Parallel Memetic Algorithms for Independent Job Scheduling in Computational Grids

Fatos Xhafa and Bernat Duran

Polytechnic University of Catalonia, Department of Languages and Informatics Systems, C/Jordi Girona 1-3, 08034 Barcelona, Spain
{fatos,bduran}@lsi.upc.edu

Summary. In this chapter we present parallel implementations of Memetic Algorithms (MAs) for the problem of scheduling independent jobs in computational grids. The problem of scheduling in computational grids is known for its high demanding computational time. In this work we exploit the intrinsic parallel nature of MAs as well as the fact that computational grids offer large amount of resources, a part of which could be used to compute the efficient allocation of jobs to grid resources.

The parallel models exploited in this work for MAs include both fine-grained and coarse-grained parallelization and their hybridization. The resulting schedulers have been tested through different grid scenarios generated by a grid simulator to match different possible configurations of computational grids in terms of size (number of jobs and resources) and computational characteristics of resources. All in all, the result of this work showed that Parallel MAs are very good alternatives in order to match different performance requirement on fast scheduling of jobs to grid resources.

1 Introduction

In this chapter we present several parallel implementation of Memetic Algorithms (MAs), both unstructured and structured, for the problem of independent job scheduling to grid resources. The scheduling problem is at the heart of any Computational Grid (CG) [9, 10]. Due to this, the job scheduling problem is increasingly receiving the attention of researchers from the grid computing community with the objective of designing schedulers for high performance grid-enabled applications. The independent job scheduling on computational grids is computationally hard. Therefore the use of heuristics is the *de facto* approach in order to cope in practice with its difficulty. Thus, the evolutionary computing research community has already started to examine this problem [1, 5, 14, 17, 7]. Yet, the parallelization of meta-heuristics, in particular of MAs [16], for the resolution of the problem has not been explored.

This work builds upon previous work on MAs for independent job scheduling [18]. One of the most advantageous characteristics of the family of Evolutionary Algorithms (EAs) is the intrinsic parallel nature of their structure.

Xhafa, F.; Duran, B. Parallel memetic algorithms for independent job scheduling in computational grids. A: "Recent advances in evolutionary computation for combinatorial optimization". Berlín: Springer, 2008, p. 219-239.
The final authenticated version is available online at https://doi.org/10.1007/978-3-540-70807-0

Holland [12] in his early works introduced the first ideas for defining a parallel structure for EAs. The main observation here is that the algorithms based on populations of individuals could have a very complex structure, yet easily decomposable in smaller structures. This decomposition could be very useful to distribute the work of the algorithm to different processors. The objective of parallelizing EAs is essentially to distributed the burden of work during the search to different processors in such a way that the overall search time within the same (sequential) exploration is reduced. Therefore, the parallelization of EAs, in general, and MAs, in particular, could be beneficial for the resolution of the independent job scheduling in computational grids. Moreover, parallelizing EAs could imply not only the reduction of resolution time or better quality of solutions; it is also a source for new ideas in re-structuring the EA algorithm, differently from its original sequential structure, which could eventually lead to better performance of the algorithm.

In this chapter three different models of parallelizing unstructured MAs and Cellular MAs (here after refereed to as MAs and cMAs, resp.) are studied: (a) the model of independent searches, referred to as Independent Runs (IR) model for MAs; (b) the Master-Slave (MS) for MAs; and, (c) parallel hybridization among the coarse-grained and fine-grained models (also referred to as Two Level Granulation) for cMAs. The IR model consists of simultaneous execution of independent searches. In the MS model, the search algorithm is executed by a *master* processor, which delegates independent sub-tasks of high computational cost to the rest of processors (*slaves*). In the context of MAs, different slave processors could apply different local search procedures [15, 13] on the individuals of the population. Finally, hybrid parallel implementation is done for cMAs by combining the coarse-grained model, at a first hierarchical level, and the fine-grained model, at a second hierarchical level.

The proposed parallel MAs are implemented in C++ and MPI using a skeleton for MAs [3], extended for the purposes of this work. The implementations are extensively tested, on the one hand, to identify a set of appropriate values for the search parameters and, on the other, to compare the results for the makespan parameter obtained by different parallel implementations and the corresponding sequential versions. To this end we have used different grid scenarios generated using a grid simulator [19] to match different possible configurations of computational grids in terms of size (number of jobs and resources) and computational characteristics of resources.

The remainder of the chapter is organized as follows. We give in Section 2 the description of independent job scheduling problem. An overview on the taxonomy of parallel models for meta-heuristics and their application to MAs and cMAs is given in Section 3. The implementation of the parallel MAs is given in Section 4. The experimental study and some relevant computational results are presented in Section 5. We conclude in Section 6 with most important aspects of this work and indicate directions for future work.

2 Job Scheduling on Computational Grids

For the purposes of this work, we consider the following scheduling scenario: the tasks being submitted to the grid are independent and are not preemptive, that is, they cannot change the resource they has been assigned to once their execution is started, unless the resource is dropped from the grid.

The problem of job scheduling is formulated by using Expected Time to Compute (ETC) matrix model (see e.g. Braun et al. [4]). In this model, we have an estimation or prediction of the computational load of each task, the computing capacity of each resource, and an estimation of the prior load of the resources. In the Expected Time to Compute matrix $ETC[t][m]$ indicates the expected time to compute task t in resource m. The entries $ETC[t][m]$ could be computed, for instance, by dividing the workload of task t by the computing capacity of resource m. An instance of the problem is defined as follows: (a) A *number* of independent (user/application) *tasks* that must be scheduled. Any task has to be processed entirely in unique resource; (b) A *number* of heterogeneous *machines* candidates to participate in the planning; (c) The *workload* (in millions of instructions) of each task; (d) The *computing capacity* of each machine (in *mips*); (e) The time $ready_m$ when the machine will have finished the previously assigned tasks; and, (f) The expected time to compute ETC matrix of size $nb\_tasks \times nb\_machines$.

In this work the objective is to minimize the makespan parameter (the problem is multi-objective in its general formulation), that is, Makespan is a the time when finishes the latest task: $makespan: \min_{S_i \in Sched}\{\max_{j \in Jobs} F_j\}$, where F_j denotes the time when task j finalizes, $Sched$ is the set of all possible schedules and $Jobs$ the set of all jobs to be scheduled. For the purposes of computation, it is better to express makespan in terms of the *completion time* of machines. Let $completion[m]$ indicates the time in which machine m will finalize the processing of the previous assigned tasks as well as of those already planned tasks for the machine. Its value is calculated as follows:

$$completion[m] = ready\_times[m] + \sum_{\{j \in Tasks \mid schedule[j]=m\}} ETC[j][m].$$

Then, $makespan = \max\{completion[i] \mid i \in Machines\}$.

3 Overview of the taxonomy of the parallel models for Memetic Algorithms

In this work we are based on the classification of the parallel models according to the taxonomy introduced by Crainic and Toulose [8] (see Fig. 1).

Low level parallelism: The model of low level parallelism is based on the simultaneous execution of the operations of the execution flow of the

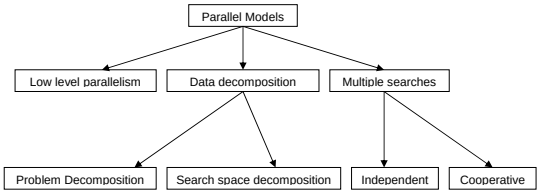

Fig. 1. Classification of Parallel Models.

sequential method that can be executed in different processors. In this type of parallelism belongs the Master-Slave (MS) model, in which the search method is organized in a main process (*master*) that carries out the search and several processes (*slaves*), which execute the different sub-tasks of the search that the master delegates to them. This model is interesting for EAs given that in such algorithms several independent operations can be easily identified and performed in parallel.

Data decomposition: This type of parallelism considers the data decomposition for the resolution of the problem. Two classes of parallelism can be distinguished here according to whether the problem is decomposed into smaller problems or whether the space of solutions is divided into smaller regions (each processor solves exactly the same problem but treating a smaller data subset): (a) *Problem decomposition*: the problem is divided into subproblems that are assigned to different processes, the solutions of the subproblems are later combined into the solution of the original problem; and, (b) *Search space decomposition*: the decomposition is carried out in the space of solutions by assigning disjointed regions to slave processors.

3.1 Multiple searches

The parallel schemes in this category are those that perform several simultaneous searches of the same space of solutions. The searches can be more or less independent among them, but in any case they work as independent entities in their own and thus different searches need not to have the same configuration. The strategy of multiple searches differs thus according to the degree of dependence established among them.

Independent searches: The independent searches consist of searches among which there is no communication at all. Notice that in this model we run a sequential version of the MAs /cMAs so this model doesn't introduce modifications to the sequential version of the algorithm. Yet, this model is particularly interesting when different searches use different configurations that is, different starting populations and values for the search parameters. In this setting larger regions of search space could be explored by different searches. We refer to this model as *Independent Runs - Different Strategies* (IR-DS). The advantage of this model is not in the reduction in the execution time, but in the possibility of each independent search to explore a region of

the solution space within the same time and therefore the possibility to find a better solution is increased. The different configurations are automatically generated alleviating thus the burden to the user to manually introduce the values of the parameters. Moreover, the automatic configuration of different independent searches allows a much more versatile resolution of the problem, which is very suitable in the planning in real time of jobs in a computational grid due to its dynamics.

Cooperative searches: The cooperative search model consist of searches with a certain degree of cooperation and interdependence. The different strategies that fall into this category are distinguished through several parameters: a) the topology that establishes the relation among the processes; b) the communication mechanism; c) the information exchanged by the processes; d) the time period used to exchange information; and e) the synchronism among the processes (synchronized or asynchronous communication).

Regarding the Parallel Evolutionary Algorithms (PEAs), two cooperative search strategies are usually distinguished: the coarse-grained and the fine-grained strategy.

The *coarse-grained strategy* divides the global population of the EA in sub-populations assigned to different processors and each processor applies the EA on the corresponding sub-population. This sub-population, however, doesn't evolve independently but rather according to migration policy of individuals in such a way that the exchanged individuals influence the evolution of the sub-populations. Thus, new parameters that guide the search appear: a policy of selection of the individuals to be exchanged (*emigration policy*), the number of individuals to be exchanged (*migration rate*), the time interval when the exchange takes place (*migration interval*) and the policy that integrates the exchanged individuals into the corresponding sub-populations (*immigration policy*). Besides, depending on how the migratory flows are established among the sub-populations, in the coarse-grained strategy two models are distinguished: (a) *Island model*: an exchange of individuals among any couple of sub-populations (islands) can be done; and (b) *Steeping-stone model*: a relation among the sub-populations is established so that only exchanges of individuals among neighboring sub-populations can be done (it is necessary to establish a certain topology among the sub-populations).

In the *fine-grained strategy* the population of individuals is equally divided into small sub-populations, with trend of a cardinality 1, that is, just an individual is assigned to each processor. The subsets of individuals are related through a topology in neighboring groups influencing, thus, on the convergence of the population. The cMA designed in this work follows this model in a sequential setting, where the individuals of the population are structured in a 2D toroidal grid and the individuals are only combined with the neighboring individuals (according to the type of chosen neighborhood).

3.2 Hybrid models

There are several strategies based on the hierarchical combination of the parallel models presented above, where different types of parallelism are established at different hierarchical levels. In [6] are surveyed some of the hybridization possibilities studied in the literature.

Coarse-grained + fine-grained: the coarse-grained model is combined at a first level, by dividing the population into sub-populations and establishing a migration mechanism among them, with the fine-grained model at a second level, in which each sub-population is structured in a topology (see Fig. 2, left). This strategy was originally proposed in [11].

Coarse-grained + Master-Slave: different processors (coarse-grained model at a first level) divide the population into sub-populations and each master processor delegates tasks to slave processors following the model Master-Slave at a second level (see Fig. 2, right). This models simply reduces the computational cost of the coarse-grained model.

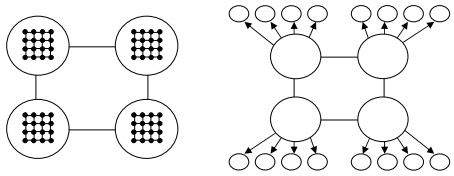

Fig. 2. Hierarchic PEA: coarse-grained model at 1st level and fine-grained model at 2nd level (left); Hierarchic PEA: coarse-grained at 1st level and MS at 2nd level (right).

Coarse-grained + coarse-grained: this model applies the coarse-grained at two hierarchical levels. The population is divided into sub-populations, and each sub-population on its own is divided into sub-populations; two different mechanisms of migration for the two hierarchical levels are applied (see Fig. 3, left).

Coarse-grained + fine-grained for cMA algorithm: in this model the grid structure of the population is distributed to different processors in such a way that two hierarchical levels are established: the first formed by the processors related among them in a grid topology (coarse-grained steeping-stone model) and the other one by the grid of individuals in each processor (see Fig. 3, right). The joint performance of all processors has to produce the evolution of only one population structured in a grid (fine-grained). Notice that the division of the population in sub-populations (coarse-grained model) at the first level establishes blocks of semi-isolated individuals and thus the behavior of a structured and parallelized population differs from the behavior of the same population in a sequential model.

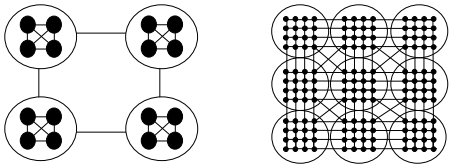

Fig. 3. Hierarchic PEA: coarse-grained model at 1st and 2nd level (left); Parallel cMA coarse-grained + fine-grained (right).

4 Design and implementation of the parallel EAs (PEAs)

We show now the design and implementation of the parallel models for MAs using an algorithmic skeleton.

4.1 Extension of the MALLBA skeletons

The implementation of the parallel models uses as a basis the MALLBA skeletons [2] adopted and extended here for the case of MAs and cMAs. The skeleton encapsulates the different models of parallelism in different classes through inheritance from the Solver base class (Solver_Seq, Solver_IR and Solver_MS, shown in Fig. 4). The design of the skeleton achieves the separation between the *provided* part, which implements the generic parallel models, from the *required* part, which implements the problem dependent features of the skeleton.

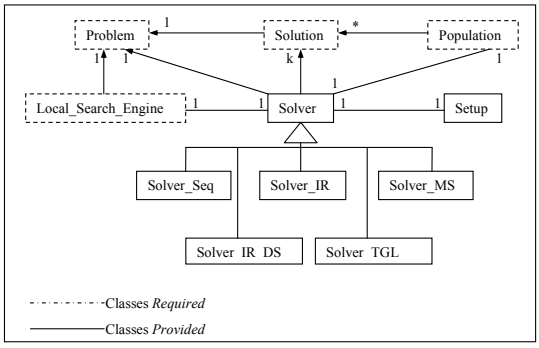

Fig. 4. The parallel skeleton for MAs and cMAs.

As can be seen from Fig. 4, apart from IR and MS implementations, two new parallel models have been implemented, namely, a) the model IR_DS for the Parallel MAs (PMA) and Parallel Cellular MA (PCMA) and b) Two Level Granulation model for PCMA. Also, the Solver_MS class has been extended to support PMA algorithm.

Communication cost in the parallel models: A key aspect in the implementation of the different parallel models is to achieve a reduced communication time among the processors. Depending on the running environment the communication cost could suppose a bottleneck that reduces drastically the expected speed up. In order to minimize the communication time we have minimized the amount of information exchanged among the processors. More precisely, the data exchanged most frequently in PEAs is the solution to be manipulated by the processors. This is not critical in the IR and IR-DS models but could be critical in case of MS and TLG models since solutions are frequently exchanged. Instead of sending the whole solution, we implemented sending/receiving routines, which send only *changes* in the new solution, given that the processors know the previous solution. This approach reduces drastically the communication time since the changes are those produced by the local perturbation, which are small or very small changes.

Generation and use of different search strategies: A search strategy is defined as a starting population and a set of values for the search parameters. In order to establish the different configurations of the searches, which are automatically generated and assigned to processors. The user is required to only provide the range of values for each parameter $(liminf, limsup)$ and a probability distribution (Constant, Normal, Uniform, Exponential or Triangular) for choosing the value of the parameter. Regarding the MS model, we have also introduced the novelty that each slave processor can use its own strategy for the search, that is, the individuals of the same population are locally perturbed using different local search methods. It is important to notice that this feature of the parallel implementation implies that the behavior of the MS model is different from the sequential algorithm.

4.2 Communication and migration in TLG model for the cMAs

As we already mentioned, in this model we have two hierarchical levels, the coarse-grained and fine-grained levels. Regarding the first we have to decide the exchange protocol for the individuals and regarding the second we have to specify the migration policy.

Communication in the coarse-grained level: Two types of communications can be established. a) *Synchronous communication*: the processors exchange the individuals in pre-established stages of the algorithm, precisely at the end of each iteration, in such a way that the neighboring processor will not have the modifications carried out until the exchange take place; and b) *Asynchronous communication*: any sub-population can dispose of the updated individuals of the neighboring sub-populations, that is, the modification of an individual is immediately available for the rest of sub-populations. The communication model used in this work is a) since it combines a synchronous model of semi-isolated sub-populations with the asynchronous model in each sub-population.

Fine-grained level: This second hierarchical level corresponds to the sub-population level, which preserves the grid topology but now a set of migration operators must be defined in order to "export" and "import" individuals in neighboring sub-populations. Due to this, now the 2D torodidal grid topology is not applicable; instead, a virtual topology is used (see Fig. 5).

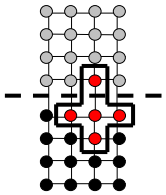

Fig. 5. Virtual topology of neighboring sub-populations.

4.3 Design and implementation of the coarse-grained steeping stone level

Given the specifications of a global population, that is, the parameters of population height, population width, number of recombinations and number of mutations, the global population is divided into several sub-populations, assigned to the different available processors in such a way that cMA can be applied to the sub-populations and by achieving the global evolution of a unique population with the specified parameters. Two roles are distinguished for the processors: a *main processor* in charge of mapping the sub-populations to the processors and to carefully distribute the number of recombinations and mutations to be performed by the processors, and the *secondary processors* that work on sub-populations in a coordinated way among them and with the main processor. The algorithm run by the main processor is shown in Fig. 6.

```
Mapping of the population to p processors
For each independent_run do
    While stopping condition is not met do
        For i=0 to p do
            send signal of continuation to processor i
            calculate #recombinacions and #mutacions for processor i
            send #recombinacions and #mutacions to processor i
        EndFor
    EndWhile
    Receive the result from the execution of each processor
    Calculate the best solution and running time statistics
EndFor
```

Fig. 6. Main processors' algorithm in the coarse-grained steeping stone level.

Regarding the secondary processors, these have to execute the cMA algorithm on their sub-population. In this case, the algorithm is essentially the sequential version of the algorithm except that new elements are added to perform the distribution of the population (see Fig. 7).

```
Receive the sub-population size and the set of
neighboring processors V.
/*Construct sub-population P*/
For each independent_run do
    Initialize the grid P(t=0) with n individuals
    /*Permutations containing the order used for
    the recombination and mutation of the individuals*/
    Initialize rec_order and mut_order
    For each individual i of P, apply local search on i
    Evaluate the individuals of P
    Exchange the shared individuals with V
    /*Reception of the stopping_signal */
    While not (stopping_signal) do
        Receive #recombinations and #mutations
        For j = 1 to #recombinations do
            SelectToRecombine S ⊆ N_P[rec_order.current()]
            i' := Recombination of S
            Local search on i'; Evaluation of i'
            Substitute P[rec_order.current()] by i'
            rec_order.next()
        EndFor
        For j = 1 to #mutations do
            i = P[mut_order.current()]
            i' := Mutation of i; Local search on i'; Evaluation of i'
            Substitute P[mut_order.current()] by i'
            mut_order.next()
        EndFor
        Update rec_order and mut_order
        Exchange the updates of the solutions shared with V
        Receive stopping_signal
    EndWhile
    Send results to the main processor
EndFor
```

Fig. 7. Secondary processors's algorithm in the coarse-grained steeping stone level.

Next, we detail the set of operators used in the coarse-grained level, such as mapping of the population or the exchange of individuals (the rest of cMA operators are omitted here).

Mapping of the population: The mapping of the population consists in dividing it into p grids, where p is the number of secondary processors. Therefore, we have to fix the value p_h (number of cells per column), p_w (number of cells per row) and h_p (the cell width). The following conditions must hold $p = p_h \times p_w,\ h_p \times w_p \times p = h \times w$, where h and w are the height and the width of the global population, resp. It is attempted to preserve the proportions of the population (relation between the height and the width) in each of the small grids, in such a way that the subdivision of the population has to be the "squarest" possible, where the difference between p_h and p_w is minimum. The

ideal case is that where p has an exact square root ($p_h = p_w = \sqrt{p}$), yielding to perfect proportions: $h_p = h/\sqrt{p}$ and $w_p = w/\sqrt{p}$, $\frac{h}{w} = \frac{h_p}{w_p}$.

The extreme case corresponds to p a prime number; in this case the population would remain in p portions in a same dimension (in our approach divided into p rows).

Next, the topology of the secondary processors is established as a 2D toroidal grid, in which each processor is assigned the corresponding neighbors. More precisely, a vector of 8 positions contains the identifiers of the 8 neighboring processors. The processors will exchange the individuals according to the neighborhood pattern established at fine-grained level.

Computing the number of recombination and mutations: Given the total number of recombination and mutations, it is necessary to distribute them uniformly for each sub-population in order to avoid possible *imbalances* (more evolved sub-populations). The distribution is thus done at random and is generated at each iteration in order to avoid the same scheme along the algorithm.

Exchange of the individuals. Two types of exchanges are distinguished: in the first, the exchange takes place just after having initialized the sub-populations and, in the second, the exchange takes place at the end of each iteration. The way the solutions are sent and received as well as the number and the set of exchanged solutions depend on the target and origin processor respectively, therefore, it depends on the neighborhood pattern established among the processors. These aspects are encapsulated in the fine-grained level, since it is at this level where the distribution of the individuals in space is known.

The exchange protocol applied here consists in sending and reception of solutions in an ordered way to avoid a deadlock situation. For each cardinal point, the processors initially export the corresponding solutions to the neighbor that it finds in that direction, next expect the reception of the solutions from the reverse direction in such a way that the exchange is guaranteed without blocking.

4.4 Design and implementation of the fine-grained level

Recall that the fine-grained level deals with the structure of the population. Each sub-population has to maintain a sequence of replicated individuals corresponding to the individuals the sub-population requires from the neighbors. The distribution of these replicas in the grid and the rules of how to interoperate with the rest of individuals are established by the cMA algorithm. Besides, the sub-population has to provide other sub-population mechanisms to extract and to import individuals according to the position (north, south,...).

Extension of the population structure: In order to manage the replicas of the neighboring individuals in the population, and that these interact with the proper individuals of the population in a transparent way, the grid

has been broadened in such a way that in the central part there is the real sub-population and in the peripheral parts are placed the replicas according to their origin. These replicas are useful only in the recombination of the neighboring individuals.

Note that the new region of replicas depends directly on the pattern of the neighborhood, avoiding to import replicas that will never be needed. We show in Fig. 8 the graphical representation of the extension of the grid with replicas for five most important topologies (panmixia, L5, L9, C9 and C13).

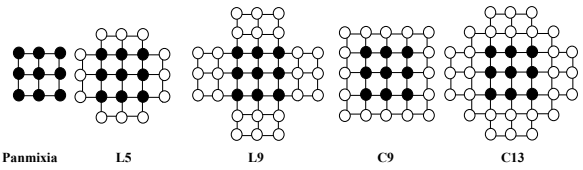

Fig. 8. Representation of a population of 3×3 individuals and the replicated region according to the neighborhood pattern.

Importing replicas: Another issue is the design of the operators for importation of replicas and immigration operators. The former is used for initializing the population and thus complete solutions are imported, while the later serves for updating the replicas. In the first case the whole solution is copied for each replica and, in the second case, are applied just the changes to the solution.

The position from which the replicas will be imported depends on the cardinal point where the individuals come from (see Fig. 9, left). The individuals are imported following a fixed arbitrary order (left-right and up-down). It should be noted however that this order should be the same in exporting the central individuals, so as to not alter the virtual neighborhoods among the individuals of the neighboring sub-populations.

Exporting replicas: The operators of exporting and emigration of individuals correspond to the opposite versions of the import operator. In the exportation operator whole copies corresponding to the central individuals are exported, while in emigration only the resulting modifications are exported as new changes of the same individuals only in case they have been modified. The export protocol coincides with that of import operator (left-right and up-down), as shown in Fig. 9 (right).

5 Experimental study

In this section we present the experimental study to evaluate the performance and improvement of the implemented parallel models for MAs and cMAs. Notice that we are especially interested to evaluate the trade-off between the

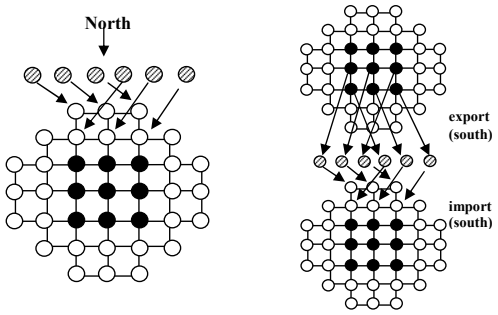

Fig. 9. Importing of individuals coming from the north neighbor in a C13 neighborhood pattern (left); Correspondence of the exporting and importing individuals between north-south neighbors (right).

quality of solutions and the computation time given the dynamic nature of grid systems. Two criteria, directly related to the performance of the parallel implementations, have been considered: (a) the speed-up (or the reduction in computation time obtained by the parallelism) and (b) the performance of those parallel versions, which behave differently from their sequential version.

For the first criterion, the speed-up is computed in the usual way, that is, $S(p) = T_s/T_p$, where T_s and T_p are the sequential and parallel execution time, resp. Regarding the second criterion, as we already mentioned, some models of parallelism considered in this work introduce changes, which make the parallel algorithms to behave differently from their sequential versions; in a certain sense, we could speak of two different algorithms, in this case. More precisely, from the parallel models considered in this work, the MS for MAs – which differs from the sequential model in the fact that different mechanisms of local search can be applied on the individuals of the same population–, and the TLG for cMA, in which the division of the population into partially isolated small grids alters the behavior that would show the same population without the coarse-grained distribution.

We note that in all computational results shown in next subsections, the number of processors refers to the total number os processors participating in the execution. Thus, in the case of IR-DS model, having p processors means that $p-1$ independent searches are executed and for the MS model there is a master processor and $p-1$ slave processors. Finally, in the TLG model, p indicates the number of subdivisions of the population.

It should also be mentioned that no speedup is provided by the independent searches model, which consists of just running the sequential version in different processors.

5.1 Speed up analysis

A group of four instances of semi-consistent ETC matrices has been generated using a grid simulator [19] and used for this experimental study; their sizes are shown in Table 1. For each instances, computational results are reported for MS implementation for the Parallel MA (PMA) and TLG implementation for the Parallel cMA (PcMA). The results are averaged over 20 executions using machines of our department PC cluster of usual configuration[1].

Table 1. Instance sizes used in the speedup analysis.

	Small	Medium	Large	Very Large
Nb. of Tasks	512	1024	2048	4096
Nb. of Resources	32	64	128	256

Performance of MS implementations

The values of the parameters used in PMA algorithm are shown in Table 2. The number of generations has been fixed to 100 and the size of the population has been settled to follow a gradual slow down according to the increase of the instance size (see Table 3).

Table 2. Parameter values used in PMA algorithm.

nb_generations	100
nb_solutions_to_recombine	3
nb_recombinations	0.2× population_size
nb_mutations	0.8× population_size
start_choice	MCT and LJFR-SJFR
select_choice	Random Selection
recombine_choice	One point recombination
recombine_selection	Binary Tournament
rec_selection_extra	0.9
mutate_choice	Rebalance-Both
mutate_extra_parameter	0.6
mutate_selection	Best Selection
local_search_choice	LMCTS
nb_local_search_iterations	10
nb_ls_not_improving_iterations	4
add_only_if_better	false

In the table, MCT and LJFR-SJFR denote two deterministic methods used to generate initial solutions; they stand for Minimum Completion Time and Longest-Job to Fastest Resource -Shortest Job to Fastest Resource, resp. LMCTS (Local MCT Swap) is the local search methods used.

[1] All computers have the same configuration running Linux as operating system (Redhat distribution).

Table 3. Size of the population established according to the instance size.

	Small	Medium	Large	Very Large
Population size	50	60	70	75

We present in Table 4 the execution and communication time among the processors for large and very large size instances. Notice that we have omitted the makespan value since, in order to measure the communication time the same local search mechanism is used in the slave processors.

Table 4. Execution and communication time for large size instances (left) and very large size instances (right) for MS implementation.

Nprocs	t(s)	tcomm(s)
1	268.98	0
3	220.46	7.06
4	161.58	6.89
5	122.15	6.54
7	91.29	6.43
8	73.09	6.32

Nprocs	t(s)	tcomm(s)
1	602.64	0
3	499.06	14.1
4	353.51	13.74
5	268.21	13.19
7	191.97	13.25
8	176.94	13.13

As can be observed, for all of instances the reduction in the execution time is very considerable with the increase of the number of processors. The reduction in time follows an evolution similar to $T_s/\log(p-1)$ function, for $p > 2$ processors.

Performance of Two Level Granulation implementation

The configuration of parameters used in TLG is shown in Table 5. The only parameter that varies is the population size whose value is fixed according to the instance size (see Table 6). It should be noticed that the population size value is fixed independently of the number of processors, therefore the population size has to be large enough to ensure admissible sub-divisions in TLG. The number of generations has been fixed to 200.

The makespan values[2], the execution and communication time in TLG implementation according to the instance size and number of processors used are given in Tables 7. Notice that the number of processors coincides with the number of sub-divisions of the population.

A qualitative improvement is observed in the value of makespan as the number of processors increases. This implies that for a same configuration and workload, the division of the cellular population in smaller grids benefits the search. Besides, a clear improvement is observed in the execution time with the increase in the number of processors.

However, TLG implementation doesn't behave as regularly as the MS, which could be explained by the fact that according to the number of processors a different structure of the sub-population is established at the coarse-

[2] In arbitrary time units.

Table 5. Parameter values used in the evaluation of the PCMA algorithm.

nb_generations	200
nb_solutions_to_recombine	3
nb_recombinations	1×population_height×population_width
nb_mutations	0.5×population_height×population_width
start_choice	MCT and LJFR-SJFR
neighborhood_pattern	C9
recombination_order	FLS
mutation_order	NRS
recombine_choice	One point recombination
recombine_selection	N Tournament
rec_selection_extra	3
mutate_choice	Rebalance-Both
mutate_extra_parameter	0.6
local_search_choice	LMCTS
nb_local_search_iterations	5
nb_ls_not_improving_iterations	$+\infty$
lsearch_extra_parameter	0
add_only_if_better	true

Table 6. Population size according to instance size.

	Small	Medium	Large	Very Large
Population size (height×width)	6×6	6×6	7×7	7×7

Table 7. Makespan values, execution and communication time for small size instances (left) and medium size instances (right) in TLG implementation.

Nprocs	Makespan	t(s)	tcomm(s)
1	1834405.18	81.45	0
3	1803143.55	53.9	1.01
4	1806527.72	32.48	1.56
6	1796988.74	33	1.51
8	1767354.44	24.74	1.7
9	1791683.11	32.61	1.68

Nprocs	Makespan	t(s)	tcomm(s)
1	1179876.25	186.26	0
3	1155363.36	102.08	1.07
4	1152594.67	71.02	1.59
6	1148556.35	61.26	1.54
8	1134427.8	39.88	1.76
9	1146915.11	47.81	1.74

Table 8. Makespan values, execution and communication time for large size instances (left) and very large size instances (right) in TLG implementation.

Nprocs	Makespan	t(s)	tcomm(s)
1	768778.44	588.93	0
3	753870.58	239.17	1.46
4	748673.41	190.07	1.92
6	745782.64	147.1	2.22
8	743934.16	104.65	2.16
9	745861.82	92.5	2.06

Nprocs	Makespan	t(s)	tcomm(s)
1	490088.47	1242.77	0
3	481851.38	515.65	2.43
4	481153.9	393.14	2.66
6	478877.1	279.29	3.93
8	477989.05	191.69	3.79
9	476626.85	175.41	3.17

grained level. As a matter of fact, the makespan improvement as well as the communication time depend more on the concrete value of the number of processors (e.g. when it is a primer number) rather than the amount of processors used. The most evident case is that when the number of processors p is a prime number in which case the grid is divided into p rows. This phenomenon minimizes the communication time significantly due to the vertical cuts in the grid (lateral communication), but on the other hand it implies a deformation of the proportions given by the user to construct the sub-populations, with-

out respecting the relation between the height and the width of the global population.

Again, the execution time follows the function $T_s/\log(p-1)$ although not as adjusted as in the case of MS implementation.

5.2 Qualitative improvement of the makespan results

The experimental study presented in previous section aimed at identifying the relation of execution and communication time with the increasing number of processors, respectively. This is important to further analyze the qualitative improvement in the makespan of the schedule, which is the main objective of the considered parallel implementations. On the other hand, we also noticed that, independently of the execution time, there could be a qualitative improvement due to the introduction of new elements in the parallel versions of the algorithms, which yields to "*new*" algorithms. In the case of MS this behavior is due to the fact that each processor can apply different local search procedures to the individuals of the population. In the TLG model the new behavior comes from the fact that by dividing a population into sub-populations, these become partially isolated since they don't share the individuals in asynchronous way, as in the sequential cMA algorithm. Finally, the IR-DS model behaves essentially as the sequential algorithm, although qualitative improvements are possible by using the best configuration in each independent search.

We used different size instances (small, medium, large and very large) to observe the improvement obtained by each parallel model according to the particular algorithm that it has been applied to, namely, Parallel MA and Parallel cMA. Again, the results are averaged over 20 executions carried out in same PC cluster (see above). The maximum execution time has been fixed to 90 seconds, which is considered as a reasonable time interval for scheduling independent jobs in real grid environments.

Comparison of IR-DS, MS and sequential implementations of PMA

We present next the configuration used in PMA as well as the makespan values obtained with IR-DS, MS and sequential implementations, respectively, of the PMA. Based on the computational results we can compare the improvements by the IR-DS and MS implementations of PMA with respect to the sequential implementation according to instance size and number of processors.

Notice that, in order to generate the different strategies for the IR model, lower and upper bounds for the parameters should be fixed. These values are given in Fig. 10; the range of the population size according to instance size is shown in Table 9.

In Fig. 10, MCT (Minimum Completion Time) and LJFR-SJFR (Longest-Job to Fastest Resource -Shortest Job to Fastest Resource) are the deterministic methods used to generate initial solutions. LMCTS (Local MCT Swap) and LMCTM (Local MCT Move) are the local search methods used, resp.

Parameter	Inferior limit	Superior limit	Distribution
nb_generations	(max 90 s)		
nb_solutions_to_recombine	2	4	uniform
nb_recombinations	0,1× population_size	0,4× population_size	uniform
nb_mutations	0,7× population_size	0,9× population_size	uniform
start_choice	MCT and LJFR-SJFR		
select_choice	Random		
select_extra_parameter	∅		
recombine_choice	One point recombination		
recombine_selection	Binary Tournament		
rec_selection_extra	0,7	1	uniform
mutate_choice	Rebalance-Both		
mutate_extra_parameter	0,5	1	normal
mutate_selection	Best Selection		
mut_selection_extra	∅		
local_search_choice	LMCTS and LMCTM		
nb_local_search_iterations	5	20	uniform
nb_ls_not_improving_iterations	0,5×nb_local_search_iterations	1×nb_local_search_iterations	uniform
lsearch_extra_parameter	∅		
Boolean parameter	**Likelihood**		
add_only_if_better	0,1		

Fig. 10. Configuration of the IR-DS implementation for PMA.

Table 9. Population size in IR-DS implementation for PMA.

	Small	Medium	Large	Very Large
Population size (sequential mode)	50	60	70	75
Population size (IR-DS) [inf, sup], distr.	[35,65] uniform	[45,75] u. [45,75] u.	[55,85] u.	[60,90] u.

The computational results for makespan parameter are given in Tables 10 and 11 (the best value of both implementations is shown in bold). As can be observed from the tables, the makespan value is improved, compared to the sequential implementation, as the number of processors increases. We exemplify the makespan reductions by both implementations in Fig. 11, where we can see that MS implementation outperforms the IR-DS implementation.

Table 10. Makespan values obtained by the PMA versions IR-DS and MS for small size instances (left) and medium size instances (right).

Nprocs	PMA(IR)	PMA(MS)
(Seq)	**1632339.51**	
3	1636429.04	**1634867.64**
4	1622557.25	**1620267.95**
5-6	1609863.35	**1609296.35**
7-8	1610404.06	**1601005.42**
9	1611584	**1594404.15**

Nprocs	PMA(IR)	PMA(MS)
(Seq)	**1008864.68**	
3	1021601.5	**989279.28**
4	992045.54	**975075.05**
5-6	986709.91	**973972.77**
7-8	976843.64	**960896.63**
9	979943.91	**952360.84**

Comparison of IR-DS, TLG and sequential implementations of PCMA

The configuration used in PCMA as well as the makespan values obtained with IR-DS, TLG and sequential implementations, respectively, of the CPMA are presented next. Based on the computational results we can compare the improvements by the IR-DS and MS implementations of CPMA with respect

Table 11. Makespan values obtained by the PMA versions IR-DS and MS for large size instances (left) and very large size instances (right).

Nprocs	PMA(IR)	PMA(MS)
(Seq)	**654102.25**	
3	663821.81	**658875.39**
4	**645805.22**	647980.86
5-6	648881.96	**644351.72**
7-8	641974.1	**639227.82**
9	640485.7	**633337.48**

Nprocs	PMA(IR)	PMA(MS)
(Seq)	**441155.97**	
3	**444951.86**	449175.51
4	**440938.81**	441038.38
5	**439542.5**	440562.97
7	438428.26	**436174.14**
9	437178.6	**432571.49**

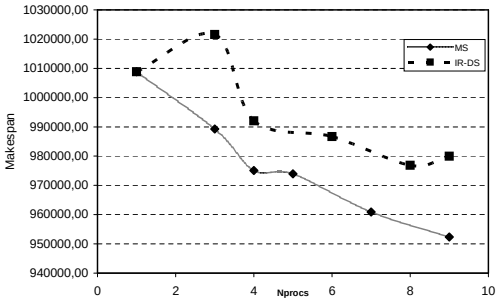

Fig. 11. Makespan reduction of IR-DS and MS implementations of the PMA for a medium size instance.

to the sequential implementation according to instance size and number of processors participated in the search.

Again, in order to generate the different strategies for the IR model, lower and upper bounds for the parameters are fixed. These values are given in Fig. 12 and the range of the population size according to instance size is shown in Table 12.

Parameter	**Inferior limit**	**Superior limit**	**Distribution**
nb_generations	(max 90 s)		
nb_solutions_to_recombine	2	4	uniform
nb_recombinations	0,8× population_height× population_width	1× population_height× population_width	uniform
nb_mutations	0,4× population_height× population_width	0,6× population_height× population_width	uniform
start_choice	MCT and LJFR-SJFR		
neighborhood_pattern	L5, L9, C9		uniform
recombination_order	FLS		
mutation_order	NRS		
recombine_choice	One point recombination		
recombine_selection	N Tournament		
rec_selection_extra	3		
mutate_choice	Rebalance-Both		
mutate_extra_parameter	0,5	1	normal
local_search_choice	LMCTS and LMCTM		uniform
nb_local_search_iterations	3	15	uniform
nb_ls_not_improving_iterations	0,3×nb_local_search_iterations	1×nb_local_search_iterations	uniform
lsearch_extra_parameter	∅		
Boolean parameter	Likelihood		
add_only_if_better	0,8		

Fig. 12. Configuration of the IR-DS implementation for PCMA.

Table 12. Population size in IR-DS implementation for PCMA.

	Small	Medium	Large	Very Large
Sizes sequential mode (height×width)	5×5	6×6	6×6	7×7
Height (IR-DS) [inf, sup], distr.	[4,6] uniform	[5,7] u.	[5,7] u.	[6,8] u.
Width (IR-DS) [inf, sup], distr.	[4,6] uniform	[5,7] u.	[5,7] u.	[6,8] u.

As can be seen from the configuration of the IR-DS model, the proportions of the structured population are not fixed, therefore the independent searches not only could vary in the number of individuals but also in the structure of the population. The patterns of the neighborhood used are those that guarantee a more acceptable behavior, discarding the panmixia and C13 since both favor a too intense search policy. The most influential operators in the search have been fixed in order to avoid the negative effect of the bad configurations of searches. The local search procedures used, are again LMCTS and LMCTM, as in the case of PMA algorithm.

An additional aspect to take into account in the context of the subpopulations is that of the their size. The decision has been to maintain the number of individuals and the pattern of the population as in the sequential model. For the TLG model better results are obtained with smaller populations; indeed, smaller size populations are computationally less expensive and the cellular structure seem to protect them from a premature convergence. We show in Table 13, the values of the height and the width according to the instance size and the number of processors used.

Table 13. Sizes of the population used in TLG.

Nprocs	Small	Medium	Large	Very Large
3	6×4	6×6	6×6	7×7
4	5×5	6×6	6×6	7×7
6	6×4	6×6	6×6	7×7
8	8×4	8×4	8×4	7×7
9	6×6	6×6	6×6	7×7

The computational results for the makespan parameter obtained from IR-DS and TLG implementations of PCMA are shown in Table 14 and Table 15 (the best value of both implementations is shown in bold). The IR-DS shows the same behavior: initially with the increasing number of processors, the makespan is reduced and later it's stagnated. In in a certain sense this behavior shows the usefulness but also the limitations of the independent searches with different strategies. Regarding TLG implementation, the results show a considerable improvement outperforming the IR-DS model. However, as in the case of MS implementation, the communication cost of the model shows to be considerable with the increase of the number of processors, which goes in detriment of the proper search time. On the other hand, it's not worth increasing the population size since it doesn't improve the quality of the results with less processors without increasing the total execution time.

Table 14. Makespan values obtained by the PMA versions IR-DS and TLG for small size instances (left) and medium size instances (right).

Nprocs	PCMA(IR-DS)	PCMA(TLG)
(Seq)	1595730.87	
3	1618178.02	**1591373.7**
4	1614640.4	**1581589.48**
5-6	1579210.6	**1572990.76**
7-8	1580631.56	**1568097.94**
9	1574928.9	**1578471.46**

Nprocs	PCMA(IR-DS)	PCMA(TLG)
(Seq)	1087018.94	
3	1071528.14	**1044949.77**
4	1032718.9	**1012936.09**
5-6	1027370.45	**1000646.61**
7-8	1017213.54	**983838.58**
9	1018227.21	**989905.15**

Table 15. Makespan values obtained by the PMA versions IR-DS and TLG for large size instances (left) and very large size instances (right).

Nprocs	PCMA(IR)	PCMA(TLG)
(Seq)	767549.31	
3	750571.49	**741375.89**
4	746843.73	**727341.88**
5-6	734531	**717319.02**
7-8	736794.5	**701782.12**
9	733531.82	**695951.31**

Nprocs	PCMA(IR)	PCMA(TLG)
(Seq)	500354.57	
3	499817.34	**399661.47**
4	496147.31	**356350.92**
5-6	493774.6	**356528.66**
7-8	492910.19	**478943.6**
9	492125.28	**475351.07**

6 Conclusions

In this chapter we have presented the implementation and the computational evaluation of three parallel models of Memetic Algorithms (MAs) for the Independent Job Scheduling problem in Computational Grids. The Independent Search with Different Strategies (IR-DS), the Master-Slave (MS) and a hybridization between the coarse-grained and fine-grained models (Two Level Granulation –TLG) are considered. The interest of these models is twofold. First, these models can contribute to finding better solutions, either by searching larger areas of the solution space or by speeding up the running time of the algorithms. Second, for the Cellular MAs (cMA), some parallel models, such as MS and TLG behave differently from their sequential versions. These structural differences could *per se* yield better performance of cMAs.

The extensive experimental results using different size instances (from small to very large) generated with a grid simulator showed that all these models provide important qualitative improvements for makespan parameter as compared to the sequential versions. From a practical perspective, the parallel implementations of MAs and cMAs are very good alternatives for scheduling in computational grids since they are able to obtain a very accentuated reduction in the makespan value of the schedule.

In our future work we plan to use the resulting Parallel MAs and parallel cMA-based schedulers as part of grid applications and evaluate their performance in a real grid environment.

References

1. A. Abraham, R. Buyya, and B. Nath. Nature's heuristics for scheduling jobs on computational grids. In *Proceedings of the 8th IEEE International Conference*

on Advanced Computing and Communications, Tata McGraw-Hill, 45-52, 2000.
2. E. Alba, F. Almeida, M. Blesa, C. Cotta, M. Daz, I. Dorta, J. Gabarr, C. Len, G. Luque, J. Petit, C. Rodrguez, A. Rojas, and F. Xhafa. Efficient parallel LAN/WAN algorithms for optimization. the mallba project. *Parallel Computing*, 32(5-6):415–440, 2006.
3. M.J. Blesa, P. Moscato, and F. Xhafa. A memetic algorithm for the minimum weighted k-cardinality tree subgraph problem. In J. Pinho de Sousa, editor, *Metaheuristic International Conference*, Vol. 1, 85–91, 2001.
4. T.D. Braun, H.J. Siegel, N. Beck, L.L. Boloni, M. Maheswaran, A.I. Reuther, J.P. Robertson, M.D. Theys, and B. Yao. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *J. of Parallel and Distributed Comp.*, 61(6):810–837, 2001.
5. 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 Comp.*, 283–289, IEEE Press, 2000.
6. E. Cant-Paz. *A Survey of Parallel Genetic Algorithms*, Calculateurs Paralleles, Reseaux et Systems Repartis, 10(2), 141-171, 1998.
7. J. Carretero and F. Xhafa. Using genetic algorithms for scheduling jobs in large scale grid applications. In *Journal of Technological and Economic Development*, 12(1):11-17, 2006.
8. T. Crainic and M. Toulouse. *Parallel Metaheuristics, Fleet Management and Logistics*, T.G. Crainic, G. Laporte editors, Kluwer, pp. 205-251, 1998.
9. I. Foster and C. Kesselman. *The Grid - Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, 1998.
10. I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the grid: Enabling scalable virtual organization. *International Journal of Supercomputer Applications*, 15(3), 200–222, 2001.
11. F. Gruau. Neural networks synthesis using cellular encoding and the genetic algorithm, Ph.D. Thesis, Universit Claude Bernard-Lyon I, France, 1994.
12. J. Holland. *Adaptation in Natural and Artificial Systems*, University of Michigan Press, Ann Arbor, 1975.
13. H.H. Hoos and Th. Sttzle. *Stochastic Local Search: Foundations and Applications*. Elsevier/Morgan Kaufmann, 2005.
14. V. Di Martino and M. Mililotti. Sub optimal scheduling in a grid using genetic algorithms. *Parallel Computing*, 30:553–565, 2004.
15. Z. Michalewicz and D.B. Fogel. *How to solve it: modern heuristics*. Springer, 2000.
16. P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Techrep N. 826, CalTech, USA, 1989.
17. A.Y. Zomaya and Y.H. Teh. Observations on using genetic algorithms for dynamic load-balancing. *IEEE Trans. on Parallel and Distributed Sys.*, 12(9):899–911, 2001.
18. F. Xhafa. A Hybrid Evolutionary Heuristic for Job Scheduling on Computational Grids. In Grosan, Abraham, and Ishibuchi (Eds.), *Hybrid Evolutionary Algorithms*, Studies in Computational Intelligence, Ch. 13, 2007. Springer.
19. F. Xhafa, J. Carretero, L. Barolli, A. Durresi. Requirements for a an event-based simulation package for grid systems. *Journal of Interconnection Networs*, Vol. **8(2)**, pp. 163-178, 2007, World Sci. Pub.

Index