Scheduling Strategies for the Bicriteria Optimization of the Robustness and Makespan

Louis-Claude Canon and Emmanuel Jeannot
LORIA, INRIA, Nancy University, CNRS, France
Email: {louis-claude.canon,emmanuel.jeannot}@loria.fr

Abstract

In this paper we study the problem of scheduling a stochastic task graph with the objective of minimizing the makespan and maximizing the robustness. As these two metrics are not equivalent, we need a bicriteria approach to solve this problem. Moreover, as computing these two criteria is very time consuming we propose different approaches: from an evolutionary metaheuristic (best solutions but longer computation time) to more simple heuristics making approximations (bad quality solutions but fast computation time). We compare these different strategies experimentally and show that we are able to find different approximations of the Pareto front of this bicriteria problem.

1 Introduction

The problem of scheduling an application modeled by a task graph with the objective to minimize its execution time is a well studied problem [6, 9]. However, in general 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 of the input data). A schedule is said *robust* if it is able to absorb part of the uncertainty and gives an allocation whose duration (makespan) is still close to a predicted value.

In this paper we tackle the bicriteria problem of scheduling an application with the objective of minimizing the makespan and maximizing the robustness. The context of this study is heterogeneous computing where machines have different speeds and capabilities for which the robustness of a schedule is even harder to achieve.

Metaheuristics such as evolutionary algorithms (EA) are well known for their capabilities to produce solutions close to the optimal [2]. Multi-objective EA (MOEA) like NSGA-II [5] are designed for seeking close to the optimal solutions for multicriteria problems. However, one of the drawbacks of metaheuristics is the extensive computation time to find solutions. This is especially the case for our problem where the evaluation of a single solution is #P-complete. On the other hand, classic heuristics have the advantage to be extremely fast and the drawback to find solutions farther from the optimal than metaheuristics.

The contribution of this paper is the following. We propose a complete suit of strategies (including heuristics and a MOEA) for our problem. The objective is to let the user choose a trade-off between rapidity and optimality of the proposed solutions. Moreover, we present theoretical facts in order to prove that our MOEA converges to the optimal solution. Lastly, for each of the proposed strategies, we are able to give an approximation of the Pareto front (the set of non-dominated solutions) of our bicriteria problem. This will help the user in choosing another trade-off between robustness and makespan.

This paper is organized as follows. In Section 2, we describe the problem. In Section 3 we propose a MOEA and introduce theoretical elements for its convergence. Our heuristics are presented in Section 4. Experiments are shown in Section 5 and final conclusions are given in Section 6.

2 Problem description

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, these costs are given by a random variable (RV). Each RV gives the probability that

an execution or communication cost is in a given interval.

The platform where executes the task graph is composed of a set of heterogeneous processors with a complete topology. A schedule consists then in allocating tasks to the processors respecting the precedence constraints (given by the edges of the DAG), and the resource constraints (no processor can execute two tasks at the same time). We use the *related model* to simulate the tasks execution on the resources. For instance let us assume that task T_i has been allocated to processor P_j. Since the cost of the task is modeled by a RV, we compute one execution by instantiating its cost using the law of its RV. Assuming that the drawn cost is c_i, then the duration is given by the product $c_i\tau_j$, where τ_j is the time to execute one instruction on processor P_j. To compute the communication time between two tasks we proceed similarly (drawing the communication cost – number of bytes to be transmitted – from the corresponding RV and computing the transfer time using the bandwidth and latency of the link used).

Since we use RV to compute communication time and task execution time, the makespan (length of the schedule) of each execution of the tasks on the resources can be different. This poses two problems:

- First of this, a schedule usually tells when a task must start. Due to the stochastic model we use, it is not possible to ensure the start time of each task. 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. It is known to be #P-complete[1] (see [8] for the details). Therefore, having an accurate evaluation of this distribution is extremely costly. We propose two methods. The first one is a Monte Carlo method that consists in simulating the makespan a sufficient number of times to obtain a good approximation of the distribution. The second method assumes some degree of independence in the graph in order to compute in polynomial time an approximation of the distribution.

Once we have a distribution of the makespan we have to define which metrics have to be optimized.

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

Concerning the schedule length, we use the *average makespan* of the distribution. Indeed, minimizing this metrics means that on the average we will minimize the overall execution time.

The problem of defining what robustness is has been studied in the literature. There exists several metrics for describing the robustness of the schedule with regards to the makespan. In [4] we have shown that a good metric is the *makespan standard deviation*. The idea is that a schedule is robust if it always gives a solution close to a given value. Therefore, if one considers the probability density of the makespan distribution, the narrower this distribution the greater the chance to always have a solution close to the average makespan. Since, how narrow is a distribution is directly related to its standard deviation, we have chosen this criterion as the robustness metric.

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 [9] and computing the makespan distribution is #P-complete, this problem is then noted: $\mathrm{NP}^{\#\mathrm{P}}$. Besides, generally, several Pareto-optimal solutions exist. Hence, it requires bicriteria scheduling strategies.

3 A provably convergent MOEA

In a bicriteria problem we need to find the Pareto front, which is the set of every non-dominated solution. Indeed, more than one solution can be optimal since 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 that we use and give theoretical elements in order to prove its convergence. We present beforehand convergence conditions and extend them.

3.1 Convergence conditions

In the mono-objective case, the general conditions under which an EA is guaranteed to converge are given by Theorem 1 of [10]. This theorem states that in order for an EA to converge to the global minimum, its Markovian kernel K should be such that, $K(x, A_\epsilon) \geq \delta > 0$ for all $x \in A_\epsilon^c = E \setminus A_\epsilon$ and $K(x, A_\epsilon) = 1$ for all $x \in A_\epsilon$, where E is the state space of the process, A_ϵ is the set of optimal states and $K(x_t, A) = \mathbf{P}\{X_{t+1} \in A \mid X_t = x_t\}$, namely the probability that the state

of the stochastic process is in A at step $t+1$, when its state is x_t at step t. Theorem 2 of [10] gives more practical implications for the mutation and selection operators, that are that the mutation should be *global* (any schedule can be attained from any other one in a single step with a nonzero probability) and should be followed by an elitist mechanism.

The generalization to the multi-objective case is given by [11]. The basic assumptions of each proposition are that some degree of elitism should be present and that the stochastic process from which offspring are generated must be a homogeneous finite Markov chain with irreducible transition matrix, roughly implying that the mutation operator should be global.

We propose ourselves to extend the existing theory to local mutation operators (mutation that cannot generate any arbitrary solution of the search space in one single step) having some properties which we detail below.

Let us first introduce some notations and definitions. 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.

We now show an auxiliary result:

Theorem 1. *Let $K_c(x,A) \geq \delta_c$ and $K_s(x,A) \geq \delta_s$ for each $x \in A$ and for each $A \subset E$. Then,*

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

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

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

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

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

By induction hypothesis, we have

$$\begin{aligned}
& \int_E (K_c\, K_m\, K_s)^{(M)}(y,A) K_m(x,dy) \\
\geq & \int_E (\delta_c\delta_s)^M\, K_m^{(M)}(y,A) K_m(x,dy)
\end{aligned}$$

And by definition,

$$K_m^{(M+1)}(x,A) = \int_E K_m^{(M)}(y,A) K_m(x,dy)$$

Consequently, the hypothesis is true for $M \geq 1$. □

The main implication of Theorem 1 is that after M generations the kernel of the EA can be bounded by the M-th iteration of the mutation operator kernel. The conditions for convergence given by Theorem 1 of [10] are then fulfilled if the mutation operator becomes global when applied M times (even if it is local at each generation) and if δ_c and δ_s are strictly positive. This implies that crossover should not systematically change solutions (the probability for the crossover to be performed should then be different from 1) and that selection and replacement should not be deterministics (which is not the case for the binary tournament for example, because the worst solution is systematically removed). The EA must still have an elitism mechanism in order to keep any optimal solution.

Adapting this convergence result to the multi-objective case is omitted due to lack of space, but can be drawn by extending [11] (especially its proposition 4) with the current theory.

3.2 MOEA implementation

3.2.1 Algorithm

Several successful modern MOEA exists such as NSGA-II [5] and IBEA [15], just to mention a few. NSGA-II is the reference metaheuristic in the area and performs similarly to IBEA for combinatorial problems. Thus, we have selected the NSGA-II algorithm as it is implemented in the ParadisEO library [3].

This MOEA takes care of the multi-objective aspect and selection phase. We have still to address the crossover and mutation operators and the evaluation part.

3.2.2 Operators

We use the chromosome representation and the crossover and mutation schemes described in [14]. Although this mutation operator is not global, we show that it becomes global when applied a given number of times.

Since the chromosome representation consists of 2 strings, it is necessary to proceed in 2 steps. The case of the assignment string is straightforward. Each time a task is selected, a new processor is chosen randomly for it without any other constraint. Thus, the probability to get 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 the probability that the mutation happens.

The schedule string is a linear extension of the 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 $\mathrm{led}(E)$ and it is shown in [7] that this diameter is upper bounded by $\mathrm{Inc}(E)$, which is the number of pairs of incomparable elements (for independent tasks, this can be up to $\frac{n(n-1)}{2}$). The probability to get a given linear extension after applying $\mathrm{led}(E)$ mutation iterations is thus lower bounded by $\delta_{m_2} = \left(\frac{p_m}{n^2}\right)^{\mathrm{led}(E)} > 0$.

Then after $M = \max(n, \mathrm{led}(E))$ mutation iterations, the resulting probability to obtain any schedule from any given one is then lower bounded by $\delta_m = \delta_{m_1}\delta_{m_2} > 0$ and thus $K_m^{(M)}(x, A) > 0$. Finally, we can apply Theorem 1 to prove the convergence of the EA, by ensuring that the crossover does not happen systematically, and that the selection operator is not deterministic.

3.2.3 Evaluation

As stated in Section 2, the evaluation of any single schedules is #P-complete and thus an accurate evaluation would be too much time consuming. To achieve a correct precision with Monte Carlo (MC) simulations requires a lot of computation time (as shown in Section 5.6). We have thus opted for an approximation scheme: we assume that all the distributions are Gaussian and we compute the final makespan distribution by determining where independence of these distributions can be assumed. This allows fast evaluation with a correct precision in most cases.

4 Heuristics

We introduce in this section a class of heuristics based on HEFT [13] able to generate a set of solutions intended to have good performance for both criteria from a stochastic task graph.

4.1 Aggregation principle

In order to take into account the bicriteria nature of the schedules that are constructed, we aggregate the average makespan with its standard deviation: $f(\mu, \sigma, a) = a \times \frac{\mu}{\mu_{\max}} + (1-a) \times \frac{\sigma}{\sigma_{\max}}$, with $a \in [0, 1]$, μ and σ the current end time mean and standard deviation respectively, and $\mu_{\max}$ and $\sigma_{\max}$ the maximum mean and standard deviation of the makespan. The parameter a allow to weight each criterion accordingly to the importance we give to each one (when $a = 1$, we are only concerned by the makespan average criterion). The goal is to cover the Pareto front by varying a.

4.2 Tasks ordering

The first part of HEFT consists of ordering the task according to their upward ranks. At this stage, we only consider average times rather than the aggregation mentioned above because it gives better results this way.

4.3 Assignment selection

We first point out that the standard deviation criterion is difficult to tackle because contrarily to the average criterion, it is non-monotonic, that is to say that the standard deviation of the end time of a given assignment can lower when tasks are added to the corresponding processor.

We have studied several strategies and present the two most relevant. The first is based on the usual EFT policy. At each step, the schedulable task with the higher rank is selected and every possible assignment to each processor is simulated. For each assignment, the aggregate criterion defined in Section 4.1 is computed and the final assignment is the one minimizing this criterion. This first heuristic scheme is called: HEFT with uncertainty level. According to how is performed the makespan distribution evaluation we have two version of this heuristic: *Hul_MC* when we use the Monte Carlo method and *Hul* when we use the same approximation scheme as for the MOEA.

The second is based on the observation that if a given assignment reduces (by the non-monotonicity property) significantly the standard deviation of the

end time on one processor, it should be preferred to an assignment that has the lowest standard deviation. Thus, it leads us to compute the overall maximum of every processor end time for each possible assignment and to apply the same minimization selection than before (by aggregating the mean and standard deviation of this overall maximum). We call it *Hulm* (HEFT with uncertainty level and maximum). Here we do not use the Monte Carlo method to evaluate the makespan distribution because we want this heuristic to be as fast as possible.

5 Experiment

5.1 Testbed

The task graphs generation, the heterogeneous platform, the stochastic simulation and the MOEA require each a set of parameters. For the two first objects, we use the coefficient of variation [1] to model heterogeneity of the task graph costs and platform resources. This coefficient is the ratio between the mean and the standard deviation of a given value (following it-self a Gamma distribution with the corresponding parameters). Each parameter susceptible to change comes with a coefficient of variation (denoted by the prefix "V_"). In Table 1, each value in bracket correspond to a single experiment while the values outside are the default ones, leading thus to 150 different experiment scenarios.

Only non-obvious parameters are described. Task graphs are generated from the *Strassen* algorithm description and randomly in two ways accordingly to [12], namely *samepred* and *layrpred*. Some of the parameters are ignored for Strassen graphs: the communication cost (it is already induced by the number of tasks and by the execution cost), V_Cost (the coefficient of variation associated with these 2 costs is zero) and the average number of edges per node. Besides, the numbers of tasks are instead: 23, 163 and 1143.

The distributions of the costs in the task graphs follow either a Beta distribution with parameters $\alpha = 2$ and $\beta = 5$ (see [4] for a justification) or an exponential or a normal one. The uncertainty level (UL) is 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). Lastly, we have to define the number of MC simulations required to achieve a meaningful precision. If we assume that the makespan distribution is normal, the theory of statistics says that 20,000 MC simulations are needed in order to have less than 5% of precision with a confidence level of 99% for both criteria (750,000 simulations would be necessary to have a precision less than 1%, which is too much time consuming).

Parameter	default { experiment }
Task graph	
GraphType	samepred layrpred strassen
TaskNumber	1000 { 10 100 1000 }
SeedApp	0 { 1 2 3 4 5 6 7 8 9 }
ExeCost (FLOP)	100 M { 10 M 1 G }
CommCost (B)	100 k { 10 k 1 M }
V_Cost	0.5 { 0.001 0.1 0.3 1 2 }
Avg Edge/Node	3. { 1. 5. }
Distribution	BETA { EXP NORMAL }
UL	1.1 { 1.0001 1.2 1.5 2 3 5 }
V_UL	0.3 { 0.001 0.1 0.5 1 2 }
Platform	
ProcessorNumber	50 { 25 100 }
SeedPlat	0 { 1 2 }
MachPower (FLOPS)	2.5 G
LinkLatency (ms)	0.1
V_Plat	0.5 { 0.001 0.1 0.3 1 2 }
LinkBandwidth (B/s)	50 M
V_LinkBandwidth	1 { 0.001 0.1 0.3 0.5 2 }

Table 1. Task graph and platform parameters

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 *Hul* and *Hulm* heuristics, we vary the parameter a from 0 to 1 by step of 0.005. For *Hul_MC*, the step size is 0.05.

5.2 Search space

In a first attempt to study the problem specificity, we characterize the search space by generating 5,000 random schedules (denoted by *rand* in the following) and extreme schedules present 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 1 depicts the search space of one experiment scenario.

An immediate observation is the apparent correlation between both criteria (even the SE and the NW sets follow a linear pattern). Table 2 summarize the correlation coefficients over every experiment scenario (a value close to 1 implies a high linear relationship between the criteria) for the *rand* schedules. Tukey's five

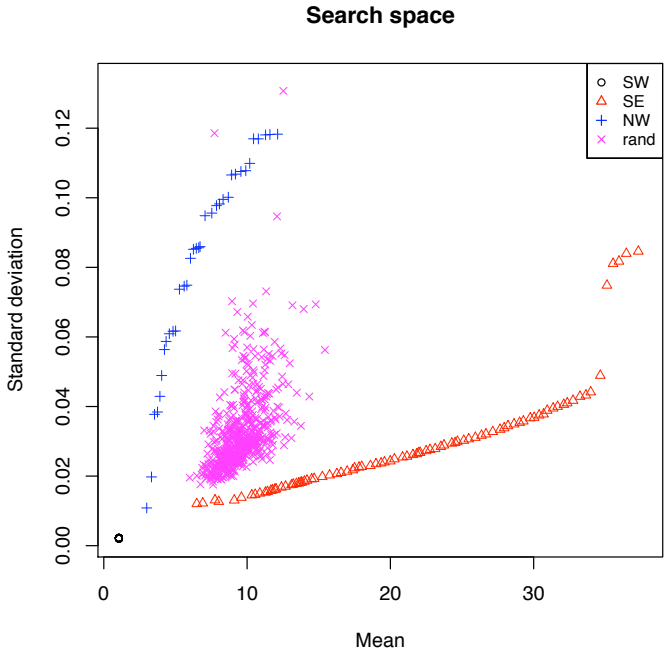

Figure 1. Mean vs. std. dev. of the makespan of different schedules in random and border cases, with a layrpred graph and TaskNumber $= 1000$

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 schedules for Strassen graphs have highly correlated means and standard deviations.

Graph	Min	25%	Med	75%	Max
STRASSEN	0.26	0.78	0.81	0.81	0.91
LAYRPRED	0.20	0.49	0.57	0.73	0.80
SAMEPRED	0.095	0.31	0.40	0.76	0.81

Table 2. Tukey's five number summary of correlation coefficients by graph kind

It is also worth noting that the SW front is almost always isolated from other regions and has a limited spread. Pareto-optimal solutions are hence quite close in regards to the global search space. When V_UL $>$ 0.3, correlations are however the worst and the SW fronts are larger. Minimizing the average makespan will therefore not necessarily induce good robustness when V_UL takes high values.

5.3 Normality test

We conduct normality tests on our schedules to validate the normality assumption we have done at several occasions (in the approximation scheme, for the confidence intervals and for the reduction of the robustness metrics to the standard deviation). We have selected the Anderson-Darling (AD) test, which is one of the best EDF omnibus tests for normality. The returned statistic corresponds more or less to the distance with a normal. Half of the schedules have a statistic lower than 31.6 (the same as a Student t distribution with 8 degrees of freedom, which is visually really close to a normal) and 84% of these are more closer to a normal than a Weibull with parameter $\lambda = 1$ and $k = 2$ (Weibull are considered similar to normal for $k = 3.4$). Therefore, it allows us to consider distributions as Gaussian in most cases.

Although, it is hard to draw any general trend in function of the kind of graph or region in the search space, we notice that Strassen graphs, *SE* and *rand* schedules give the best normality results.

5.4 Average quality

The purpose of this section is to assess the quality of the schedules generated by the strategies presented in Section 3 and 4 in term of robustness and average performance. Figure 2 is a representative example regarding the position of the approximation sets (or Pareto fronts) of each strategy. Error bars denote the confidence intervals of each schedule evaluation. We also compute the binary ϵ-indicator [16] for each pair of heuristics (in Table 3). The value $I_\epsilon(A, B)$ corresponds to the ratio between a chosen solution of A and one of B for a selected criterion. It is roughly related to the relative distance between the two Pareto 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, the two fronts are incomparable.

B\A	Hul	Hulm	Hul_MC	SW
Hul	1.00	1.18	1.01	1.00
Hulm	0.90	1.00	0.87	0.90
Hul_MC	1.04	1.20	1.00	1.04
SW	1.04	1.20	1.02	1.00

Table 3. Comparison of the Pareto front: the binary ϵ-indicator values $I_\epsilon(A, B)$ for all pairs of strategies

It is clear in this example that *Hulm* performs the

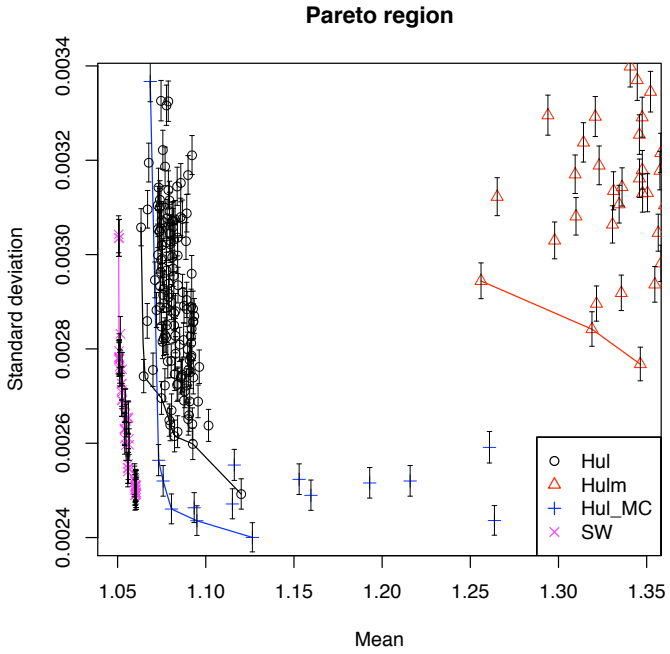

Figure 2. Mean vs. std. dev. of the makespan of different schedules in the Pareto region, with a layrpred graph and V_Cost $= 1$

worst and *SW* is better than *Hul*. $I_\epsilon(\text{SW}, \text{Hul\_MC}) > I_\epsilon(\text{Hul\_MC}, \text{SW})$ does not necessary mean that *Hul_MC* is better than the MOEA. It can be explained by the fact that the indicator is a ratio between either 2 means or between 2 standard deviations. Since this last criteria varies relatively the most, the presence of more robust solutions will have more impact on the ϵ-indicator.

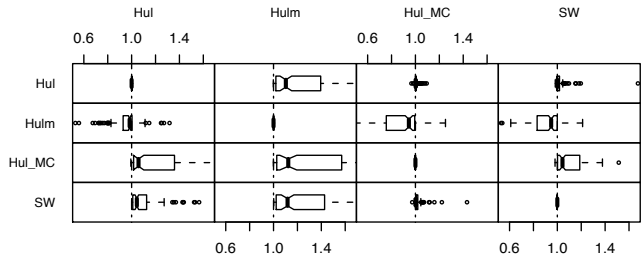

Figure 3. Comparison of the Pareto fronts: boxplot of the indicators over every experiment

Since the ϵ-indicator is a ratio, it allows doing comparison on every experiment scenario. Figure 3 represents the summary of every computed indicator in the form of boxplots. Boxplots allow to represent a five number summary of a given set (in this case, this is the 5th percentile, the first quartile, the median, the last quartile and the 95th percentile) and the outliers that exceed these values. For example, on the line *Hulm* and column *SW*, these 5 values are respectively 0.61, 0.83, 0.94, 1 and 1.21. We show that the previous remarks are general, that is that *Hulm* is the worst heuristics (despite being the most sophisticated) and that the MOEA, *Hul* and *Hul_MC* are mostly incomparable. Although it does not appear on this figure, the MOEA systematically outperforms other heuristics in term of average makespan.

5.5 Computation time

We realize the average of every heuristic over 5 runs with random graphs (Strassen graphs having different number of tasks). These measures are represented on Table 4. The second time includes the MC evaluation of the schedules (except for *Hul_MC* because it is already included and *SW* because it is negligible).

Task number	10	100	1000
Hul	0.4"/1"	19"/1.2'	5.7'/37.6'
Hulm	0.5"/1"	2'/4.6'	1h33'/2h5'
Hul_MC	1.1'	19.6'	3h24'
SW	55.8'	1h30'	6h44'

Table 4. Execution time of every strategy

It would have been possible to reduce the number of MC simulations in order to have similar execution time for *Hul* and *Hul_MC*. However, with 600 simulations, the standard deviation precision is about 25% which is quite high. Hence, *Hul_MC* is relevant only with high number of simulations.

5.6 MOEA evolution

Figure 4 illustrates the evolution of both criteria in function of the time taken by the MOEA when MC evaluations are used. The evolution of the standard deviation is less stable than with the average, which could be caused by a greater difficulty to generate robust schedules than efficient ones.

6 Conclusion

In this article we have studied the problem of maximizing the robustness and minimizing the execution time of an application modeled by a stochastic task graph. This is a very difficult problem. Minimizing the makespan is an NP-hard problem while computing

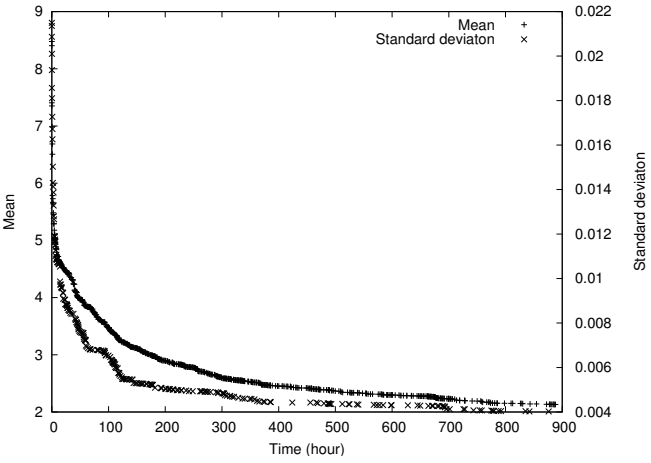

Figure 4. Evolution of the MOEA for both criteria with MC simulations

the metrics requires solving a #P-complete problem which is very time-consuming.

In order to tackle the problem of trade-off between computation time of the solution and the quality of the solution, we have proposed different strategies. The slowest strategy is our evolutionary algorithm. The proposed *Hul_MC* (an extension of the makespan-centric heuristic HEFT) is faster and but provides worse makespan solution than our MOEA. *Hul* and *Hulm* are even faster heuristics but give even worse results for both criteria. Concerning our multi-objective evolutionary algorithm, we have given theoretical elements in order to prove its convergence by extending previous results on the global nature of the mutation operator.

All these strategies are also able to give a Pareto front of the solution. Therefore we are able to help the user in choosing another trade-off between makespan and robustness.

It is also important to remark that our method can be extended to other makespan-centric heuristics (BIL, PCT, HBMCT, CC, ILHA). Evaluation of these extensions is left to future work. We also need a better evaluation mechanism for our MOEA in order to systematically outperform *Hul_MC* for the robustness. This requires either to better take into consideration the approximation of the current fitness function or to develop more precise evaluation methods.

References

[1] S. Ali, H. J. Siegel, M. Maheswaran, D. Hensgen, and S. Ali. Representing Task and Machine Heterogeneities for Heterogeneous Computing Systems. *Tamkang Journal of Science and Engineering,* Special 50th Anniversary Issue, 3(3):195–207, Nov. 2000.

[2] T. Bäck. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, Oxford, UK, 1996.

[3] S. Cahon, N. Melab, and E.-G. Talbi. ParadisEO: A Framework for the Reusable Design of Parallel and Distributed Metaheuristics. *Journal of Heuristics*, 10(3):357–380, 2004.

[4] L.-C. Canon and E. Jeannot. A Comparison of Robustness Metrics for Scheduling DAGs on Heterogeneous Systems. In *HeteroPar'07*, pages 568–567, Austin, Texas, USA, Sept. 2007.

[5] K. Deb, S. Agrawal, A. Pratab, and T. Meyarivan. A Fast Elitist Non-Dominated Sorting Genetic Algorithm for Multi-Objective Optimization: NSGA-II. In *Parallel Problem Solving from Nature VI Conference*, pages 849–858, Paris, France, 2000.

[6] H. El-Rewini, T. Lewis, and H. Ali. *Task Scheduling in Parallel and Distributed Systems*. Prentice Hall, 1994.

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

[8] J. N. Hagstrom. Computational complexity of PERT problems. *Networks*, 18(2):139–147, 1998.

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

[10] G. Rudolph. Convergence of Evolutionary Algorithms in General Search Spaces. In *International Conference on Evolutionary Computation*, pages 50–54, Nagoya, Japan, May 1996.

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

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

[13] H. Topcuoglu, S. Hariri, and M.-Y. Wu. Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing. *Transactions on Parallel and Distributed Systems*, 13(3):260–274, Mar. 2002.

[14] L. Wang, H. J. Siegel, V. R. Roychowdhury, and A. 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, Nov. 1997.

[15] E. Zitzler and S. Künzli. Indicator-Based Selection in Multiobjective Search. In *Conference on Parallel Problem Solving from Nature (PPSN VIII)*, pages 832–842. Springer, 2004.

[16] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca. Performance Assessment of Multiobjective Optimizers: An Analysis and Review. *IEEE Transactions on Evolutionary Computation*, 7(2):117–132, Apr. 2003.