Efficient Batch Job Scheduling in Grids using Cellular Memetic Algorithms

Fatos Xhafa
Dept. de Llenguatges i Sistemes Informàtics
Universitat Politècnica de Catalunya
C/Jordi Girona 1-3, 08034 Barcelona, Spain
fatos@lsi.upc.edu

Enrique Alba and Bernabé Dorronsoro
Dpto. de Lenguajes y Ciencias de la Computación
E.T.S.I. Informática
Campus de Teatinos, Málaga - 29071, Spain
{eat,bernabe}@lcc.uma.es

Abstract

Computational Grids are an important emerging paradigm for large-scale distributed computing. As Grid systems become more wide-spread, techniques for efficiently exploiting the large amount of Grid computing resources become increasingly indispensable. A key aspect in order to benefit from these resources is the scheduling of jobs to Grid resources. Due to the complex nature of Grid systems, the design of efficient Grid schedulers becomes challenging since such schedulers have to be able to optimize many conflicting criteria in very short periods of time. In this work we exploit the capabilities of Cellular Memetic Algorithms (cMAs) for obtaining efficient batch schedulers for Grid systems. A careful design of the cMA methods and operators for the problem yielded to an efficient and robust implementation. Our experimental study, based on a known static benchmark for the problem, shows that this heuristic approach is able to deliver very high quality planning of jobs to Grid nodes and thus it can be used to design efficient dynamic schedulers for real Grid systems. Such dynamic schedulers can be obtained by running the cMA-based scheduler in batch mode for a very short time to schedule jobs arriving to the system since the last activation of the cMA scheduler.

1. Introduction

One of the main motivations of the Grid computing paradigm has been the computational need for solving many complex problems from science, engineering, and business such as hard combinatorial optimization problems, protein folding, financial modelling, etc. [17]. Since the early definitions of Computational Grids by Foster and other researchers [12, 13], a fast development has taken place leading to better understanding of the Grid issues as well as the development of Grid infrastructures and middleware. Nowadays, Grid computing is a common approach in the development of large scale distributed applications from academia and industry. Early successful applications of this paradigm are NetSolve [9], and applications from MetaNeos Project such as stochastic programming and optimization [18, 20], which used the enormous computing power of computational grids.

One key issue in Computational Grids is the allocation of jobs (applications) to Grid resources. The resource allocation problem is known to be computationally hard as it is a generalization of the standard scheduling problem. In fact, due to the complex nature of Computational Grids, job scheduling is much more difficult than its standard version for sequential or LAN computation environments. Some of the features of the Computational Grids that make the problem challenging are the high degree of heterogeneity of resources, their connection with heterogenous networks, the high degree of dynamics, the large scale of the problem regarding number of jobs and resources, and other features related to existing local schedulers, policies on resources, etc.

Many approaches for scheduling in Grid applications use queuing systems or *ad hoc* schedulers that use specific knowledge of the underlying grid infrastructure to achieve an efficient resource allocation (e.g. Condor-G, Nimrod/G) [1, 14]. However, due to the dynamics and large-scale of Grids, these approaches cannot deal with the complexity of the problem. For instance, researchers from MetaNeos Project reported that for solving the difficult instance `Nug30` of the Quadratic Assignment Problem, a queue of thousands of pending jobs had to be managed for a grid of roughly 1000 machines. Moreover, most of current approaches try to optimize the throughput of the system through the minimization of the makespan. In a global computing environment, however, other objectives must be accomplished such as flowtime, which is an indicator of the QoS of the system. In fact, job scheduling in Computational Grids is multi-objective in its general formulation and therefore optimization approaches that could tackle many conflicting objectives are imperative.

1-4244-0910-1/07/$20.00 ©2007 IEEE

Meta-heuristic approaches have shown their effectiveness for a wide variety of hard combinatorial problems and also for multi-objective optimization problems. These approaches are in fact the best choice in practice and hence such approaches have already started to be examined. Among these approaches, population based heuristics, such as Genetic Algorithms (GAs), are reported for the job scheduling problem [1, 8, 10, 19, 21]. In this work we address the use of Cellular Memetic Algorithms (cMAs) [2, 3, 4, 5, 15] for efficiently scheduling jobs to Grid resources. cMAs are population-based algorithms that maintain a structured population as opposed to GAs or MAs of unstructured population. Research on cMAs has shown that, due to the structured population, this family of algorithms is able to better control the tradeoff between the exploitation and exploration of the solution space with respect to other non-structured algorithms. It should be noted that this feature is very important if high quality solutions are to be found in a very short time. This is precisely the case of the job scheduling in Computational Grids whose highly dynamic nature makes indispensable the use of schedulers that would be able to deliver high quality planning of jobs to resources very fast in order to deal with the changes of the Grid. On the other hand, population-based heuristics are potentially good also for solving complex problems in the long run providing, for many problems, near optimal solutions. This is another interesting feature to explore regarding the use of cMAs for the job scheduling problem. The evidence reported in the literature that cMAs are capable to maintain a high diversity of the population in many generations suggests that cMAs could be appropriate for scheduling jobs that periodically arrive in the Grid system since in this case the Grid scheduler would dispose longer intervals of time to compute the planning of jobs to Grid resources. Finally, cMAs are used here to solve the bi-objective case of the job scheduling, namely makespan and flowtime are simultaneously optimized. The proposed algorithm has been tested using a benchmark of instances proposed by Braun et al. [7], and its performance is contrasted to the performance of other population-based approaches for the problem such as the GA implementation of Braun et al. [7], a GA implementation by Carretero and Xhafa [10] and a Struggle GA by Xhafa [21]. Moreover, we studied the robustness of our cMA implementation since robustness is a desired property of Grid schedulers, which are very changing in nature. Because the cMA scheduler is able to deliver very high quality planning of jobs to Grid nodes, it can be used to design efficient dynamic schedulers for real Grid systems. Such dynamic schedulers are obtained by running the cMA-based scheduler in batch mode for a very short time to schedule jobs arriving in the systems since the last activation of the cMA scheduler.

The paper is organized as follows. We give in Section 2 a description of the job scheduling in computational grids. The cMAs and their particularization for job scheduling in Grids are given in Section 3. In Section 4 we give the values of parameters obtained from the tuning process. Some computational results as well as their evaluation are presented in Section 5. We end in Section 6 with some conclusions.

2. Problem description

Job scheduling in Computational Grids is a family of problems that capture most important needs of Grid applications for efficiently allocating jobs to resources in a global, heterogenous, and dynamic environment. Therefore, several versions of the problem can be formulated according to the needs of such applications. In this work we consider the version of the problem that arises quite frequently in parameter sweep applications, such as Monte-Carlo simulations [11]. In these applications, many jobs with almost no interdependencies are generated and submitted to the Grid system. In fact, more generally, the scenario in which the submission of independent jobs to a Grid system is quite natural given that Grid users independently submit their jobs or applications to the Grid system and expect an efficient allocation of their jobs/applications. We notice that the efficiency means that we are interested to allocate jobs as fast as possible and to optimize two conflicting criteria: *makespan* and *flowtime*. These two optimization criteria are among the most important of a grid system. Indeed, for a Grid system, makespan measures the productivity (throughput) of the system and the flowtime measures its *QoS*.

In our scenario, jobs are originated from different users/applications, have to be completed in unique resource unless it drops from the Grid due to its dynamic environment (*non-preemptive* mode), are independent of each other and could have their hardware and/or software requirements over resources. On the other hand, resources could dynamically be added/dropped from the Grid, can process one job at a time, and have their own computing characteristics regarding consistency of computing. More precisely, we use the Expected Time to Compute (ETC) model by Braun et al. [7] to formalize the instance definition of the problem as follows:

- A *number* of independent (user/application) *jobs* to be scheduled.
- A *number* of heterogeneous *machines* candidates to participate in the planning.
- The *workload* of each job (in millions of instructions).
- The *computing capacity* of each machine (in *mips*).

- Ready time $ready_m$ indicates when machine m will have finished the previously assigned jobs.
- The Expected Time to Compute (ETC) matrix ($nb\_jobs \times nb\_machines$) in which $ETC[i][j]$ is the expected execution time of job i in machine j.

Optimization criteria. We consider the job scheduling as a bi-objective optimization problem, in which both makespan and flowtime are simultaneously minimized. These criteria are defined as follows:

- *Makespan* (the finishing time of latest job) defined as $\min_S \max\{F_j : j \in Jobs\}$,
- $Flowtime$ (the sum of finishing times of jobs), that is, $\min_S \sum_{j \in Jobs} F_j$,

where F_j is the finishing time of job j in schedule S.

For a given a schedule, it is quite useful to define the *completion time* of a machine which indicates the time in which the machine will finalize the processing of the previous assigned jobs as well as of those already planned for the machine. Formally, for a machine m and a schedule S, the completion time of m is defined as follows:

$$completion[m] = ready_m + \sum_{j \in S^{-1}(m)} ETC[j][m]. \quad (1)$$

We can then use the values of completion times to compute the makespan as follows:

$$makespan = \max\{completion[i] \mid i \in Machines'\}. \quad (2)$$

In order to deal with the simultaneous optimization of the two objectives we have used a simple weighted sum function of makespan and flowtime, which is possible since both parameters are measured in the same unit (time units). However, the makespan and flowtime values are in incomparable ranges, since flowtime has a higher magnitude order over makespan, and its difference increases as more jobs and machines are considered. For this reason, the value of mean flowtime, $flowtime/nb\_machines$, is used instead of flowtime. Additionally, both values are weighted in order to balance their importance. Fitness value is thus calculated as:

$$fitness = \lambda \cdot makespan + (1-\lambda) \cdot mean\_flowtime, \quad (3)$$

where λ has been *a priori* fixed after a preliminary tuning process (see Section 4).

3 A Cellular Memetic Algorithm for Resource Allocation in Grid Systems

3.1 Cellular Memetic Algorithms

In Memetic Algorithms (MAs) the population of individuals could be unstructured or structured. In the former, there is no relation between the individuals of the population while in the latter individuals can be related to only some other specific individuals of the population. The structured MAs are usually classified into *coarse-grained* model and *fine-grained* (Cellular MAs) model [2, 4, 5, 15]. In Cellular MAs the individuals of the population are spatially distributed forming neighborhoods and the evolutionary operators are applied to neighbor individuals making thus cMAs a new family of evolutionary algorithms. As in the case of other evolutionary algorithms, cMAs are high level algorithms whose description is independent of the problem being solved. Thus, for the purposes of this work, we have considered the cMA template given in Algorithm 1.

As it can be seen, this template is quite different from the canonical cGA approximation [4, 5], in which individuals are updated in a given order by applying the recombination operator to the two parents and the mutation operator to the obtained offspring. In the case of the algorithm proposed in this work, mutation and recombination operators are applied to individuals independently of each other, and in different orders. This model was adopted after a previous experimentation, in which it performed better than the cMA following the canonical model for the studied problems. After each recombination (or mutation), a local search step is applied to the newly obtained solution, which is then evaluated. If this new solution is better than the current one, it replaces the latter in the population. This process is repeated until a termination condition is met.

Algorithm 1 A Cellular MA template.

Initialize the mesh of **n** individuals P(t=0);
Initialize permutations *rec_order* and *mut_order*;
For each $i \in P$, **LocalSearch**(i);
Evaluate(P);
while not stopping condition **do**
  for j = 1 . . .#recombinations **do**
    SelectToRecombine $S \subseteq N_{P[rec\_order.current]}$;
    i′ = **Recombine**(S);
    LocalSearch(i′); **Evaluate**(i′);
    Replace $P[rec\_order.current]$ by i′;
    rec_order.next();
  end for
  for j = 1 . . .#mutations **do**
    i = $P[mut\_order.current()]$;
    i′ = **Mutate**(i);
    LocalSearch(i′); **Evaluate**(i′);
    Replace $P[rec\_order.current]$ by i′;
    rec_order.next();
  end for
  Update *rec_order* and *mut_order*;
end while

3.2 Application of the cMA to job scheduling

In order to solve the job scheduling problem through cMAs, we have to particularize the template given in Algorithm 1 with the specific knowledge of the problem at hand. We notice that the template is generic and flexible in a way that several implementations could be derived by fixing the *ingredients* of the template according to the concrete problem aiming at identifying the best implementation. We give next the description of the cMA particularization for job scheduling.

Population's topology and neighborhood structure. The topology of the population is a two-dimensional toroidal grid of `pop_height`×`pop_width` size. Regarding the neighborhood patterns, several well-known patterns are used for this work: *L5* (5 individuals), *L9* (9 individuals), *C9* (9 individuals) and *C13* (13 individuals) (see Fig. 1). In our experimental study we identified the pattern that leads to the best performance for the job scheduling problem. It should be noticed that both the topology of the population and the neighborhood pattern are very important in deciding the selective pressure of the algorithm and therefore have a direct influence on the tradeoff between exploration and exploitation of the algorithm [3, 6].

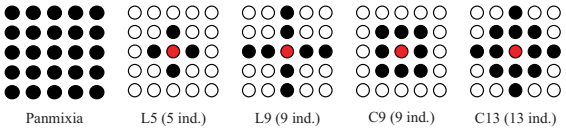

Figure 1. Neighborhood patterns.

Cell updating. Unlike many unstructured MAs, in cMAs the population is kept constant by applying cell updating mechanisms by which an individual of the population is updated with a new offspring obtained by either recombination or mutation process (see later for the definition of these two operators). Two well-known methods of cell updating are the synchronous and asynchronous updating. For the purpose of this work, we have considered the asynchronous updating since it is less computationally expensive and usually shows a good performance in a very short time [15], which is interesting for the scheduling problem given the dynamic nature of Grid systems. In the asynchronous mode, cell updating is done sequentially (an individual is aware of other neighbor individual updates during the same iteration). The following asynchronous mechanisms have been implemented and experimentally studied for our job scheduling problem:

- **Fixed Line Sweep (FLS)**: The individuals of the *grid* are updated in a sequential order row by row.
- **Fixed Random Sweep (FRS)**: The sequence of cell updates is at random. This sequence is defined at the beginning of the algorithm and it is the same during the cMA iterations.
- **New Random Sweep (NRS)**: At each itertion, a new cell update sequence (at random) is applied.

It should be noted that recombination and mutation are independent processes in cMAs (cf. `rec_order` and `mut_order` in the cMAs template) and therefore different update orders are used.

cMA methods and operators

The performance of any cMA heavily depends on the design and implementation of the methods and operators, which, on the other hand, depend on the chosen chromosome representation.

Solution representation. A feasible solution, *schedule*, is represented as a vector of size nb_jobs in which its jth position (an integer value) indicates the machine where job j is assigned: $schedule[j] = m, m \in \{1, \ldots, nb\_machines\}$.

Fitness. The individual fitness is computed by simultaneously minimizing makespan and flowtime of the schedule: $\min\ \lambda \cdot makespan + (1 - \lambda) \cdot mean\_flowtime$, in which $\lambda = 0.75$ was experimentally fixed.

Population initialization. One individual is generated using the *Longest Job to Fastest Resource - Shortest Job to Fastest Resource (LJFR-SJFR)* heuristic [1]. The rest are randomly obtained from the first individual by large perturbations. The LJFR-SJFR method has been chosen because it tries to simultaneously minimize both makespan and flowtime. LJFR minimizes the makespan and is alternated with the SJFR which minimizes the flowtime. The method starts by increasingly sorting jobs w.r.t. their workload. At the beginning, the first $nb\_machines$ longest jobs are assigned to the $nb\_machines$ idle machines (the longest job to the fastest machine and so on). For the remaining jobs, at each step the fastest machine (that has finished its jobs) is chosen to which is assigned alternatively either the shortest job (SJFR) or the longest job (LJFR).

Selection operator for recombination. We have used the N-Tournament operator in which N individuals compete in the selection process.

Recombination operator. The One-Point recombination of two individuals has been chosen. This operator consists of splitting the two chromosomes into two parts (in a randomly selected point), and joining each part of one parent chromosome with the other part of the chromosome of the second parent.

Mutation operator. The mutation is done by *rebalancing* of machine loads of the given schedule. The load factor of a machine m is defined as load_factor(m) = completion$[m]$/makespan (load_factor$(m) \in (0, 1]$). The idea behind this choice is that in a schedule, some machines could be overloaded (when its completion time is equal to the current makespan –load_factor$(m) = 1$–) and some others less overloaded (regarding the overloaded machines, we sort the machines in increasing order of their completion times and 25% first machines are considered less overloaded), in terms of their completion times. It is useful then to mutate the schedule by a load balancing mechanism, which transfers a job assigned to an overloaded machine to another less loaded machine.

Replacement strategy. An individual is replaced by the newly generated one (its offspring) only if the latter is better than the former in terms of their fitness value.

Local search methods. Local search is a proper feature of Memetic Algorithms. As it can be seen from the template of Algorithm 1, each individual is improved by a local search. Improvement of the descendants is thus done not only by means of genetic information but also by local improvements. The presence of this local search method in the algorithm does not increase selection pressure too much due to the exploration capabilities intrinsic to the cellular model. Three local search methods have been implemented and experimentally studied. These are the Local Move (LM), Steepest Local Move (SLM) and Local Minimum Completion Time Swap (LMCTS). The first method is similar to the mutation operator (a randomly chosen job is transferred to a new randomly chosen machine). In SLM method, the job transfer is done to the machine that yields the best improvement in terms of the reduction of the completion time and in LMCTS method, two jobs assigned to different machines are swapped; the pair of jobs that yields the best reduction in the completion time is applied.

4 Tuning of parameters

After implementing the cMA template and the specific methods and operators, tuning the parameters is a crucial step in order to achieve a good performance. As it can be seen from the previous section, many parameters are involved in the cMA, whose values influence in a straightforward way on the search process. The tuning process was done by using randomly generated instances of the problem according to the ETC matrix model. In this way we would expect a robust performance of our cMA implementation since no specific instance knowledge is used in fixing the values of the parameters. An extensive experimental study was done in order to identify the best configuration for the cMA. Thus, we experimentally studied the choice of the local search method, the topology/neighborhood pattern, selection operator and the cell update orders. We give in Figs. 2, 3, 4 and 5, the graphical representation for the makespan reduction of the considered local search methods, neighborhood patterns, selection mechanisms, and the cell update orders, respectively. The results are obtained after making 20 independent runs in AMD K6 450MHz computers.

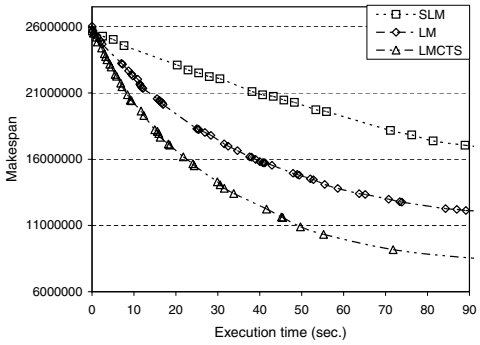

Figure 2. Makespan reduction obtained with three local search methods.

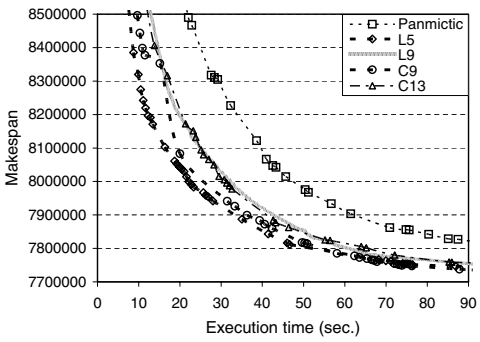

Figure 3. Makespan reduction obtained with different neighborhood patterns.

Additive evaluation. From the graphical representations we can easily observe that:

- LMCTS method performs best among the three considered local search methods. In fact, a clear difference in the behavior of the considered local search methods is observed, though all of them provide an accentuated reduction in the makespan value (see Fig. 2).

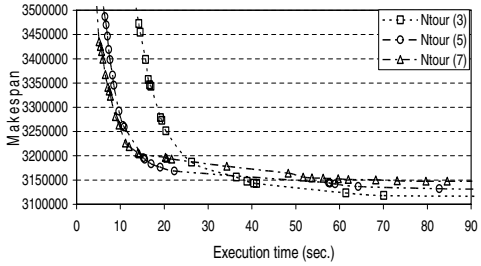

Figure 4. Makespan reduction obtained with different values of N-Tournament selection.

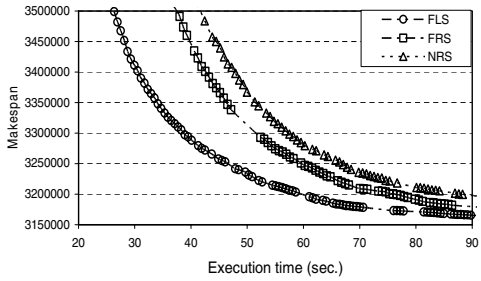

Figure 5. Makespan reduction obtained with different recombination orders.

- For the neighborhood patterns, a similar behavior is observed, except for the panmixia pattern, which as expected performed worse. L5 and C9 show the best behavior. L5 yields a very fast reduction however C9 performs better in the "long run" (see Fig. 3).
- A similar behavior was observed for values of $N = 3$, 5 and 7 of the N-Tournament selection. The best performance is obtained for the value of $N = 3$ (see Fig. 4).
- As regards to the cell updating for the recombination operator, the three considered mechanisms performed similarly, the FLS being the best performer (see Fig. 5).

The resulting configuration, which is used then to obtain the computational results, is given in Table 1.

5. Computational results

In this section we present some computational results obtained for the benchmark of instances by Braun et al. [7] for distributed heterogenous systems.

Benchmark description. The instances of this benchmark are classified into 12 different types of ETC matrices, each of them consisting of 100 instances, according to three parameters: job heterogeneity, machine heterogeneity and consistency. Instances are labelled as $u\_x\_yyzz.k$ where:

- u stands for uniform distribution (used in generating the matrix).

Table 1. Values of the parameters.

max_exec_time	(max 90s)
population_height	5
population_width	5
nb_solutions_to_recombine	3
nb_recombinations	25
nb_mutations	12
start_choice	LJFR-SJFR Longest / Shortest Job to Fastest Resource
neighborhood_pattern	C9
recombination_order	FLS (Fixed Line Sweep)
mutation_order	NRS (New Random Sweep)
recombine_choice	One-Point recombination
recombine_selection	3-Tournament
mutate_choice	Rebalance
local_search_choice	LMCTS (Local Minimum Completion Time Swap)
nb_local_search_iterations	5
add_only_if_better	true
λ	0.75

- x stands for the type of consistency (c–consistent, i–inconsistent, and s means semi-consistent). An ETC matrix is considered consistent when, if a machine m_i executes job j faster than machine m_j, then m_i executes all the jobs faster than m_j. Inconsistency means that a machine is faster for some jobs 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 jobs (hi means high, and lo means low).
- zz indicates the heterogeneity of the resources (hi means high, and lo means low).

Note that all instances consist of 512 jobs and 16 machines. We report computational results for 12 instances, which are made up of three groups of four instances each. These three groups represent different Grid scenarios regarding the computing capacity. The first group corresponds to consistent ETC matrices (for each of them combinations between *low* and *high* are considered), the second represent instances of inconsistent computing capacity and the third one to semi-consistent computing capacity.

Results are obtained by running the cMA implementation for 90 seconds (a single run) and 10 runs per instance.

Results for makespan parameter. We give in Table 2 the computational results[1] for the makespan objective, where the first column indicates the name of the instance, the second one the best makespan obtained by Braun et al.'s GA, the third one the best makespan (out of 10 runs) by our cMA implementation, and the last column gives the difference (in %) between the best makespan reported by the Braun et al. GA and cMA for each instance.

[1] Values are in arbitrary time units.

Table 2. Comparison of makespan values: Braun et al.'s GA and cMA.

Instance	Braun et al. GA	cMA	Δ (%)
u_c_hihi.0	8050844.5	**7700929.751**	4.35
u_c_hilo.0	156249.2	**155334.805**	0.59
u_c_lohi.0	258756.77	**251360.202**	2.86
u_c_lolo.0	5272.25	**5218.18**	1.03
u_i_hihi.0	**3104762.5**	3186664.713	2.57
u_i_hilo.0	**75816.13**	75856.623	0.88
u_i_lohi.0	**107500.72**	110620.786	2.82
u_i_lolo.0	**2614.39**	2624.211	0.37
u_s_hihi.0	4566206	**4424540.894**	3.10
u_s_hilo.0	98519.4	**98283.742**	0.24
u_s_lohi.0	130616.53	**130014.529**	0.46
u_s_lolo.0	3583.44	**3522.099**	1.71

Table 3. Comparison of makespan values: GAs in [10, 21] and cMA.

Instance	GA (Carretero&Xhafa)	Struggle GA (Xhafa)	cMA
u_c_hihi.0	7752349.37	7752689.08	**7700929.751**
u_c_hilo.0	155571.80	156680.58	**155334.805**
u_c_lohi.0	**250550.86**	253926.06	251360.202
u_c_lolo.0	5240.14	5251.15	**5218.18**
u_i_hihi.0	**3080025.77**	3161104.92	3186664.713
u_i_hilo.0	76307.90	**75598.48**	75856.623
u_i_lohi.0	**107294.23**	111792.17	110620.786
u_i_lolo.0	**2610.23**	2620.72	2624.211
u_s_hihi.0	**4371324.45**	4433792.28	4424540.894
u_s_hilo.0	983334.64	98560.04	**98283.742**
u_s_lohi.0	**127762.53**	130425.85	130014.529
u_s_lolo.0	3539.43	3534.31	**3522.099**

Table 4. Comparison of flowtime values obtained with LJFR-SJFR and cMA.

Instance	LJFR-SJFR	cMA	Δ (%)
u_c_hihi.0	2025822398.665	**1037049914.209**	48.8
u_c_hilo.0	35565379.565	**27487998.874**	22.7
u_c_lohi.0	66300486.264	**34454029.416**	48.0
u_c_lolo.0	1175661.381	**913976.235**	22.2
u_i_hihi.0	3665062510.364	**361613627.327**	90.0
u_i_hilo.0	41345273.211	**12572126.577**	69.0
u_i_lohi.0	118925452.958	**12707611.511**	89.0
u_i_lolo.0	1385846.186	**439073.652**	89.0
u_s_hihi.0	2631459406.501	**513769399.117**	80.0
u_s_hilo.0	35745658.309	**16300484.885**	54.0
u_s_lohi.0	86390552.327	**15179363.456**	82.0
u_s_lolo.0	1389828.755	**594665.973**	57.0

Table 5. Comparison of flowtime values obtained by Struggle GA and cMA.

Instance	Struggle GA (Xhafa)	cMA	Δ (%)
u_c_hihi.0	1039048563	**1037049914.209**	0.19
u_c_hilo.0	27620519.9	**27487998.874**	0.48
u_c_lohi.0	34566883.8	**34454029.416**	0.32
u_c_lolo.0	917647.31	**913976.235**	0.40
u_i_hihi.0	379768078	**361613627.327**	4.78
u_i_hilo.0	12674329.1	**12572126.577**	0.81
u_i_lohi.0	13417596.7	**12707611.511**	5.29
u_i_lolo.0	440728.98	**439073.652**	0.38
u_s_hihi.0	524874694	**513769399.117**	2.12
u_s_hilo.0	16372763.2	**16300484.885**	0.44
u_s_lohi.0	15639622.5	**15179363.456**	2.94
u_s_lolo.0	598332.69	**594665.973**	0.61

The comparison of the cMA with two other versions of GAs [10, 21] is given in Table 3.

Results for flowtime parameter. Computational results for flowtime parameter are given in Table 4 where we also give the flowtime value obtained by the *ad hoc* heuristic LJFR-SJFR aiming at identifying the improvement obtained by the cMA over the initial flowtime. Next, in Table 5 we compare the flowtime values against the ones obtained by the Xhafa's Struggle GA [21].

5.1 Evaluation and discussion

As it can be seen from Tables 2 and 3, cMA performs better than Braun et al.'s GA for all but inconsistent computing instances. This observation is interesting if the Grid characteristics were known in advance, since cMA seems to be more appropriate for consistent and semi-consistent Grid scenarios. Additionally, when compared against two other versions of GAs, namely Carretero&Xhafa's GA [10] and Xhafa's Struggle GA [21] (both of them use the same simultaneous approach), cMA obtains better schedules than the two compared GAs for half of the considered instances, and for the rest of the instances, the solutions found by cMA have a similar quality than the best of the other two GAs. Also, we have observed that the standard deviation of the best makespan from the averaged makespan is very small (roughly 1%), which is an indicator of the robustness of the cMA implementation.

Regarding the flowtime values, we observed that a significant improvement was obtained by the cMA over the initial flowtime value of the LJFR-SJFR method. The good performance of the cMA regarding flowtime value was next confirmed by comparing with the Struggle GA [21] where, as it can be seen from Table 3, cMA outperforms Struggle GA for all considered instances.

6. Conclusions and future work

In this work we have presented an implementation of Cellular Memetic Algorithms (cMAs) for the problem of job scheduling in Computational Grids when both makespan and flowtime are simultaneously minimized. cMAs are a family of population-based metaheuristics that have turned out to be an interesting scheduler due to their structured population, which allows to better control the

tradeoff between the exploitation and exploration of the search space. We have implemented and experimentally studied several methods and operators of cMA for the job scheduling in Grid systems, which is a challenging problem in today's large-scale distributed applications.

Our experimental study showed that cMAs are a good choice for scheduling jobs in Computational Grids given that they are able to deliver high quality plannings in a very short time. This last feature makes cMAs useful to design efficient dynamic schedulers for real Grid systems, which can be obtained by running the cMA-based scheduler in batch mode for a very short time to schedule jobs arriving in the systems since the last activation of the cMA scheduler.

In our future work we would like to better understand some issues raised by the experimental study such as the good performance of the cMA for consistent and semi-consistent Grid Computing environments and the not so good performance for inconsistent computing instances. Also, we plan to extend the experimental study by considering other operators and methods as well as using grid simulator packages to study the performance of cMA-based scheduler(s) in longer periods of time. Another interesting line for future research are to tackle the problem with a multi-objective algorithm in order to find a set of non-dominated solutions to the problem. Finally, evaluating our cMA with larger size grid instances is being done using instances generated according to the ETC model.

Acknowledgments

Research partially supported by ASCE Project TIN2005-09198-C02-02, Project FP6-2004-IST-FETPI (AEOLUS) and MEC TIN2005-25859-E. E. Alba and B. Dorronsoro acknowledge that this work has been partially funded by the Spanish MEC and FEDER under contract TIN2005-08818-C04-01 (the OPLINK project). We are grateful to Bernat Duran for his technical assistance.

References

[1] A. Abraham, R. Buyya and B. Nath. Nature's Heuristics for Scheduling Jobs on Computational Grids, *The 8th IEEE International Conference on Advanced Computing and Communications (ADCOM 2000)* India, 2000.

[2] E. Alba and M. Tomassini. Parallelism and evolutionary algorithms, *IEEE Transactions on Evolutionary Computation*, 6(5), 443-462, 2002.

[3] E. Alba, J.M. Troya. Cellular Evolutionary Algorithms: Evaluating the Influence of Ratio. Proc. of the 6th International Conference on Parallel Problem Solving from Nature, LNCS, vol. 1917, 29-38, 2000.

[4] E. Alba, B. Dorronsoro, and H. Alfonso. Cellular Memetic Algorithms Evaluated on SAT, *XI Congreso Argentino de Ciencias de la Computación (CACIC)*, 2005.

[5] E. Alba, B. Dorronsoro, and H. Alfonso. Cellular Memetic Algorithms, *Journal of Computer Science and Technology*, 5(4), 257-263, 2006.

[6] E. Alba and B. Dorronsoro. The exploration/exploitation tradeoff in dynamic cellular evolutionary algorithms. *IEEE Trans. on Evolutionary Computation*, 9(2):126–142, 2005.

[7] H. Braun, T. D.and Siegel, N. Beck, L. Bölöni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, and B. Yao. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing*, 61(6):810–837, 2001.

[8] R. Buyya, D. Abramson, and J. Giddy. Nimrod/G: An architecture for a resource management and scheduling system in a global computational grid. In *The 4th Int. Conf. on High Performance Computing, Asia-Pacific Region, China*, 2000.

[9] H. Casanova and J. Dongarra. Netsolve: Network enabled solvers. *IEEE Computational Science and Engineering*, 5(3):57–67, 1998.

[10] J. Carretero and F. Xhafa. Using genetic algorithms for scheduling jobs in large scale grid applications, *J. of Technological and Economic Development –A Research Journal of Vilnius Gediminas Technical University*, 12(1):11–17, 2006.

[11] H. Casanova, A. Legrand, D. Zagorodnov and F. Berman. Heuristics for Scheduling Parameter Sweep Applications in Grid Environments, *Heterogeneous Computing Workshop*, 349-363, 2000.

[12] I. Foster and C. Kesselman. *The Grid - Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, 1998.

[13] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid. *Int. J. of Supercomputer Applications*, 15(3), 2001.

[14] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke. Condor-G: A computation management agent for multi-institutional grids. In *Proceedings of the 10th IEEE Symposium on High Performance Distributed Computing*, 2001.

[15] M. Giacobini, M. Tomassini, A.G.B. Tettamanzi and E. Alba. Selection Intensity in Cellular Evolutionary Algorithms for Regular Lattices. *IEEE Transactions on Evolutionary Computation*. 9(5):489–505, October 2005.

[16] P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Tech. report 826, California Institute of Technology, USA, 1989.

[17] F. Luna, A.J. Nebro, E. Alba. Observations in Using Grid-enabled Technologies for Solving Multi-objective Optimization Problems. *Parallel Computing*. 32:377-393. June 2006.

[18] L. Linderoth and S. Wright. Decomposition algorithms for stochastic programming on a computational grid. *Computational Optimization and Applications (Special issue on Stochastic Programming.)*, 24:207–250, 2003.

[19] V. D. Martino and M. Mililotti. Sub optimal scheduling in a grid using genetic algorithms. *Parallel Computing*, 30:553–565, 2004.

[20] S. Wright. Solving optimization problems on computational grids. *Optima*, 65, 2001.

[21] F. Xhafa. An experimental study on GA replacement operators for scheduling on grids, In *The 2nd International Conference on Bioinspired Optimization Methods and their Applications (BIOMA 2006)*, pp. 121-130, Ljubljana, Slovenia, 9-10 October 2006.