A Two-Phase Heuristic for the Scheduling of Independent Task on Computational Grids

Frédéric Pinel, Johnatan Pecero and Pascal Bouvry
Computer Science and Communications
University of Luxembourg
firstname.lastname@uni.lu

Samee U. Khan
Dept. of Electrical and Computer Engineering
North Dakota State University, Fargo, ND, USA
samee.khan@ndsu.edu

***Abstract*—The sensitivity analysis of a cellular genetic algorithm with local search is used to design a new and simpler heuristic for the problem of scheduling independent tasks. The proposed heuristic improves the previously known Min-Min heuristic. Moreover, it provides schedules of similar quality to the reference cellular genetic algorithm in a significantly reduced runtime. This heuristic is evaluated across twelve different classes of scheduling instances.**

***Keywords*-Energy-efficient Scheduling Algorithms**

I. Introduction

The assignment of tasks to computing resources in a distributed computing system is a challenging problem. The aforementioned is further complicated with the introduction of energy minimization as a system criterion.

This work contributes to this problem by suggesting a new heuristic for the scheduling of independent tasks that can be applied to the energy-efficient operation of distributed systems.

An interesting feature of this work, is the path that leads to its definition, which is a direct consequence of the sensitivity analysis performed earlier in [1], on an elaborate cellular genetic algorithm, termed PA-CGA [2].

The heuristic, termed as 2PH, is a simple extension to the well-known scheduling heuristic for independent task scheduling, the Min-Min [3].

The rest of the paper is organized as follows. Section II presents the previous work on the sensitivity analysis of the PA-CGA evolutionary algorithm (EA). Section III describes the new heuristic and provides a comparison with the Min-Min heuristic and the PA-CGA evolutionary algorithm.

II. Sensitivity Analysis of a Cellular GA

Ref. [1] performed sensitivity analysis on a cellular genetic algorithm, PA-CGA [2], which was designed to schedule independent tasks on a distributed system. This section briefly presents the results from the aforementioned previous papers.

Sensitivity analysis aims to identify how uncertainty in each of the parameters influences the uncertainty in the system output [4]. This technique can answer the following important question: given uncertainty in system parameters, which ones affect (the most and the least) the system output. (also known as screening).

In the previous work [1], the analyzed output was the quality of the solution produced by the PA-CGA EA. The quality of the solution, the schedule of tasks, is further defined in the next Section, which also presents the scheduling problem in more details.

Sensitivity analysis is not only useful for parameter tuning, but also at design-time. The work presented here is an example of such a process. The results of the analysis are used to design a new heuristic to solve the independent task scheduling problem.

The chosen sensitivity analysis method is based on decomposing the variance of the output, as indicated in [4]. The exact implementation used is an extension to the Fourier Amplitude Sensitivity Test proposed in [5], called Fast99. Ref. [5] allows the computation of first order effects and interactions for each parameter. Parameters interaction occurs when the effect of the parameters on the output is not a sum of their single (first order) effects.

A. Parallel Asynchronous Cellular GA

In the previous work, the sensitivity analysis was performed on a parallel asynchronous cellular genetic algorithm [2] for scheduling of independent tasks in a computational grid.

The problem that the EA attempts to solve arises quite frequently in parameter sweep applications, such as the Monte-Carlo simulations [6]. In these applications, many tasks with almost no interdependencies are generated and submitted to the computational grid to be efficiently scheduled. Efficiency means to allocate tasks as fast as possible and to optimize some criteria, such as makespan or flowtime. Makespan is among the most important optimization criterion of a grid system. Indeed, it is a measure of the system's productivity (throughput). Task scheduling is treated as a single objective optimization problem, in which the makespan is minimized. *Makespan*, the finishing time of latest task, is defined as:

$$\min_{S} \max\{F_t : t \in Tasks\}, \tag{1}$$

where F_t is the finishing time of task t in a schedule S.

More precisely, assuming that the computing time needed to perform a task known *a priori* (assumption that is usually made in the literature [7], [8], [9]), the problem is represented with the Expected Time to Compute (ETC) modeled by Braun et al. [7]. The instance definition of the problem is as follows:

- nb_tasks: the *number* of independent (user/application) *tasks* to be scheduled.
- $nb\_machines$: the *number* of heterogeneous *machine* candidates to participate in the planning.
- The *workload* of each task (in millions of instructions).
- The *computing capacity* of each machine (in *MIPS*).
- $ready_m$: Ready time indicating when machine m will have finished the previously assigned tasks.
- The Expected Time to Compute (ETC) matrix ($nb\_tasks \times nb\_machines$) in which $ETC[t][m]$ is the expected execution time of task t on machine m.

The two benchmark instances used for this analysis consisted of 512 tasks and 16 machines. Both instances represented different classes of ETC matrices. The classification is based on three parameters: (a) task heterogeneity, (b) machine heterogeneity, and (c) consistency [10]. Instances are labelled as g_x_yyzz where:

g	stands for Gamma distribution (used for generating the matrix).
x	stands for the type of consistency (c for consistent, i for inconsistent, and s for semi-consistent). An ETC matrix is considered consistent when the following is true: if a machine m_i executes a task t faster than machine m_j, then m_i executes all tasks faster than m_j. Inconsistency means that a machine is faster for some tasks and slower for some others. An ETC matrix is considered semi-consistent if it contains a consistent sub-matrix.
yy	indicates the heterogeneity of the tasks (hi means high, and lo means low).
zz	indicates the heterogeneity of the resources (hi means high, and lo means low).

B. Algorithm

The EA analyzed is a parallel asynchronous CGA (PA-CGA) [2], which is based on the study reported in [11]. Cellular genetic algorithms (cGAs) [12] are a kind of GA with a structured population in which the individuals are spread in a two dimensional toroidal mesh and are only allowed to interact with their neighbors. The algorithm iteratively considers each individual in the mesh. This individual may only interact with individuals belonging to its neighborhood, moreover parents are chosen among the neighbors using a given criterion. The crossover and mutation operators are applied to the individuals, with probabilities p_c and p_m, respectively. Following the crossover and mutation operations, the algorithm computes the fitness value of the new offspring individual (or individuals), and inserts it (or one of them) instead of the current individual in the population following a given replacement policy. This loop is repeated until termination condition(s) are fulfilled.

In the PA-CGA, the population is partitioned into a number of contiguous blocks with a similar number of individuals (Figure 1). Each block contains $pop\_size/\#threads$ individuals, where $\#threads$ represents the number of concurrent threads executed. To preserve the exploration characteristics of the CGA, the communication between individuals of different

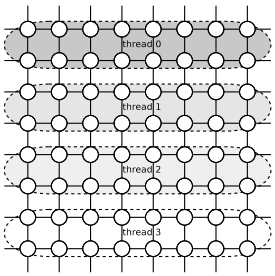

Fig. 1. Partition of an 8×8 population over 4 threads.

blocks is made possible. This neighborhood may include individuals from other population blocks. This allows an individual's genetic information to cross block boundaries.

The different threads evolve their population block independently and do not wait for each other to complete their generation (the evolution of all the individuals in their block) before pursuing their evolution. Therefore, if a breeding loop takes longer for an individual of a given thread, the individuals evolved by the other threads may go through more generations.

The combination of a concurrent execution model with the neighborhoods crossing block boundaries, lead to concurrent access to shared memory. To ensure safe concurrent memory access, the access is synchronized to individuals with a POSIX [13] read-write lock. This high-level mechanism allows concurrent reads from different threads, but not concurrent reads with writes, nor concurrent writes. In the two latter cases, the operations are serialized.

The algorithm implements a local search operator, H2LL (Algorithm 1), which is conceived for the scheduling problem considered in this research. This operator moves a randomly chosen task from the most loaded machine (a machine's load is the total of the tasks completion times assigned to the machine) to a selected candidate machine among the N least loaded. A candidate machine is selected if its new completion time, with the addition of the task moved, is the smallest of all the candidate machines. This new completion time must also remain inferior to the makespan. This operation is performed several times (a parameter of the local search).

Algorithm 1 Pseudo-code for H2LL, our local search.

```
1: for all iter iterations do
2:    sort machines on ascending completion time
3:    task ← random task of the last machine in machines;
4:    best_score ← CT[last machines]; {makespan}
5:    for all mac in N first machines do
6:       new_score ← CT[mac] + ETC[mac][task];
7:       if new_score < best_score then
8:          best_mac ← mac;
9:          best_score ← new_score;
10:      end if
11:   end for
12:   move task to best_mac if any
13: end for
```

The following parameters have been used in the analysis of PA-CGA. The population is initialized randomly, except

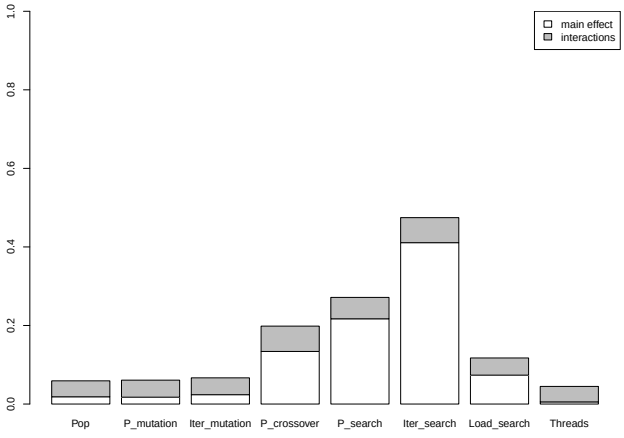

Fig. 2. Sensitivity analysis, hihi instance.

for one individual obtained with the Min-Min heuristic [3]. The selection operator used is binary tournament. The recombination operator used is the one-point (opx) crossover and the mutation operator moves one randomly chosen task to a randomly chosen machine. The neighborhood shape used is linear 5 (L5), also known as the Von Neumann neighborhood, which is composed of the 4 nearest individuals (measured in Manhattan distance), plus the individual evolved. The replacement strategy is the "replace if better" *i.e.*, the newly generated offspring replaces the current individual if it improves the parent fitness value.

C. Results of the Sensitivity Analysis

Figure 2 depicts for each parameter studied, their linear and non-linear effects on the output for the problem instance with high tasks and resources heterogeneity. The quality of the solution is recorded as the average makespan over 4 independent runs.

This study clearly showed that the local search parameters and notably the maximum number of iterations, influence the output the most. It plays a role twice as big as the second most influential parameter: the local search rate. Because the chosen method for the sensitivity analysis is quantitative, it allows such comparisons, whereas qualitative methods can only indicate the order of importance. This result is consistent with related works in the scheduling literature which highlighted the importance of the local search when dealing with hybrid metaheuristics.

These results also highlighted that other parameters play a limited role *i.e.*, population size, mutation rate, iterations, as well as the number of threads. This is also beneficial because values that have a positive impact on the other aspects of the algorithm, such as runtime can be selected without impacting the quality of the solutions. Indeed, the proposed algorithm was designed to be run for a limited period of time (wall clock); therefore choosing a smaller population and a higher number of threads will provide more generations.

TABLE I
SETTINGS FOR THE COMPARISON WITH OTHER ALGORITHMS IN THE LITERATURE.

Instance size	128 tasks $\times$ 16 machines
Instance classes	12
Instances per class	30
PA-CGA runtime	1-5 seconds
PA-CGA population	8×8
PA-CGA thread(s)	1
PA-CGA search iterations	5
2PH search iterations	30

III. A TWO-PHASE HEURISTIC

The previous section recapped the findings of the sensitivity analysis performed in [1]. It clearly showed that the specifically designed local search operator, H2LL, was very important to the quality of the schedules found. More precisely, it also found that the number of iterations for which to perform this local search was of prime importance.

This algorithm is important because it improves on a well-known heuristic, the Min-Min [3], which has been recently applied to the problem of energy-efficient scheduling of tasks [14]. Therefore, improving this algorithm should lead to improvements in their derivative applications to energy-efficiency.

A. The 2PH Heuristic Description

The algorithm proposed is simply the execution of the Min-Min, followed by the local search operator H2LL, originally designed for the PA-CGA.

The number of iterations for the local search in H2LL is increased from 5 to 30. This is because the sensitivity analysis indicated that this parameter influences the quality of the schedules. Additionally, the local search is performed only once for the 2PH. In contrast with PA-CGA which executes it for each individual in the population at each generation. Therefore more iterations can be afforded in the 2PH.

Although this new heuristic is simple, there is *a priori* evidence that it should perform well. The next section describes how it compares to other algorithms.

B. Configuration for Simulations

The 2PH extends the Min-Min heuristic with an additional phase, the local search. Therefore, it seems natural to evaluate how this additional phase improves the results of Min-Min. This is the first point of comparison.

Secondly, the 2PH differs from the PA-CGA EA for the evolutionary part, and the number of iterations of local search (5 for PA-CGA versus 30 for the 2PH). Therefore, it also seems natural to examine the impact of these differences on the schedules produced. Wallclock times for the 2PH and the PA-CGA EA implementations are useful to measure the performance of the algorithms. Moreover, defining schedules for independent tasks is often a time-critical activity.

Table I summarizes the different points of comparison for the evaluation of the 2PH. A total of 360 instances were used in the comparison (30 instances of each class). The PA-CGA

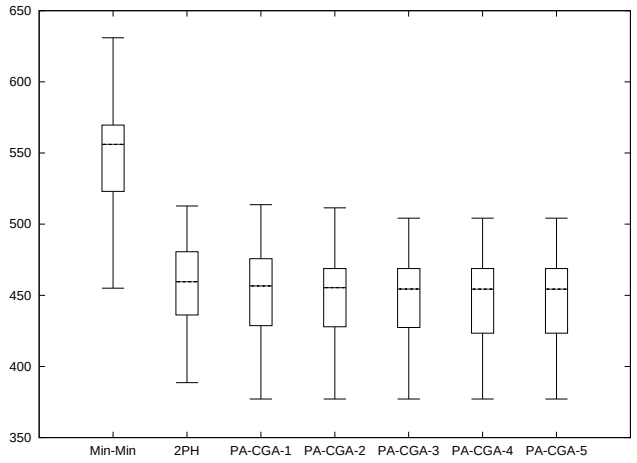

Fig. 3. Makespan results for the g_c_hihi instances.

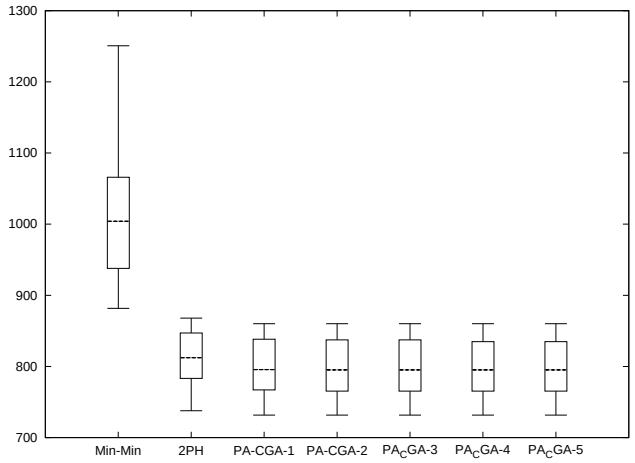

Fig. 4. Makespan results for the g_c_hilo instances.

EA was run for 1 to 5 seconds, wallclock time. The original article [2] ran the algorithm for 90 seconds: however, the 2PH only took 3 milliseconds to complete these instances. Therefore the same treatment was chosen for all algorithms. One thread for PA-CGA was chosen for the same reason. Moreover, the sensitivity analysis showed that the number of threads does not play the biggest role in the search for good solutions. Finally, PA-CGA with 1 thread completes over 100,000 evaluations per second of runtime, which is sufficient for an EA. It should be noted that PA-CGA initializes its population randomly (uniform distribution) except for one individual, which is the result of the Min-Min heuristic. As mentioned earlier, 30 iterations were chosen for the 2PH instead of 5 for the PA-CGA EA. The other parameters for PA-CGA have identical values to those chosen for the sensitivity analysis.

C. Simulation Results

This section presents the simulation results of the different algorithms: (a) the Min-Min heuristic, (b) the 2PH, and (c) the PA-CGA EA with various runtimes, in seconds.

The graphics are box-and-whisker plots. They show the minimum and maximum makespans, the first and third quartiles and the median value, for the 30 instances of each ETC class.

Overall, the 2PH improves the quality of the resource allocation significantly over Min-Min, and provides results of similar quality to PA-CGA. These results are good because it achieves this in about 3 milliseconds on a Xeon E5400 server-class machine.

The results for the different ETC classes are detailed next. In the consistent instances, the results for the g_c_lohi instances stand out because all the algorithms provide similar results. The results for the g_c_lolo instances show that PA-CGA provides better solutions than the 2PH.

In the semi-consistent instances, the results for the g_s_lohi instances are similar across all the algorithms. The results for

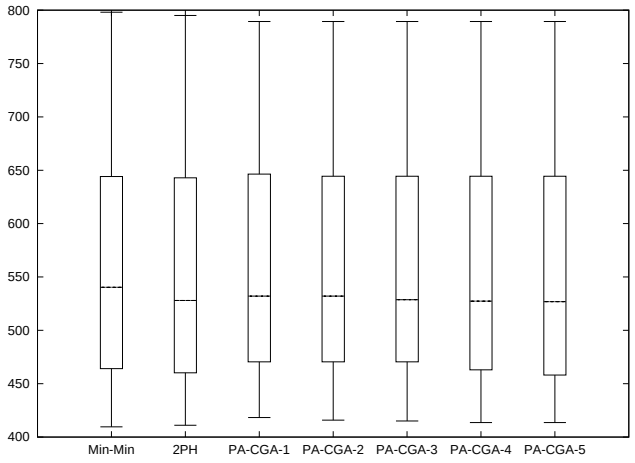

Fig. 5. Makespan results for the g_c_lohi instances.

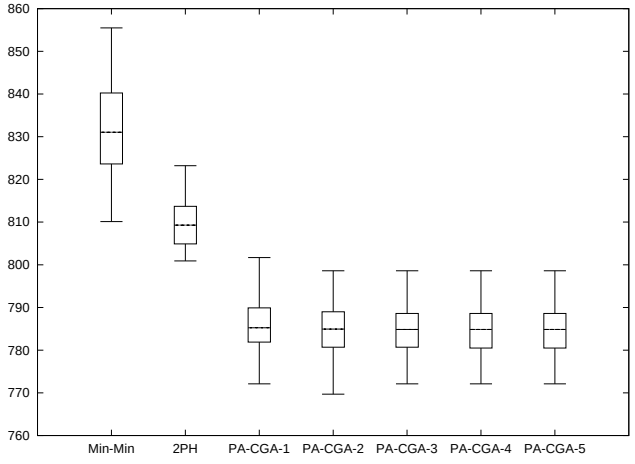

Fig. 6. Makespan results for the g_c_lolo instances.

the g_s_lolo instances show that the 2PH provides the best solutions.

In the inconsistent instances, the results for the g_i_lohi

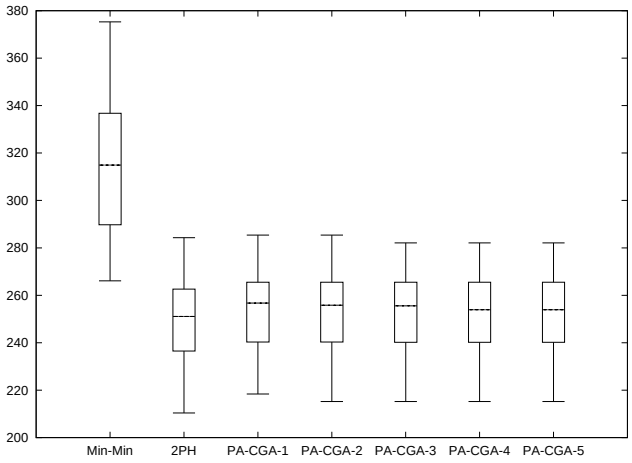

Fig. 7. Makespan results for the g_s_hihi instances.

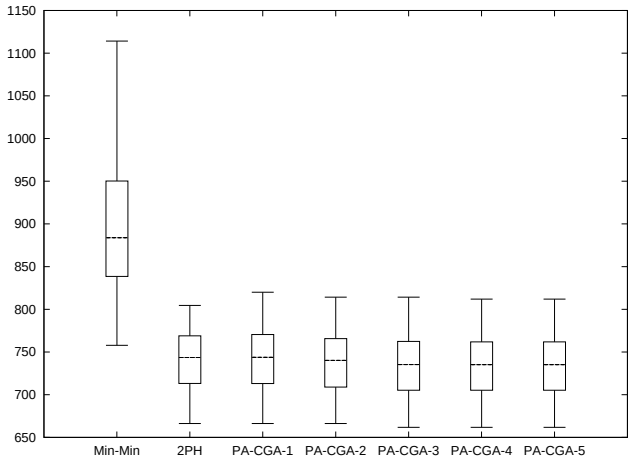

Fig. 8. Makespan results for the g_s_hilo instances.

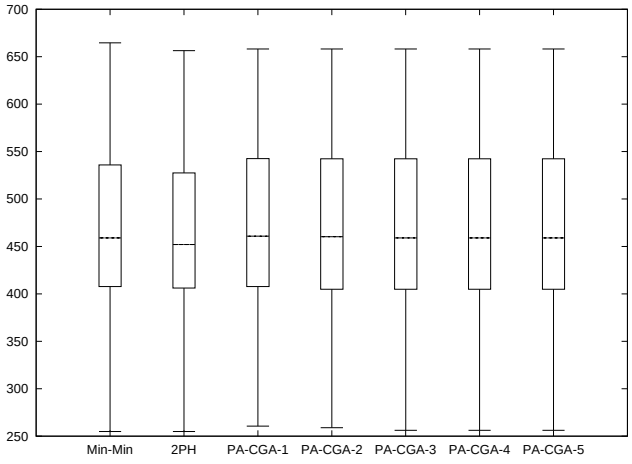

Fig. 9. Makespan results for the g_s_lohi instances.

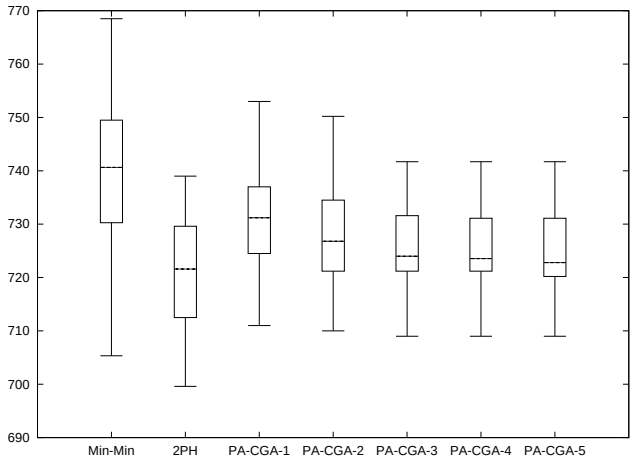

Fig. 10. Makespan results for the g_s_lolo instances.

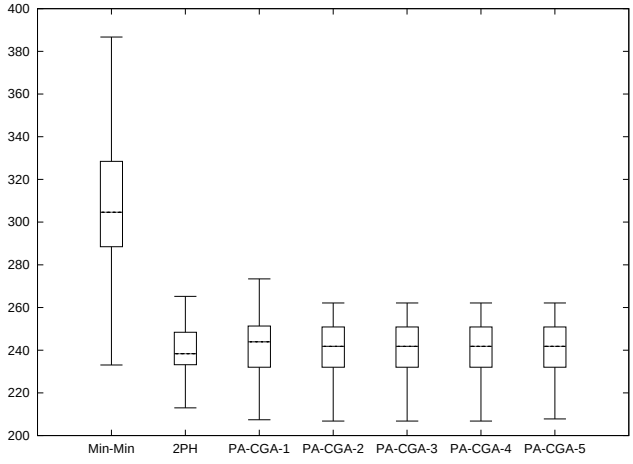

Fig. 11. Makespan results for the g_i_hihi instances.

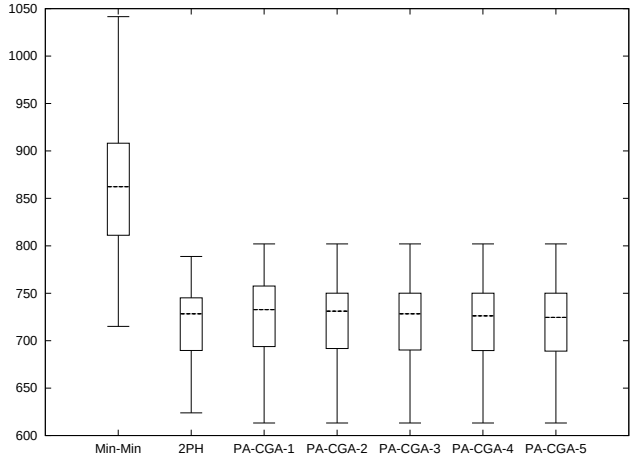

Fig. 12. Makespan results for the g_i_hilo instances.

instances are again similar across all the algorithms. The results for the g_i_lolo instances show that the 2PH provides the best solutions, like the results for the g_s_lolo instances.

IV. Conclusions

This paper exploits the results of the sensitivity analysis of a parallel asynchronous cellular genetic algorithm, with

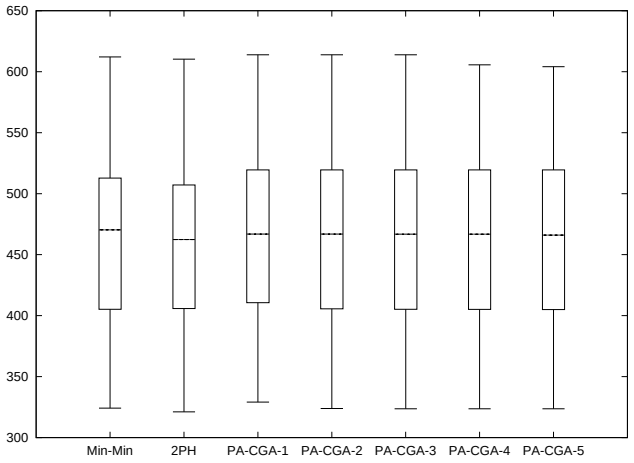

Fig. 13. Makespan results for the g_i_lohi instances.

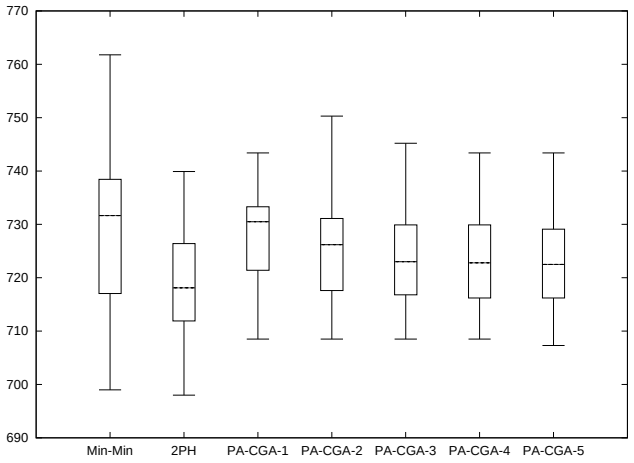

Fig. 14. Makespan results for the g_i_lolo instances.

local search. The analysis lead to the design of a simple 2-phase heuristic for the scheduling of independent tasks. This new heuristic was compared against two algorithms from the literature, (a) the parallel asynchronous cellular genetic algorithm, and (b) the Min-Min heuristic. In most problem instances, it found equivalent schedules in much less time (milliseconds versus seconds) to the cellular genetic algorithm. It also significantly improves the schedules found by the Min-Min heuristic, with little additional computation cost. Moreover, this little computational cost scales well with the problem sizes.

This new heuristic can be applied to solve the problem of energy-efficient scheduling of independent tasks, where Min-Min is currently used.

An interesting extension of this current work could be the inclusion of the analysis of how this second phase succeeds in improving Min-Min, and confirm how larger instances can be quickly processed by this two-phase heuristic.

ACKNOWLEDGMENT

This work is supported by the Fonds National de la Recherche Luxembourg, CORE Project Green-IT.

REFERENCES

[1] F. Pinel, G. Danoy, and P. Bouvry, "Evolutionary algorithm parameter tuning with sensitivity analysis," in *Proc. of the International Joint Conference on Security and Intelligent Information Systems*, L. Bolc, Ed. Lecture Notes in Computer Science, 2011 [to appear].

[2] F. Pinel, B. Dorronsoro, and P. Bouvry, "A new parallel asynchronous cellular genetic algorithm for scheduling in grids," in *Proceedings of the 2010 IEEE International Symposium on Parallel and Distributed Processing, Workshops and Phd Forum, IPDPSW 2010*, 2010, p. 206b.

[3] O. H. Ibarra and C. E. Kim, "Heuristic algorithms for scheduling independent tasks on nonidentical processors," *Journal of the ACM*, vol. 24, no. 2, pp. 280–289, 1977.

[4] A. Saltelli, S. Tarantola, F. Campolongo, and M. Ratto, *Sensitivity Analysis in Practice: A Guide to Assessing Scientific Models*. Wiley, 2004.

[5] A. Saltelli, S. Tarantola, and K. Chan, "A quantitative, model independent method for global sensitivity analysis of model output," *Technometrics*, vol. 41, pp. 39–56, 1999.

[6] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman, "Heuristics for scheduling parameter sweep applications in grid environments," in *Heterogeneous Computing Workshop*, 2000, pp. 349–363.

[7] T. D. Braun, H. J. Siegel, N. Beck, L. L. Bölöni, M. Maheswaran, A. I. Reuther, J. P. Robertson, M. D. Theys, B. Yao, D. Hengsen, and R. F. Freund, "A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems," *Journal of Parallel and Distributed Computing*, vol. 61, no. 6, pp. 810–837, 2001.

[8] A. Ghafoor and J. Yang, "Distributed heterogeneous supercomputing management system," *IEEE Comput.*, vol. 26, no. 6, pp. 78–86, 1993.

[9] M. Kafil and I. Ahmad, "Optimal task assignment in heterogeneous distributed computing systems," *IEEE Concurrency*, vol. 6, no. 3, pp. 42–51, 1998.

[10] S. Ali, H. J. Siegel, M. Maheswaran, D. Hensgen, and S. Ali, "Representing task and machine heterogeneities for heterogeneous," *Journal of Science and Engineering, Special 50 th Anniversary Issue*, vol. 3, pp. 195–207, 2000.

[11] F. Pinel, B. Dorronsoro, and P. Bouvry, "A new parallel asynchronous cellular genetic algorithm for de novo genomic sequencing," in *Proceedings of the IEEE International Conference on Soft Computing and Pattern Recognition (SOCPAR09)*, 2009, pp. 178–183.

[12] E. Alba and B. Dorronsoro, *Cellular Genetic Algorithms*, ser. Operations Research/Compuer Science Interfaces. Springer-Verlag Heidelberg, 2008.

[13] IEEE and The Open Group, "Posix (ieee std 1003.1-2008, open group base specifications issue 7)," http://www.unix.org, 2008.

[14] F. Pinel, J. Pecero, S. U. Khan, and P. Bouvry, "Memory-aware green scheduling on multi-core processors," in *Proceedings of the Second International Workshop on Green Computing (2010)*, ser. ICPP, 2010.