size N at step t. Let $M > 0$ be an integer. Let G be the homogeneous stochastic matrix[6] describing the transition behavior from $P(t)$ to $Q(t+M-1)$ (the offspring population after M generations). Let S be the set of non-dominated solutions as computed in algorithm 1. If matrix G is positive then $\delta_{A^*}(S(t)) \to 0$ and $|S(t)| \to \min\{N, |A^*|\}$ with probability one and in mean as $t \to \infty$.

Proof: The proof is inspired from Proposition 4 of [26]. If at some step t, $S(t)$ contains a non-optimal solution s, then as G is positive, the EA generates in finite time, with probability one, a Pareto-optimal solution r that dominates s. Indeed, G positive implies that any solution can be generated from any one. Now, lines 3 to 8 of our EA ensures that r replaces s in S. Moreover the same lines of the algorithms ensures that a Pareto-optimal solution of S stays in S forever. □

This means that our MOEA converges (*i.e.*, the set of non-dominated solutions S tends to be composed of Pareto-optimal solutions only as $t \to \infty$ and is the largest possible) if the transition matrix G is positive (*i.e*, all its elements are strictly positive).

We now have to show that $\exists M > 0$, the transition matrix G between $P(t)$ and $Q(t+M-1)$ is positive. We use the mutation operator proposed by Wang et al. in [33]. This operator is local, that is to say, it can generate only local neighbors from a given solution. However, we prove that M such mutations are equivalent to a global mutation operator (any solution can be generated from any other one), which is established by Lemma 1. Hence, after M steps, we are able to generate any solutions.

6. The dimension of G is $\binom{\eta}{N}$ where η is the number of feasible schedules.

Algorithm 1 Modified version of the NSGA-II main loop

1	$R(t) = P(t) \cup Q(t)$	combine parent and offspring populations
2	$\mathcal{F} = \texttt{fast-non-dominated-sort}(R(t))$	$\mathcal{F} = (\mathcal{F}_1, \mathcal{F}_2, \dots)$ all dominated fronts of $R(t)$
3	foreach $b \in \mathcal{F}_1$	for each non-dominated solution b of step t
4	$D_b = \{s \in S(t) : f(b) \prec f(s)\}$	D_b: set of solutions of S dominated by b
5	$N_b = \{s \in S(t) : f(s) \prec f(b)\}$	N_b: set of solutions of S that dominates b
6	if $D_b \neq \emptyset$ then $S(t) = S(t) \setminus D_b$	remove dominated solutions from s
7	if $N_b = \emptyset \wedge \lvert S(t)\rvert < N$ then $S(t) = S(t) \cup \{b\}$	add b to S (guarantee that $\lvert S\rvert \leq N$)
8	$S(t+1) = S(t)$	
9	foreach $\mathcal{F}_i \in \mathcal{F}$	
10	$p_i = \frac{1}{i^l \times \sum_{k=1}^{\lvert\mathcal{F}\rvert} \frac{1}{k^l}}$	compute the probability of selecting the front $\mathcal{F}_i$
11	$\texttt{crowding-distance-assignment}(\mathcal{F}_i)$	calculate crowding-distance $c_{i,j}$ of each solution j from the front $\mathcal{F}_i$
12	foreach $s_j \in \mathcal{F}_i$	for each solution s_j of front $\mathcal{F}_i$ sorted descendingly by their $c_{i,j}$
13	$p_{i,j} = p_i \times \frac{1}{j^l \times \sum_{k=1}^{\lvert\mathcal{F}_i\rvert} \frac{1}{k^l}}$	compute the probability of selecting solution j from the front $\mathcal{F}_i$
14	$P(t+1) = \texttt{draw}(N, R(t), \{p_{i,j}\})$	select N distinct elements from $R(t)$ drawn accordingly to the probabilities $p_{i,j}$
15	$Q(t+1) = \texttt{make-new-pop}(P(t+1))$	use selection, crossover and mutation to create a new population $Q(t+1)$
16	$t = t+1$	increment the generation counter

Lemma 1: The local mutation operator proposed by Wang et al. in [33] is equivalent to a global mutation operator if applied $M = \max(n, \text{led}(E)) \leq \max(n, \frac{n(n-1)}{2})$ times consecutively, where n is the number of tasks and led(E) is linear extension diameter [15] of the input DAG.

Proof: The chromosome representation consists of 2 strings: the matching string that describes the processor where each task is mapped and the scheduling string that defines the execution order of the tasks.

The number of consecutive mutation needed in order to obtain any matching string is obtained as follows. Each time a task is selected, it is mapped to a new processor chosen randomly for it without any other constraint. Thus, the probability of getting a given assignment string from an initial one with n mutation iterations is lower bounded by $\delta_{m_1} = \left(\frac{p_m}{nP}\right)^n n! > 0$, where n is the number of tasks, P the number of processor and $p_m > 0$ the probability that the mutation happens.

The schedule string is a linear extension of the partial ordered set (poset) E obtained from the task graph G and any of its local modifications should respect the order (corresponding to the precedence constraint). The maximum number of permutations needed to obtain any linear extension from any other is called the linear extension diameter led(E) and it is shown by Felsner and Reuter [15] that this diameter is upper bounded by Inc(E), which is the number of pairs of incomparable elements. Hence $\text{led}(E) \leq \frac{n(n-1)}{2}$. The probability of getting a given linear extension after applying led(E) mutation iterations is thus lower bounded by $\delta_{m_2} = \left(\frac{p_m}{n^2}\right)^{\text{led}(E)} > 0$.

Then, after $M = \max(n, \text{led}(E)) \leq \max(n, \frac{n(n-1)}{2})$ mutation iterations, the resulting probability of obtaining any schedule from any given one is lower bounded by $\delta_m = \delta_{m_1}\delta_{m_2} > 0$. Thus, We can obtain a global mutation by successively applying local mutations. □

The above lemma assumes that mutations are performed consecutively. However, in our EA, mutation happens between crossover and selection operations. Lemma 2, given bellow, is necessary to prove that having M consecutive full steps (crossover, mutation and selection) has no impact on the convergence of the EA under the assumptions that any solutions might be left unchanged by the crossover operator or kept in the solution space after the selection process.

Let us first introduce some notations and definitions. Let H be the set of possible states of a stochastic process and X_t be a state of the process at step t. A Markovian kernel K is defined as $K(x, H) = \Pr[X_{t+1} \in H \mid X_t = x]$, namely the probability that the state of the stochastic process is in H at step $t+1$ when its state is x at step t. The product kernel (K_c, K_m, K_s) represents the Markovian kernel of the entire EA process, K_c being the kernel of the crossover operator, K_m for the mutation operator and K_s for the selection. Moreover, $K^{(M)}$ is the M-th iteration of the kernel K.

Lemma 2: $\forall H \subset A$ and $\forall y \in H$. Let $K_c(y, H) \geq \delta_c$ and $K_s(y, H) \geq \delta_s$

Then, for each $x \in A$,

$$\left(K_c\, K_m\, K_s\right)^{(M)}(x, H) \geq \left(\delta_c \delta_s\right)^M K_m^{(M)}(x, H)$$

Proof: (by induction) Let $K_c(x, H) \geq \delta_c 1_H(x)$ and $K_s(x, H) \geq \delta_s 1_H(x)$ for each $x \in A$ and for each $H \subset A$ and where $1_H(x)$ denotes the indicator function for some set H ($1_H(x) = 1$ if $x \in H$, 0 otherwise). We obtain the basis of the induction for $M = 1$ by developing $\left(K_c\, K_m\, K_s\right)(x, H)$,

$$\begin{aligned}
\left(K_c\, K_m\, K_s\right)(x, H) &= \int_A \left(\int_A K_c(x, dz) K_m(z, dy)\right) K_s(y, H) \\
&\geq \int_A \left(\int_A \delta_c 1_{dz}(x) K_m(z, dy)\right) \delta_s 1_H(y) \\
&\geq \delta_c \delta_s \int_H \left(\int_A 1_{dz}(x) K_m(z, dy)\right) \\
&\geq \delta_c \delta_s \int_H K_m(x, dy) \\
&\geq \delta_c \delta_s K_m(x, H) \qquad (1)
\end{aligned}$$

Now assume that the hypothesis is true for $M > 1$.

Equation 1 induces that

$$\begin{aligned}(K_c \quad & K_m\, K_s)^{(M+1)}(x,H)\\ = \quad & \int_A (K_c\, K_m\, K_s)^{(M)}(y,H)\,(K_c\, K_m\, K_s)(x,dy)\\ \geq \quad & \delta_c\delta_s \int_A (K_c\, K_m\, K_s)^{(M)}(y,H) K_m(x,dy)\end{aligned}$$

By induction hypothesis, we have $\int_A (K_c\, K_m\, K_s)^{(M)}(y,H)K_m(x,dy) \geq \int_A (\delta_c\delta_s)^M K_m^{(M)}(y,H)K_m(x,dy)$ and by definition, $K_m^{(M+1)}(x,H) = \int_A K_m^{(M)}(y,H)K_m(x,dy)$ Consequently, the hypothesis is true for $M \geq 1$. □

We can finally prove the main result of this section:

Theorem 1: Algorithm 1 converges in finite time with probability 1.

Proof: Proposition 1 shows that we have to prove that matrix G is positive, where G defines the transition probability of the population between step t and step $t + M - 1$ ($M > 0$). Lemma 1 shows that, using the Markovian kernel formalism, $K_m^{(M)}(i,\{j\}) > 0$ for any state i and j and for $M \geq \max(n, \text{led}(E))$. Let $g_{i,j} \in G$ be the transition probability between state i and state j when the full stochastic process is applied M times. By definition, $g_{i,j} = (K_c\, K_m\, K_s)^{(M)}(i,\{j\})$. Lemma 2 shows $g_{i,j} \geq (\delta_c\delta_s)^M K_m^{(M)}(i,\{j\}) > 0$ provided that δ_c and δ_s are not equal to zero. Since the algorithm ensures that the crossover probability is strictly lower than 1 (hence $\delta_c > 0$) and that selection phases are non-deterministic – see Section 7.1 – (hence $\delta_s > 0$). □

8 Heuristics

We introduce in this section a heuristic based on HEFT [31] able to generate a set of solutions intended to have good performance for both criteria from a stochastic task graph.

The idea of this heuristic is the following. The user provides an angle value a between $0°$ and $90°$. At each step of the heuristic, we consider a new task for scheduling (the order in which tasks are selected will be given later). We simulate the execution of this task on each of the p processors and compute the average makespan and the standard deviation of the new schedule (up to this task). This leads to p points in the criteria space (makespan,standard deviation). We then remove the points that are dominated to retain $p' \leq p$ points. To give the same importance to all the criterion, we proceed to a rescaling and change of coordinate for the points to exactly fit in the square of size 1. Then, according to the selected angle value, we select the point (and hence map the task to the corresponding processor) that is the closest to the line that starts from the origin and make an angle a with the x-axis (see Fig. 7). The intuition beyond this heuristic is to choose the processor that leads to the desired trade-off between makespan and standard deviation. When $a = 0$ the task is mapped to the processor that minimize the standard deviation, when $a = 90$ the task is mapped to the processor that minimize the makespan.

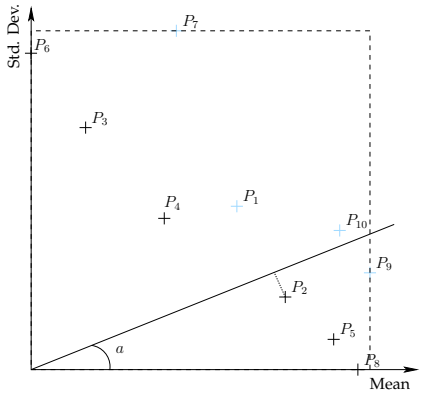

Fig. 7. Example of the Rob-HEFT heuristic with $p = 10$. Among the 10 generated solutions, $p' = 6$ are non-dominated. The square of size 1 is shown in dash lines. P_2 is selected as the solution closest to the line that makes an angle a with the x-axis and that passes through the origin. P_{10} is not selected as it is dominated by P_2 and P_5.

To rank the task in the order that they are selected we compute for each task their bottom level (average and standard deviation value). Then, based on a, we aggregate (after normalization) the average and standard deviation to obtain the rank. If $a = 0$ we only take the average and if $a = 90$ we only take the standard deviation. Tasks are then considered by the smallest rank first.

This heuristic is called Rob-HEFT (HEFT with robustness) when we use the same approximation scheme for computing the makespan distribution as for the MOEA. We provide another version called Rob-HEFT-MC when the evaluation of the makespan distribution is done by the Monte-Carlo method.

Another angle-based geometrical selection of a solution in a bi-criteria space has been proposed by Assayad et al. in [3]. However, the difference with our approach is that the selected solution is the one for which the projection on the line is the smallest. The drawback of this approach is that some solutions can never be selected even if they are not dominated (*i.e.* only solutions on the convex hull can be selected).

9 Experimental comparison of strategies

9.1 Setup

As mentioned in Section 6.2, we perform the robustness evaluation by calculating the standard deviation of the makespan.

We use the same experimental setup as the one described in Section 5 for evaluating the robustness metrics.

Concerning the MOEA setup, we consider populations of 200 chromosomes over 1,000 generations. The crossover and mutation probabilities are respectively

0.25 and 0.35. For *Rob-HEFT* and *Rob-HEFT-MC* heuristics, we vary the parameter a from 0 to 90 by step of 0.45. Varying a helps in finding a non-dominated front.

9.2 Search space

In a first attempt to study the problem specificity, we characterize the search space by generating extreme schedules on the border fronts denoted by *SW*, *SE* and *NW* (obtained with the MOEA, when the objectives are alternatively maximized and minimized). These 3 last metaheuristics are named after the intercardinal directions (*NW* –North West– consists in minimizing the average makespan while maximizing the standard deviation). *SW* is thus designed to find optimal solutions for both criteria. Figure 8 depicts the search space of one experiment scenario. Additionally, the previous random schedules are also represented.

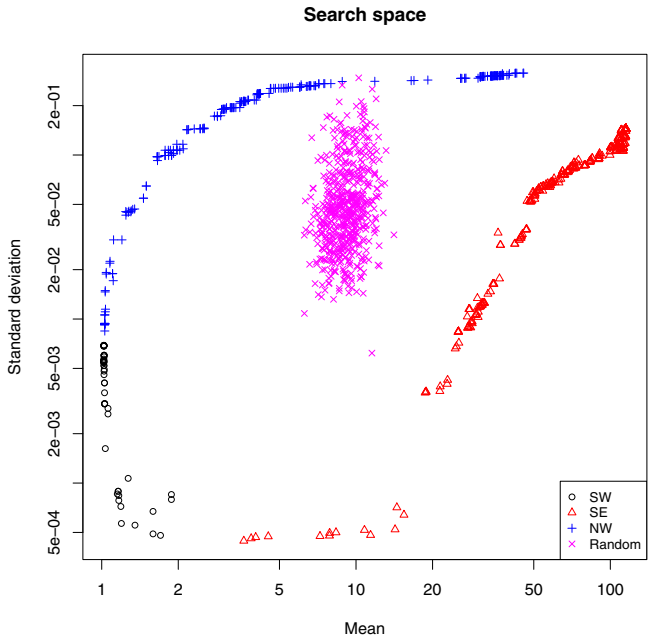

Fig. 8. Mean vs. std. dev. of the makespan of different schedules in random and border cases, with a layrpred graph with TaskNumber $= 1000$ and CV_UL $= 2$

It is worth noting that the *SW* front has a limited spread. Pareto-optimal solutions should hence be close to each other in regards to the entire search space. When CV_UL > 0.3, correlations are however the worst and *SW* fronts are larger. Therefore, minimizing the average makespan does not necessarily induce good robustness when CV_UL takes high values, *i.e.* when the uncertainty range of the task and communication durations is subject to large variations.

9.3 Average quality

The purpose of this section is to assess the quality of the schedules generated by the strategies presented in Section 7 and 8 in term of robustness and average performance. Figure 9 is a representative example regarding the position of the approximation sets (or non-dominated fronts) of each strategy. Error bars denote the confidence intervals of each schedule's evaluated objectives.

Assessing the quality of an approximation set has been shown to be extremely difficult as several criteria can be measured for a given set (spread of the front, optimality, ...). Among the numerous existing quality indicators, we have chosen the binary ϵ-indicator which is presented and recommended by Zitzler *et al.* [36]. Let A and B be two set of non-dominated solutions, then the value $I_\epsilon(A, B)$ corresponds to the ratio between a chosen solution of A and one of B for a selected criterion. To be precise, $I_\epsilon(A, B)$ equals the minimum factor ϵ such that all solutions of B whose criteria have been multiplied by ϵ are dominated by at least one solution of A. It is roughly related to the relative distance between two non-dominated fronts and if we have $I_\epsilon(A, B) \leq 1$ and $I_\epsilon(B, A) > 1$, then the approximation set A is better than B. In other cases, fronts are incomparable.

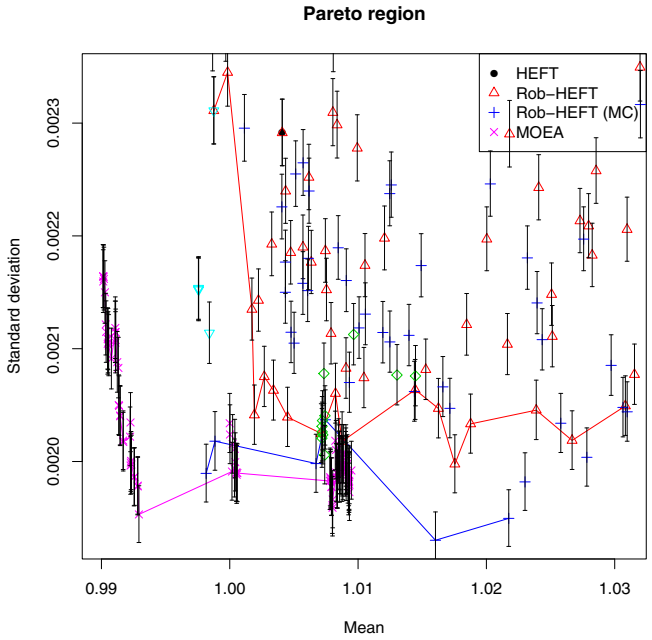

Fig. 9. Mean vs. std. dev. of the makespan of different schedules in the Pareto region, with a layrpred graph and TaskNumber $= 1000$. Lines join non-dominated solutions of each front. Dominance is considered based on the confidence intervals, namely a solution is dominated if its best-case objectives are worse than the worst-case objectives of another solution. Hence, line slopes can be positive.

For instance, in Figure 9, we have $I_\epsilon(\text{MOEA}, \text{Rob-HEFT}) = 0.991$ and $I_\epsilon(\text{Rob-HEFT}, \text{MOEA}) = 1.024$. This means that MOEA is better than *Rob-HEFT*.

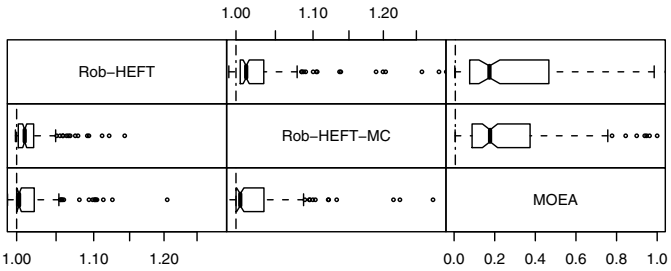

Fig. 10. Comparison of the non-dominated fronts using the ϵ indicator: boxplot of the indicators over every experiment

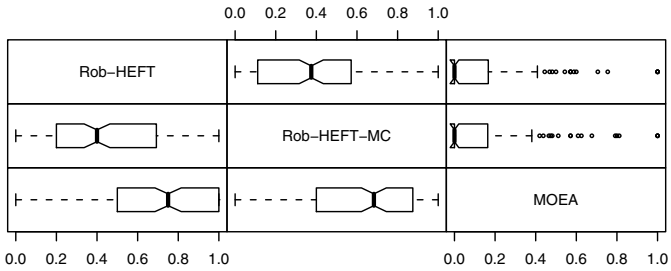

Fig. 11. Comparison of the non-dominated fronts using the coverage indicator: boxplot of the indicators over every experiment

Since the ϵ-indicator is a ratio, it allows doing comparison on every experiment scenario. Figure 10 represents the summary of every computed indicator $I_\epsilon(A, B)$ in the form of boxplots (A for the fronts on the rows and B for the fronts on the columns). Boxplots allow to represent a five number summary of a given set (the extreme of the lower whisker, the first quartile, the median, the third quartile and the extreme of the upper whisker) and the outliers that exceed these values. For example, on the line *Rob-HEFT* and column *MOEA*, these 5 values are respectively 1.00, 1.01, 1.04, 1.12 and 1.28. Results show that the MOEA, *Rob-HEFT* and *Rob-HEFT-MC* are mostly incomparable. We observe that the ϵ-indicators between *Rob-HEFT* and *Rob-HEFT-MC* are very close to 1, which means that both fronts are generally very close.

As it is difficult from the ϵ-indicator to clearly compare the MOEA with *Rob-HEFT* or *Rob-HEFT-MC*, we use another metric for comparing the fronts. This metric, called the coverage indicator [35] $I_C(A, B)$, counts the fraction of solutions in B that are weakly dominated by at least on solution in A. The closest this indicator to one, the better the front A compared to B.

In Fig. 11, we plot the boxplot of coverage indicator $I_C(A, B)$ (A for the fronts on the rows and B for the fronts on the columns). From this boxplot we see that the MOEA outperforms both *Rob-HEFT* and *Rob-HEFT-MC*. For instance, in half of the cases, 68% of the solutions found by *Rob-HEFT-MC* are dominated by the MOEA, whereas in 75% of the cases, only 16% of the solutions found by MOEA are dominated by *Rob-HEFT-MC*. We also see that *Rob-HEFT-MC* is slightly better than Rob-HEFT on that metric.

9.4 Computation time

We measured the average runtime of every heuristic over 6 runs with random graphs (layrpred graphs having different number of tasks). These measures are represented on Table 4. Times do not include MC evaluations of the schedules (except for *Rob-HEFT-MC* because it is intrinsic).

Task number	10	100	1000
HEFT	0.2″	2″	23″
Rob-HEFT	0.3″	12″	4′44″
Rob-HEFT-MC	30″	30′37″	6h00
MOEA	17′34″	43′57″	7h39

TABLE 4
Execution time of every strategy

10 Conclusion

In this paper, we have studied the problem of scheduling stochastic task graphs with the goal of maximizing the robustness and minimizing the makespan on a heterogeneous environment.

We have experimentally compared the robustness metrics on a set of different task graphs. This empirical study was intended to determine the relationship between these metrics. The first conclusion we can draw is that several of them are equivalent mostly due to the implication of the central limit theorem. Consequently, the simplest of these metrics, certainly the standard deviation, is sufficient in most real cases and denotes the absolute dispersion of the makespan, its entropy, etc. (which are all related).

Despite a fairly high correlation value between the makespan mean and standard deviation, a set of Pareto-optimal solutions exists most of the time.

Based on our understanding of the robustness metrics, we have proposed different strategies: two heuristics and a multi-objective evolutionary algorithm (MOEA). The goal of having different strategies is to tackle the trade-off between computation time needed to generate the solutions and their qualities. Concerning our MOEA, we have proved its convergence by extending previous results on the global nature of the mutation operator.

We have then conducted an experimental study of our heuristics for the scheduling problem. The slowest strategy is our evolutionary algorithm. The proposed *Rob-HEFT-MC* (an extension of the makespan-centric heuristic HEFT) is slightly faster but often provides worse solutions than our MOEA. *Rob-HEFT* is a faster heuristic and gives similar results to *Rob-HEFT-MC* for both criteria. All these strategies are able to give an approximation of the set of Pareto-optimal solutions. Therefore, we are able to help the user in choosing another trade-off between makespan and robustness when necessary.

In future work, we need a better evaluation mecha-

nism for our MOEA in order to systematically outperform the proposed heuristics for the robustness.

Moreover, the proposed scheduling heuristics are all static ones. This means that the scheduling decisions are made prior to the execution. Coupling static solution with dynamic decision in order to adapt to the change would be very interesting.

11 Acknowledgments

We sincerely thank the reviewers for their careful reading and remarks and especially reviewer 2 for his detailed comments.

References

[1] Shoukat Ali, Anthony A. Maciejewski, Howard Jay Siegel, and Jong-Kook Kim. Measuring the Robustness of a resource Allocation. *IEEE Transaction on Parallel and Distributed Systems*, 15(7):630–641, July 2004.

[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] Ismail Assayad, Alain Girault, and Hamoudi Kalla. A Bi-Criteria Scheduling Heuristic for Distributed Embedded Systems under Reliability and Real-Time Constraints. In *International Conference on Dependable Systems and Networks*, pages 347 – 356, Florence, Italy, June 2004.

[4] Charles H. Bennett and John Gill. Relative to a Random Oracle A, $\mathbf{P}^A \neq \mathbf{NP}^A \neq$ co-$\mathbf{NP}^A$ with Probability 1. *SIAM Journal on Computing*, 10(1):96–113, 1981.

[5] Ladislau Bölöni and Dan C. Marinescu. Robust scheduling of metaprograms. *Journal of Scheduling*, 5(5):395–412, September 2002.

[6] Sébastien Cahon, Nordine Melab, and El-Ghazali Talbi. ParadisEO: A Framework for the Reusable Design of Parallel and Distributed Metaheuristics. *Journal of Heuristics*, 10(3):357–380, 2004.

[7] Louis-Claude Canon and Emmanuel Jeannot. Precise Evaluation of the Efficiency and the Robustness of Stochastic DAG Schedules. Research Report 6895, INRIA, April 2009.

[8] Charles E. Clark. The greatest of a finite set of random variables. *Operations Research*, 9(2):145–162, March/April 1961.

[9] Andrew J. Davenport, Christophe Gefflot, and J. Christopher Beck. Slack-based Techniques for Robust Schedules. In *Proceedings of the Sixth European Conference on Planning (ECP-2001)*, pages 7–18, Toledo, Spain, September 2001.

[10] Kalyanmoy Deb, Amrit Pratab, Samir Agrawal, and T. Meyarivan. A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, April 2002.

[11] Didier Dubois and Henri Prade. *Possibility Theory: An Approach to Computerized Processing of Uncertainty*. Plenum Press, New York, 1988.

[12] Hesham El-Rewini, Theodore G. Lewis, and Hesham H. Ali. *Task Scheduling in Parallel and Distributed Systems*. Prentice Hall, 1994.

[13] Darin England, Jon Weissman, and Jayashree Sadagopan. A New Metric for Robustness with Application to Job Scheduling. In *14th IEEE International Symposium on High Performance Distributed Computing (HPDC-14)*, pages 135–143, July 2005.

[14] Helene Fargier, Philippe Fortemps, and Didier Dubois. Fuzzy scheduling: Modelling flexible constraints vs. coping with incomplete knowledge. *European Journal of Operational Research*, 147(2):231–252, 2003.

[15] Stefan Felsner and Klaus Reuter. The Linear Extension Diameter of a Poset. *SIAM Journal on Discrete Mathematics*, 12(3):360–373, 1999.

[16] Apostolos Gerasoulis, Jia Jiao, and Tao Yang. Experience with Graph Scheduling for Mapping Irregular Scientific Computation. In *First IPPS workshop on Solving Irregular Problems on Distributed Memory Machines*, pages 1–8, Santa Barbara, CA, USA, April 1995.

[17] Apostolos Gerasoulis and Tao Yang. A Comparison of Clustering Heuristics for Scheduling Directed Acycle Graphs on Multiprocessors. *Journal of Parallel Distributed Computing*, 16(4):276–291, 1992.

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

[19] Yaochu Jin and Jürgen Branke. Evolutionary Optimization in Uncertain Environments–A Survey. *IEEE Transactions on Evolutionary Computation*, 9(3):303–3017, June 2005.

[20] Seung-Jean Kim, Stephen P. Boyd, Sunghee Yun, Dinesh D. Patil, and Mark A. Horowitz. A Heuristic for Optimizing Stochastic Activity Networks with Applications to Statistical Digital Circuit Sizing. *Optimization and Engineering*, 8(4):397–430, December 2007.

[21] Erik Learned-Miller and Joseph DeStefano. A probabilstic upper bound on differential entropy. *IEEE Transactions on Information Theory*, Under revision, 2007.

[22] Joseph Y-T. Leung, editor. *Handbook of Scheduling*. Chapman & Hall/CCR, 2004.

[23] J. W. S. Liu and C. L. Liu. Bounds on scheduling algorithms for heterogeneous computing systems. In *Proceedings of IFIP Congress 74*, pages 349–353, 1974.

[24] Arfst Ludwig, Rolf H. Mohring, and Frederik Stork. A Computational Study on Bounding the Makespan Distribution in Stochastic Project Networks. *Annals of Operations Research*, 102(1–4):49–64, February 2001.

[25] Makoto Matsumoto and Takuji Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, 1998.

[26] Günter Rudolph and Alexandru Agapie. Convergence Properties of Some Multi-Objective Evolutionary Algorithms. In *Congress on Evolutionary Computation*, pages 1010–1016, La Jolla, California, USA, July 2000.

[27] Johnatan Eliabeth Pecero Sanchez. *Local-Global scheduling interactions*. PhD thesis, Institut National Polytechnique de Grenoble, 2008.

[28] Vladimir Shestak, Jay Smith, Howard Jay Siegel, and Anthony A. Maciejewski. A Stochastic Approach to Measuring the Robustness of Resource Allocations in Distributed Systems. In *Proceedings of the 2006 International Conference on Parallel Processing (ICPP'06)*, pages 459–470, Columbus, Ohio, USA, August 2006.

[29] Zhiao Shi, Emmanuel Jeannot, and Jack J. Dongarra. Robust Task Scheduling in Non-Deterministic Heterogeneous Computing Systems. In *Proceedings of IEEE International Conference on Cluster Computing*, pages 1–10, Barcelona, Spain, September 2006. IEEE.

[30] 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.

[31] Haluk Topcuoglu, Salim Hariri, and Min-You Wu. Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing. *Transactions on Parallel and Distributed Systems*, 13(3):260–274, March 2002.

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

[33] Lee Wang, Howard Jay Siegel, Vwani R. Roychowdhury, and Anthony A. Maciejewski. Task Matching and Scheduling in Heterogeneous Computing Environments Using a Genetic-Algorithm-Based Approach. *Journal of Parallel and Distributed Computing*, 47(1):8–22, November 1997.

[34] Eckart Zitzler and Simon Künzli. Indicator-Based Selection in Multiobjective Search. In *Conference on Parallel Problem Solving from Nature (PPSN VIII)*, pages 832–842, Birmingham, UK, September 2004. Springer.

[35] Eckart Zitzler and Lothar Thiele. Multiobjective optimization using evolutionary algorithms – A comparative case study. *Lecture Notes in Computer Science*, pages 292–304, 1998.

[36] Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M. Fonseca, and Viviane Grunert da Fonseca. Performance Assessment of Multiobjective Optimizers: An Analysis and Review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, April 2003.

Louis-Claude Canon Louis-Claude Canon received his Master degree of computer science in 2007 from both the ESEO (Ecole Suprieure d'Electronique de l'Ouest) and the University of Angers. He is conducting research since then at the Loria (Laboratoire Lorrain de Recherche en Informatique et ses Applications) under the supervision of Emmanuel Jeannot in order to prepare his PhD on uncertainty management in parallel systems. His main research interests include scheduling, multi-objective combinatorial optimization, and stochastic optimization. More informations are available at: http://www.loria.fr/~canon.

Emmanuel Jeannot Emmanuel Jeannot is a full-time researcher at INRIA (The French National Institute for Computer Science and Control) and is doing his research at the LORIA (Laboratoire Lorrain de Recherche en Informatique et ses Applications) in Nancy, France. From sept. 1999 to sept. 2005 he was associate professor at the Universit Henry Poincar, Nancy 1. He got his PhD and Master degree of computer science in 1996 and 1999, respectively, both from Ecole Normale Suprieure de Lyon. His main research interests are scheduling for heterogeneous environments and grids, data redistribution, grid computing software, adaptive online compression, programming models, experimental computer science and managing uncertainties in large-scale environment. More information is available at: http://www.loria.fr/~ejeannot.