Enhancing the genetic-based scheduling in computational grids by a structured hierarchical population ☆

Joanna Kołodziej[a,*], Fatos Xhafa[b,1]

[a]*University of Bielsko-Biała; Department of Mathematics and Computer Science; ul. Willowa 2, 43-309 Bielsko-Biała, Poland*
[b]*Department of Computer Science and Information Systems; Birkbeck, University of London; Malet Street, Bloomsbury, London WC1E 7HX, UK.*

Abstract

Independent Job Scheduling is one of the most useful versions of scheduling in Grid systems. It aims at computing efficient and optimal mapping of jobs and/or applications submitted by independent users to the Grid resources. Besides traditional restrictions, mappings of jobs to resources should be computed under high degree of heterogeneity of resources, the large scale and the dynamics of the system. In view of the challenging nature of the problem, heuristic and meta-heuristic approaches are the most feasible candidate methods due to their ability to deliver high quality solutions in reasonable computing time. One class of meta-heuristics, less explored for the problem, is that of Hierarchic Genetic Strategy (HGS), a variant of Genetic Algorithms (GAs) that distinguishes for its capability of concurrent search of the solution space. In this work we present an implementation of HGS for Independent Job Scheduling in dynamic Grid environments. We consider the bi-objective version of the problem in which makespan and flowtime are simultaneously optimized. Building on our previous work, we improve the HGS scheduling strategy for the problem, by enhancing its main branching operation. The resulting HGS-based scheduler is evaluated under the heterogeneity, the large scale and dynamics conditions using a Grid simulator. The experimental study showed that the HGS implementation outperforms existing GA-based schedulers proposed in the literature.

Keywords: Scheduling, Computational Grid, Genetic Algorithms, Hierarchic Genetic Strategies, ETC Matrix Model, Grid Simulation.

☆ A preliminary version of this paper appeared at *23rd European Conference on Modelling and Simulation (ECMS 2009), Madrid, Spain*

* Corresponding author

Email addresses: jkolodziej@ath.bielsko.pl (Joanna Kołodziej), fatos@dcs.bbk.ac.uk; fatos@lsi.upc.edu (Fatos Xhafa)

[1] On leave from Technical University of Catalonia, Spain.

1. Introduction

Computational Grids (CGs) has emerged as an alternative to super-computing for solving large-scale problems within a reasonable time, by making use of large amounts of computational resources virtually joined in CGs. At the center of CGs is the mapping of tasks and applications submitted by independent user to Grid resources. Indeed, matching the computational needs of applications within resource availability in the system is crucial to achieve the efficiency and scalability under the conditions of heterogeneity, large scale and dynamics of the CGs.

Despite of many known approaches from traditional scheduling in super-computing, cluster computing and LANs, the design of efficient schedulers in Grid environments remains still a challenging task. In fact, scheduling problem in Grid systems can be seen as a family of problems. Depending on the restrictions imposed by the application needs, the problem model depends on the number of objectives to optimize (single vs. multi-objective), the environment (static vs. dynamic), processing mode (immediate vs. batch), tasks dependencies (independent vs. dependent), among others. In this paper we consider the bi-objective scheduling problem in the Computational Grid, referred to as the *Independent Job Scheduling*, in which two main metrics of the scheduling effectiveness, namely makespan and flowtime, are simultaneously optimized. We assume that tasks are processed in a batch mode and there are no dependencies among tasks so that they can be independently computed on Grid resources.

Independent Batch Job Scheduling is one of the most useful versions of scheduling in Grid systems. There are many realistic application scenarios in which arises the need of independent job scheduling. First, this version is suitable to Grid systems given the nature of its users, who submit jobs or monolithic application in independent manner to the system. Second, Grid systems are most useful for massive parallel processing, in which large amounts of data are processed independently. In such case, batch mode is appropriate given that Grid-enabled application manage voluminous data, could run periodically and also because large amount of data have to be transferred, replicated and accessed by the applications. Thus, tasks and applications are grouped in *batches* and scheduled as a group. Real life examples of batch scheduling include processing of large log data files of online systems (e.g. banking systems, virtual campuses, health systems, etc.), processing of large data sets from scientific experimental simulations (e.g. High Energy Physics, Parameter sweep applications, etc.), data mining in bio-informatics applications, etc. Even under the independent nature of jobs and the batch processing, the problem is computationally hard to solve [8]. Moreover, it becomes more challenging due to their high degree of heterogeneity of resources, the large scale and the dynamics of Grid systems.

In view of the challenging nature of the problem, heuristic and meta-heuristic approaches are the most feasible candidate methods due to their ability to deliver high quality solutions in reasonable computing time. One class of meta-heuristics, less explored for the problem, is that of Hierarchic Genetic Strategy (HGS), a variant of Genetic Algorithms (GAs) that distinguishes for its capability of concurrent search of the solution space. The objective of this work is to define an effective genetic-based batch scheduler, which can be easily implemented in a dynamic Grid environment. We used the framework of the Hierarchical Genetic Strategy (HGS) [10] to design the *Hierarchic Genetic Scheduler* (hereafter HGS-Sched), which is based on the family of dependent genetic processes enabling a concurrent search in the optimization domain. Building on

our previous work [12], we improve the HGS scheduling strategy for the problem, by enhancing its main branching operation. Further, the resulting HGS-based scheduler is evaluated under the heterogeneity, the large scale and dynamics conditions using a Grid simulator [21].

The remainder of this paper is structured as follows. We define the Independent Job Scheduling problem in Section 3. The specification of HGS-Sched procedures and a short description of genetic operators used in HGS are given in Section 4. Section 5 presents the experimental analysis of the performance of HGS-Sched. The paper ends in Section 6 with some final remarks.

2. Related work

Several stochastic and heuristic optimization methods have been proposed for Job Scheduling in Computational Grids. Monte Carlo methods, Simulated Annealing (SA), Tabu Search (TS), Genetic Algorithms (GA), among others, attempt to avoid the premature convergence to the local minima. An interesting implementation of SA for Grid scheduling was proposed by Yarhan and Dongarra in [28]. In their work the Annealing Scheduler is compared to the ad-hoc scheduler by the experimental evaluation in a real life scenario. The authors performed their experimental study in the network of three US Grid clusters, namely UTK TORC, UTK MSC and UIUC OPUS clusters. Abraham et al. [1] present three heuristic schedulers, namely SA, TS and GA, which can be additionally hybridized with each other in order to improve the scheduling efficiency.

GAs are well known for their robustness and have been applied successfully to solve scheduling problems in a variety of fields. The GAs approaches for Grid scheduling are addressed in several works. Zomaya and Teh [29] used GAs for dynamic load balancing. Braun et al. [4] compare the efficiency of a simple GA-based scheduler and methods from the set of ten static meta-task mapping heuristics from the literature, including Min-Min, Min-Max, Minimum Completion Time (MTC) algorithms [3]. The experimental study were performed for the static benchmark for Independent Job Scheduling in distributed heterogenous computing environment. The instances in this benchmark were defined using the ETC matrix model [2]. The same type of scheduling problem is considered by Xhafa et al. [22], where the authors examine several variations of GAs operators in order to identify a configuration of operators and parameters that works best for the problem. Then, the efficiency of GA-based scheduler with the appropriate combination of operators is compared with the effectiveness of the GAs approach presented in [4]. The method presented in [22] were extended by plugging the GA scheduler into Grid simulator [21] to perform the experiments in a dynamic environment. The results of the evaluation of the GA scheduler were reported in [24, 7, 27].

Recently some parallel GA frameworks are used for designing the effective Grid schedulers. Lim et al. [14] propose Grid-Enabled Hierarchical Parallel Genetic Algorithm (GE-HPGA) based on HPGA framework. Another method of improving the scheduling quality is the hybridization of the heuristics with Local Search methods for the problem [19]. A GA-TS hybrid version was proposed by Xhafa et al. [26]. Other approaches for the problem include the use of Fuzzy Particle Swarm Optimization [15] and economic-based approaches [6].

We propose in this paper a HGS-based scheduler using the HGS framework [10]. HGS has been successfully applied in solving continuous global optimization problems with multi-modal

and weakly convex objective functions [20]. It was also used as an efficient method for Permutation Flowshop Scheduling [13] as well as for solving some practical engineering problems [11].

3. Statement of the problem

In the version of Independent Batch Scheduling, it is assumed that jobs submitted to the Grid are independent and not preemptive. Additionally, we consider that tasks are processed in batch mode [23]. The problem is formalized using the *Expected Time to Compute (ETC)* matrix model [2] (see Subsec. 3.1). Based on this model, we define several optimization criteria as well as the objective function used in this work for the scheduling problem (see Subsec. 3.2).

3.1. The Expected Time to Compute model

In the *ETC* model an instance of the problem is defined by the following data, which has to be provided in input:

- the estimation of workload of each task (usually in millions of instructions);
- the computing capacity of each machine (usually in millions of instructions per second, MIPS);
- the estimation of the prior load of each available machine;
- the *ETC* matrix, whose entries $ETC[t][i]$ define the estimation of the time needed for the completion task t in machine i.

The workload of all tasks submitted to the Grid system is defined by the workload vector $WL = [wl_1, \ldots, wl_n]$, where wl_t denotes the computational load of the task t and n is the total number of tasks. The computing capacity of the Grid resources can be characterized by the vector $CC = [cc_1, \ldots, cc_m]$, in which cc_i denotes the computing capacity of machine i and m is the total number of machines. The workload of each submitted task can be estimated based on the specifications provided by the users, on historical data, or it can be obtained from predictions [9]. Having the vectors WL and CC, the entries of the ETC matrix can be computed as the ratio:

$$ETC[t][i] = \frac{wl_t}{cc_i}. \tag{1}$$

Next, we use the ETC matrix to express optimization criteria as well as the objective function used in this work.

3.2. Optimization criteria and objective function

The problem of scheduling jobs in Computational Grid is multi-objective in its general setting as the quality of the solutions can be measured under several criteria. Similarly as in [22], [24] and [7] we consider the scheduling in Grids as the bi-objective global optimization in which *makespan* and *flowtime* are to be minimized. The makespan is defined as the finishing time of the latest task and can be calculated by the following formulae:

$$Makespan = \min_{S \in Schedules} \max_{j \in Tasks} F_j, \tag{2}$$

where F_j denotes the time when task j is finalized, $Tasks$ denotes the set of all tasks submitted to the Grid system and $Schedules$ is the set of all possible schedules.

The second criterion is the flowtime, expressed as the sum of finalization times of all the tasks. It can be defined in the following way:

$$Flowtime = \min_{S \in Schedules} \sum_{j \in Tasks} F_j \tag{3}$$

Both makespan and flowtime are expressed in arbitrary time units. In fact, their values are in incomparable ranges (flowtime has a higher magnitude order over makespan and its values increase as more jobs and machines are considered), therefore $mean\_flowtime = flowtime/nb\_machines$ is used to evaluate the flowtime criterion.

In the multi-objective optimization two fundamental models are useful: the hierarchical and the simultaneous optimization. In the *hierarchical* case, the objectives are sorted according to their importance in the model. The optimization process starts by optimizing the most important objective. Then, when further improvements are not possible, the second objective is optimized, without worsening the value of the first objective. The method proceeds until all objectives have been considered. In the *simultaneous method* all objectives are optimized simultaneously.

We used the simultaneous method in this work[2] expressed as follows:

$$fitness = \lambda \cdot makespan + (1 - \lambda) \cdot mean\_flowtime. \tag{4}$$

4. Hierarchic Genetic Strategy based scheduler

The main idea of HGS-Sched is to enable a concurrent search of the optimal schedules in Grid environment by the simultaneous execution of many dependent evolutionary processes. The dependency relation among processes has a tree structure with the restricted number of levels. Every single process is interpreted as the branch in this structure and can be defined as a sequence of evolving populations. Many known GA-based schedulersb (e.g. [**?** 22]) can be implemented as main mechanisms of the evolution in HGS-Sched branches. The process in the core of the structure governs all search procedures and it is active until the stopping criterion for the whole strategy is satisfied. The HGS-Sched mechanism is different from the mechanism of the hierarchical and branching schedulers applied for solving the grid scheduling problems as well as the classical job-shop problems (see e.g. [5]).

A simple graphical representation of the HGS-Sched structure is presented in Fig. 1.

Each branch in this structure is characterized by a *degree* parameter $j \in \mathbb{N}$. The degree of the given branch corresponds to the accuracy of the search of the active process in this branch. The unique branch of the lowest degree 0 is called *root*. The processes of lower degree are not expected to be as effective as the global and local optimizers. The main idea of the execution of such processes is just the detection of the promising regions in the optimization domain, where more accurate processes of the higher degrees are further activated.

[2]We used $\lambda = 0.75$ in the main parameter settings based on the experimental tuning results presented in [24].

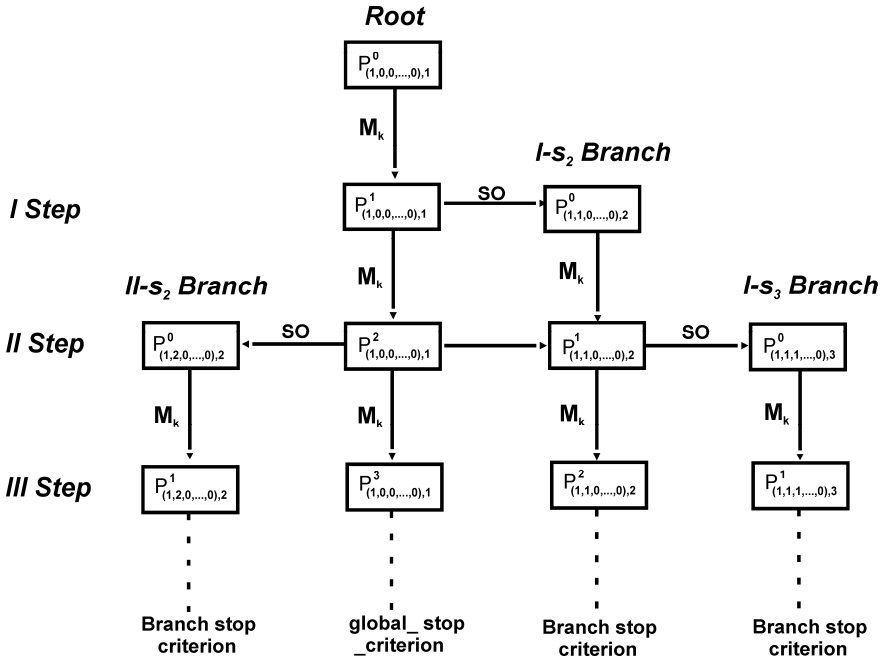

Figure 1: An example structure of HGS-Sched after execution of 3 meta-epochs

We denote by $P^e_{i,j}; (j \in \{1, \ldots, M\}, M \in \mathbb{N})$ the population evolving in the branch of degree j, where:

- $e \in \mathbb{N}$ defines the global meta-epoch counter;
- M is the maximal degree of a branch;
- i is the unambiguous branch identifier, $i = (i_1, \ldots, i_M)$, $i_h = 0$ for $h > j$, which describes the "history of creation" of the given branch [20]

A k-periodic meta-epoch, denoted by $M_k, (k \in \mathbb{N})$, can be defined as a finite k-generational evolutionary process terminated by the selection of the best adapted individual. The outcome of the meta-epoch started from the population $P^e_{i,j}$ is expressed by the following formulae:

$$M_k(P^e_{i,j}) = (P^{e+l}_{i.j}, \widehat{x}); \tag{5}$$

where $\widehat{x}$ is the best adapted individual in the meta-epoch.

The algorithm starts from an initial population generated in the root. A new branch of the higher degree can be created by using the *Sprouting Operation (SO)* after running a meta-epoch in the parental branch. In Sections 4.1 and 4.2 we define the genetic mechanism implemented in HGS-Sched branches. The reduction of some ineffective branches operating in similar regions in the optimization domain can be performed by using the *Branch Reduction Operation* (see Section 4.5).

Algorithm 1 Genetic engine template

1: Generate the initial population P^0 of size μ; $t = 0$
2: Evaluate P^0;
3: **while** not termination-condition **do**
4: Select the parental pool T^t of size λ; $T^t := Select(P^t)$;
5: Perform crossover procedure on pars of individuals in T^t with probability p_c; $P^t_c := Cross(T^t)$;
6: Perform mutation procedure on individuals in P^t_c with probability p_m; $P^t_m := Mutate(P^t_c)$;
7: Evaluate P^t_m ;
8: Create a new population P^{t+1} of size μ from individuals in P^t and P^t_m ; $P^{t+1} := Replace(P^t; P^t_m)$
9: $t := t + 1$;
10: **end while**
11: **return** Best found individual as solution;

4.1. GA engine in HGS scheduling branches

In all HGS-Sched branches, we used as evolutionary processes the GA template of Alg. 1 (a similar template can be found in [22].)

The general framework of the Alg. 1 is based on the idea of the classical $(\mu + \lambda)$ evolutionary strategy (see e.g. [18]), adapted to the scheduling problem through the implementation of specific schedule encoding methods and genetic operators.

4.2. Representations of individuals and genetic operators

In this section we specify the basic methods of chromosome representation and various genetic operators we applied in GA-engine of our hierarchical scheduler. We used in our work two methods of encoding of the schedules in Grids, namely *direct representation* and *permutation-based encoding*. In the direct representation each schedule is defined as the schedule vector $x = [x_1, \ldots, x_{nb\_task}]^T$, coordinates of which are the numbers of machines to which the particular tasks are assigned, i.e. $x_i \in [1, nb\_machines]; i = 1, \ldots nb\_tasks$. In permutation-based representation we define for each machine a sequence of tasks assigned to that machine. The tasks in the sequence are increasingly sorted with respect to their completion times. Then all task sequences are concatenated into one global vector u, which is in fact the permutation of tasks to machines. In this representation some additional information about the numbers of tasks assigned to each machine is required. We defined then the vector v of the size $nb\_machines$, in which the numbers of tasks assigned to the following machines are specified as its coordinates. A schedule in this representation is then defined as the pair of vectors:

$$x = (u; v), u = [u_i, \ldots, u_{nb\_task}]^T, v = [v_1, \ldots, v_{nb\_machines}]^T \tag{6}$$

where $u_i \in [1, \ldots, nb\_task]$ and $v_j \in [1, \ldots, nb\_task]$.

The following vector $[1, 2, 1, 4, 3, 1, 2, 4, 3, 3]^T$ is an example of the schedule for 4 machines and 10 tasks encoded by the direct representation method. The same schedule in the permutation-based representation is as follows: $([1, 3, 6, 2, 7, 5, 9, 10, 4, 8]^T; [3, 2, 3, 2]^T$.

We used in this work the direct representation for the encoding of the individuals in the base populations denoted by P^t in Alg. 1. The permutation-based representation allowed us to apply some additional types of genetic operators, which increases thus the number of possible configurations in the genetic engine of HGS-Sched.

4.3. Initial population

The initial population in GA is usually generated randomly. However in the case of scheduling problems it can be useful to combine the random method with another heuristic for creating a small amount of individuals (usually single individual) to increase the diversity of the population. The detailed characteristics of several initialization methods for scheduling in Grids can be found in [22]. In our approach we propose the *MTC + LJFR-SJFR* method for the initialization of the population in HGS-Sched root. In this method all but two individuals are generated randomly. Two solutions are created by using the *Longest Job to Fastest Resource - Shortest Job to Fastest Resource (LJFR-SJFR)* and *Minimum Completion Time (MCT)* heuristics.

In LJFR-SJFR heuristic both makespan and flowtime are minimized simultaneously (LJFR (min makespan) is alternated with the SJFR (min flowtime)). Initially the number of *nb_machines* tasks with the highest workload are assigned to the available machines sorted increasingly to their computing capacities. Then the remaining unassigned tasks are allocated in the remaining resources. In the MCT heuristic, a given task is assigned to the machine yielding the earliest completion time.

4.4. Genetic operators

The genetic procedures used in Alg. 1 are selected from the following set of operators:

- **Selection operators:** Linear Ranking Selection, N-Tournament Selection;
- **Crossover operators:** One Point Crossover, PMX, OX, CX;
- **Mutation operators:** Move, Swap, Rebalancing;
- **Replacement operators:** Steady State, Elitist Generational.

All those operators are typical for genetic algorithms used in the combinatorial optimization. We briefly describe here the mutation operators whose definition is more problem dependent.

We consider in this paper three variants of mutation operators: *Move*, *Swap* and *Rebalancing*. In *Move* mutation a task is moved from one machine to another one. Although the task can be appropriately chosen, this mutation strategy tends to unbalance the number of jobs per machine. In *Swap* mutation the number of task assigned to the given machine remains unchanged, but two tasks are swapped in two different machines. Finally, the *Rebalancing* method makes use of the load balancing technique.

4.5. HGS-Sched branching operators

In this section we define two branching operators for HGS-Sched, namely *HGS-Sched Sprouting Operator (SO)* and *Branch Comparison Operator (BC)*. We used them for the extension of the tree structure of the strategy. *BC* operator is the main mechanism in *Branch Reduction Operation*, which reduces the number of branches operating in the same region in the optimization domain.

The sprouting operator *SO* operator is defined by the following formulae:

$$SO(P^e_{i,j}) = (P^e_{i,j}, P^0_{i',j+1}), \tag{7}$$

where $\widehat{x}$ is the best adapted individual found in the parental branch $P^e_{i,j}$ after execution of e-th meta-epoch.

The outcome of the operator SO is the initial population for a new branch of degree $j+1$ denoted by $P^0_{i',j+1}$, where $i' = (i_1, \ldots, i_{j-1}, 1, 0, \ldots, 0)$. The individuals for that population are selected from the s_j-neighborhood ($1 \leq s_j \leq nb\_tasks$) of the best adapted individual $\widehat{x}$ in the parental population $P^e_{i,j}$. This neighborhood can be defined by using the operator A_{s_j}, which "cuts out" a s_j-length prefix from a given chromosome x, i.e.:

$$A_{s_j}(x) = \{\tilde{x}, |\tilde{x}| = s_j, |x| \geq s_j\}, \tag{8}$$

where $|x|$ denotes the length of x.

The s_j-neighborhood of the schedule x contains all individuals which can differ from x by $(n - s_j)$-length suffixes in their genotypes. It can be achieved by the permutation of tasks in the suffixes in the permutation representation, or task transfers to another machines in the suffixes in the case of direct representation of individuals.

The values of s_j are different in branches of the different degrees. These parameters are calculated using the following formulae:

$$s_j = s^j \cdot nb\_tasks, \tag{9}$$

where $s \in [0, 1]$ is a global strategy parameter called *neighborhood parameter* and j is the branch degree.

The sprouting operation can be activated or not, depending on the outcome of *Branch Comparison* operator. This operator is also used in the *Branch Reduction Operation* for the reduction of the branches of the same degree operating in the same (or similar) s_j- neighborhood.

The BC operator is defined in the following way: $BC : Q \rightarrow \{0, 1\}$ and :

$$BC(X, Y, s_j) = \begin{cases} 1, & \exists x \in X, \exists y \in Y : \ A_{s_j}(x) = A_{s_j}(y) \\ 0, & \text{otherwise,} \end{cases} \tag{10}$$

where $Q = \{(X, Y, s_j)\}$ and X, Y- are the populations in branches of degrees j and $j+1$ respectively in the case of conditional sprouting of the new branch from the parental one. In the case of reduction of the numbers of branches This operator is activated after execution of at least two meta-epochs in the root. The value 1 means that no new branch can be sprouted form a given parental branch or that two populations in the branches of the same degree j evolve in the same (or closely located) s_j-neighborhoods.

The main disadvantage of the implementation of the BC operator in our previous approach [12] is its high computational cost. In fact, all populations on a given tree level have been scanned and individuals with the same prefixes should be indicated. In this work we propose the use of hash techniques to reduce the execution time of the BC procedure.

Similarly as in [25], where hash techniques were successfully applied for improving the efficiency of the Struggle Algorithm, we define the hash table with the *task-resource allocation* key

denoted by k. The value of this key is calculated as the sum of the absolute values of the subtraction of each position and its precedent in the s_j-length suffix in direct representation of the schedule vector (reading the suffix in a circular way). The hash function f_{hash} is defined as follows:

$$f_{hash}(k) = \begin{cases} 0, & k < k_{min} \\ \left\lfloor N \cdot \left(\frac{k - k_{min}}{k_{max} - k_{min}} \right) \right\rfloor & k_{min} \leq k < k_{max} \\ N - 1, & k \geq k_{max} \end{cases} \tag{11}$$

where k_{min} and k_{max} correspond respectively to the smallest and the largest value of k in the population, and N is the population size.

In the case of the conditional sprouting of the new branches of the degree $j + 1$ from the parental branch of the degree j the keys are calculated for the best individual in the parental branch and individuals in all populations in all active branches of the degree $j+1$. If there is any individual in the higher degree branches, for which the key matches the key of the best adapted individual in the parental branch, then the value of BC is 1 and no branch of the degree $j + 1$ is sprouted in that moment.

In the case of the comparison of the branches of the same degree j, all branches, in which there exists the individuals with the identical keys have to be reduced and a single joint branch is created (the value of BC is 1). The individuals to this branch are generated using the implemented replacement mechanism from the union of the populations from all reduced branches.

5. Experimental study

In this section we present the results of the experimental evaluation of our HGS-Sched implementation for static and dynamic versions of the scheduling problem in Grid. We integrated the C++ code for HS-Sched algorithm with the Grid simulator *HyperSim-G* presented in [7, 27].

Although the aim of our experimental analysis was to test the HGS-based scheduler in a dynamic environment, we found it useful to initially evaluate the proposed method for a benchmark of static instances. We conducted our experimental study into two steps. First we validated HGS-Sched meta-heuristic by comparing it with state-of the-arts GAs results in the literature on a classical benchmark of the problem (Subsec. 5.1). After that, we evaluated the efficiency of four GA-based meta-heuristics on two benchmarks composed by a set of large size static (Subsec. 5.3.1) and dynamic (Subsec. 5.3.2) instances generated using a Grid simulator [21]. Finally, the statistical analysis of the results is given in Subsec. 5.4.

5.1. Evaluation of the algorithm on a static benchmark

The objective of the study presented in this section is to show the efficiency of the HGS-Sched algorithm on the classical benchmark (see below). For that, our algorithm is compared versus some other GA-based schedulers that were previously applied to the problem.

Table 1: Comparison of crossover operators for makespan and flowtime values.

Operator	Average Makespan	Average Flowtime
PMX	8182258.77332	1083433775.989
OX	9283953.76642	1253654022.331
CX	**7684490.43898**	**10452860938.452**

5.1.1. Benchmark description

The subset of static instances was taken from a larger benchmark, whose instances are classified into 12 types of ETC matrix, according to task heterogeneity, machine heterogeneity and consistency of computing. There are 512 tasks and 16 machines defined in each instance. Instances are labelled by *u_x_yyzz.0*, where

- *u* denotes the uniform distribution used in generating the ETC matrix;
- *x* means the type of consistency (*c*-consistent, $\tilde{i}$-inconsistent and *s* means semi-consistent);
- *yy* indicates the heterogeneity of the jobs (*hi* means high, and *lo* means low);
- *zz* expresses the heterogeneity of the resources (*hi* means high, and *lo* means low).

We started our experimental study by finding an appropriate combination of genetic operators for HGS-Sched. Then we proceeded with a short comparison analysis of the performances of our algorithm and GA-based schedulers presented in [4] and [22]. We used the hash procedure for the reduction the ineffective branches in HGS-Sched structure (see Section 4.5). For the initial population, one individual was generated using LJFR-SJFR, and the rest of individuals were obtained by largely and randomly perturbing it.

5.2. Tuning of GA operators for HGS-Sched

The sizes of populations for HGS-Sched branches were set as follows:

- base populations in root and sprouted branches of degree 1: 50 and 18 respectively;
- the intermediated populations: 48 and 16 – in the case of Elitist replacement method; 38 and 14 – in the case of Steady State replacement method.

Each experiment was repeated 30 times under the same configuration of operators and parameters and the average makespan and flowtime values were computed.

Table 1 shows the results obtained for selected crossover operators and Rebalancing mutation. In fact, the experiments were restricted to the permutation-based methods, as the operators of this representation showed to outperforms those of direct representation [12, 22]. As can be seen from Table 1, CX outperforms the rest of the operators for both makespan and flowtime criteria.

The results of the tuning the mutation operators are presented in Table 2. The three selected mutation methods (Move, Swap and Rebalancing) were combined them with CX operator. Rebalancing showed to outperform significantly Move and Swap mutation operators.

Table 2: Comparison of mutation operators for makespan and flowtime values.

Operator	Average Makespan	Average Flowtime
Move	8853148.607664	1087530440.192
Swap	19031484.397466	2283034176.802
Rebalancing	**7696260.50065**	**10413560968.452**

Table 3: Comparison of performance of replacement operators.

Operator	Average Makespan	Average Flowtime
Steady State	7701052.84523	1041400700.341
Elitist Generational	**7696260.50065**	**10413860968.452**

Regarding the replacement of individuals, two methods were selected: the Steady State and Elitist Generational methods. The results for both methods are given in Table 3. The Elitist Generational performed better although the differences in the results are minor. We then decided to apply Elitist Generational in the tests with the static benchmark and to use both methods in two alternate HGS-Sched implementation for the study of larger size instances (static and dynamic case).

5.2.1. Comparison analysis of GA-based schedulers

Once the operators performing best were identified, we used the best combination of HGS-Sched operators to compare the HGS-Sched performance with that of two other GA-schedulers presented in Braun et al. [4] and Xhafa et al. [22].

The values of HGS-Sched parameters for all tests are given in Table 4. The $nb\_of\_metaepochs$ defines a global stopping criterion for the whole strategy, which is usually the maximal number of meta-epochs run in the root of the structure. We denoted by $cross\_prob$ the probability of crossover in the branches of all degrees and by $mut\_prob(0)$ and $mut\_prob(1)$ the probabilities of mutation for the root and the branches of degree 1, respectively. The linear ranking method was used as selection procedure.

In order to make a fair comparison we defined for all algorithms the *work* parameter W, which expresses the amount of work of the particular scheduler. The objective was to observe the performance of the algorithms within the similar amount of work, even though they use different genetic operators and parameters. We defined the work of GA in Eq. (12).

$$W = (mut\_prob + cross\_prob) \times pop\_size \times nb\_of\_generations \quad (12)$$

Table 4: HGS-Sched global parameter values

Parameter	Value
degrees of branches (j)	0 and 1
$period\_of\_metaepoch$ - (k)	200
$nb\_of\_metaepochs$	10
neighborhood parameter - (s)	0.5
$mut\_prob(0)$	0.4
$mut\_prob(1)$	0.2
$cross\_prob$	0.8

For HGS-Sched parameter W is calculated as given in Eq. (13).

$$W = \Big[(mut\_prob(0) + cross\_prob) \times pop\_size(0) + \\ + \sum_{j=1}^{M} \left((mut\_prob(j) + cross\_prob) \times \right. \\ \left. \times pop\_size(j) \times nb\_br(j)\right)\Big] \times period\_of\_metaepoch \times nb\_of\_metaepochs \quad (13)$$

where:

- M is the maximal degree of the branch,
- $nb\_br(j)$ is the number of sprouted branches of degree j,
- $pop\_size(j)$ denotes the size of population in the branch of degree j.

Based on Eqs. (12) and (13), the amount of work of three analyzed schedulers are as follows:

- for GA in [4]: $W = (0.4 + 0.6) * 200 * 1000 = 200000$,
- for GA in [22]: $W = (0.4 + 0.8) * 68 * 2500 = 240000$, and
- for HGS-Sched: $W = W = ((0.2 + 0.8) * 8 * 12 + (0.4 + 0.8) * 28) * 200 * 10 = 259200$.

As can be seen form the above calculation, the values of work parameters are in the same range ensuring thus a fair comparison between the considered algorithms.

Comparison of makespan values.. We present in Table 5 the average makespan values obtained by the three GA-based schedulers.

Table 5: Comparison of average makespan values for three genetic algorithms.

Instance	Braun et al.	Xhafa et al.	HGS-Sched
u_c_hihi	8050844.500	7610176.437	**7607223.586402**
u_c_hilo	156249.200	155251.196	**154944.93441**
u_c_lohi	258756.770	248466.775	**247899.224211**
u_c_lolo	5272.250	5226.972	**5216.134866**
u_i_hihi	3104762.500	**3077705.818**	3078022.298866
u_i_hilo	75816.130	75924.023	**75699.81442**
u_i_lohi	107500.720	**106069.101**	108646.831972
u_i_lolo	2614.390	**2613.110**	2620.150313
u_s_hihi	4566206.000	4359312.628	**4343467.785157**
u_s_hilo	98519.400	98334.640	**98179.968964**
u_s_lohi	130616.530	127641.889	**126822.766276**
u_s_lolo	3583.440	**3515.526**	3555.009945

It can be observed that HGS-Sched outperforms the GA by Braun et al. for all instances and the GA by Xhafa et al. for 75% of considered instances. The HGS-Sched meta-heurisctic showed to be most effective for consistent and semi-consistent ETC matrices. This observation is useful in considering HGS-Sched as a basis for Grid scheduler that could have consistent computing features.

Table 6: Average flowtime values for benchmark instances (in arbitrary time units).

Instance	**Xhafa et al.** Steady State GA	**Xhafa et al.** Struggle GA	**HGS-Sched**
u_c_hihi	1048333229	1039048563	**1038168110.125**
u_c_hilo	27687019.4	27620519,	**27428920.1234**
u_c_lohi	34767197.1	34566883,8	**34240865.3567**
u_c_lolo	920475.17	917647,31	**915823.12398**
u_i_hihi	378010732	379768078	**356653620.984**
u_i_hilo	12775104.7	12674329,1	**12266233.125**
u_i_lohi	13444708.3	13417596,7	**12921130.58765**
u_i_lolo	446695.83	440728,98	**440255.6532**
u_s_hihi	526866515	524874694	**521943320.3322**
u_s_hilo	16598635.5	**16372763,2**	16446320.578
u_s_lohi	15644101.3	15639622,5	**15333843.1117**
u_s_lolo	605375.38	**598332,69**	602810.81222

Comparison of flowtime values.. We report in Table 6 the flowtime values obtained in the experimental study with the Steady State GA presented in [22], Struggle GA presented in [24] and HGS-Sched algorithm (Braun et al. did not report the flowtime values). From the results, we can conclude that HGS-Sched outperforms Steady State GA for all types of ETC matrices and it provides better results than Struggle GA for ten instances. In the case of flowtime, HGS-Sched performed coherently for all three groups of instances (consistent, semi-consistent and inconsistent ETC matrices).

Results of summative evaluation for the static benchmark.. The results of the experimental comparison analysis of the performance of three GA-based schedulers on the static benchmark show that HGS-Sched is more effective than two other GA-based meta-heuristics in simultaneously minimizing makespan and flowtime. The reduction of makespan by the HGS-Sched was not so efficient only for the inconsistent ETC matrices.

5.3. Experimental study of GA schedulers on large-size static and dynamic instances

Following the evaluation of HGS scheduler using instances of the static benchmark, we analyzed next its performance using realistic Grid scenarios. The benchmark of new instances of larger sizes has been generated by the *HyperSim-G* Grid simulator [21].

Our experiments were conducted for four variants of GA-based schedulers, namely *GA-Elitist Generational*, *GA-Steady State*, *HGS-Elitist Generational* and *HGS-Steady State*. The GA and HGS parameter settings are given in Tables 7and 8, resp. (Similar parameter settings for GA-based schedulers were used in [27].)

The performance of GA-based schedulers was analyzed for two types of Grid environment: static and dynamic. In the former, the number of tasks and the number of machines remain constant during the simulator run. In the later case, the number of tasks and machines may vary over time. In both cases four Grid size scenarios: small (32 hosts/512 tasks), medium (64 hosts/1024 tasks), large (128 hosts/2048 tasks), and very large (256 hosts/4096 tasks). The capacity of the resources and the workload of tasks are randomly generated by a normal distribution, denoted $N(\mu, \sigma)$. It is assumed that all tasks submitted to the system must be scheduled and all machines in the system can be used.

Table 7: GA settings for large static and dynamic benchmarks

Parameter	Elitist Generational	Steady State
evolution steps	$5 * (nb\_jobs)$	$20 * (nb\_jobs)$
population size (pop_size)	$\lceil(\log_2(nb\_jobs))^2 - \log_2(nb\_jobs)\rceil$	$4 * (\log_2(nb\_jobs) - 1)$
intermediate pop.	$pop\_size - 2$	$(pop\_size)/3$
selection method	LinearRanking	
crossover method	CX	
cross probab.	0.8	1.0
mutation method	Rebalancing	
mutation probab.	0.2	
replace_only_if_better	false	
replace_generational	false	
initialization	LJFR-SJFR + MCT + Random	
max_time_to_spend	40 secs (*static*) / 25 secs (*dynamic*)	

Table 8: HGS-Sched settings for large static and dynamic benchmarks

Parameter	Elitist Generational	Steady State
period_of_metaepoch	$5 * (nb\_jobs)$	$20 * (nb\_jobs)$
nb_of_metaepochs	10	
degrees of branches (j)	0 and 1	
population size in the root(r_pop_size)	$3 * (\lceil((\log_2(nb\_jobs))^2/ - \log_2(nb\_jobs))/(10.4)\rceil$	$3 * (\lceil 4 * (\log_2(nb\_jobs) - 1)/(11.8)\rceil)$
population size in the sprouted branches (b_pop_size)	$(\lceil((\log_2(nb\_jobs))^2/ - \log_2(nb\_jobs))/(10.4)\rceil$	$(\lceil(4 * (\log_2(nb\_jobs) - 1)/(11.8)\rceil)$
intermediate pop. in the root	$r\_pop\_size - 2$	$abs((r\_pop\_size)/3)$
intermediate pop. in the sprouted branch	$b\_pop\_size - 2$	$abs((b\_pop\_size)/3)$
selection method	LinearRanking	
crossover method	CX	
cross probab.	0.8	1.0
mutation method	Rebalancing	
mutation probab. in root	0.4	
mutation probab. in the sprouted branches	0.2	
replace_only_if_better	false	
replace_generational	false	
initialization	LJFR-SJFR + MCT + Random	
max_time_to_spend	70 secs (*static*) / 40 secs (*dynamic*)	

Table 9: Setting for the grid simulator for generating large static instances

	Small	Medium	Large	Very Large
Number of hosts	32	64	128	256
Resource capacities (in MIPS)	$N(1000, 175)^*$			
Total number of tasks	512	1024	2048	4096
Workload of tasks	$N(250000000, 43750000)$			
Host selection	All			
Task selection	All			
Number of runs	30			

Table 10: Makespan values (±s.d.) for large static instances (s.d.: standard deviation)

Meta-heuristic	Small	Medium	Large	Very Large
GA-Elitist Generational	3988113.784	4007586.118	4023746.479	4169054.398
	(±72006.550)	(±50105.695)	(±42185.823)	(±25747.948)
GA-Steady State	3988118.830	3993515.185	**4021368.781**	**4127259.199**
	(± 72001.378)	(±49728.050)	(±41201.608)	(±25872.462)
HGS-Elitist Generational	**3988112.261**	**3992434.325**	4045232.077	4134498.487
	(± 72008.149)	(±49942.644)	(±45579.987)	(±26874.793)
HGS - Steady State	4012057.101	4108747.951	4121977.943	4136884.354
	(±78200.951)	(±58994.804)	(±41121.498)	(±25842.542)

5.3.1. Experimental study using Grid static instances

The parameter setting for the simulator for static Grid scenarios is presented in Table 9.

Each experiment was repeated 30 times under the same configuration of operators and parameters. The averaged makespan and flowtime values obtained by four GA-based schedulers are presented in Tables 10 and 11.

The best results for makespan and flowtime in the cases of small and medium Grid sizes were achieved by HGS-Elitist Generational scheduler. In the larger Grid systems GA-Steady State algorithm outperform the other meta-heuristics for both makespan and flowtime. It can be observed that as the instance size is doubled, the makespan values increase slowly while the flowtime values increase considerably.

5.3.2. Experimental study in dynamic setting

In this section we applied four considered genetic-based schedulers to dynamic instances in order to evaluate them in more realistic Grid scenario. The benchmark of the dynamic instances

Table 11: Flowtime values (±s.d) for large static instances (s.d.: standard deviation)

Meta-heuristic	Small	Medium	Large	Very Large
GA - Elitist Generational	1085676446	2169597733.454	4336424647.141	8694552320.869
	(±20217021.689)	(±27233178.413)	(±44432177.763)	(±56530933.972)
GA - Steady State	1085655077.793	2169431289.314	**4334715011.418**	**8692986359.587**
	(±20371529.692)	(±27062114.940)	(±44455953.563)	(±56442033.933)
HGS - Elitist Generational	**1085575340.426**	**2169308217.698**	4336077721.299	8693447951.636
	(±20326272.800)	(±27063647.498)	(±44327155.275)	(±56288034.985)
HGS - Steady State	1086071183.744	2169852393.768	4336176645.169	8693540827.579
	(±20160724.169)	(±27327858.722)	(±44507669.562)	(±56254201.350)

Table 12: Settings for the dynamic grid simulator

	Small	Medium	Large	Very Large
Init. hosts	32	64	128	256
Max. hosts	37	70	135	264
Min. hosts	27	58	121	248
MIPS	$N(1000, 175)$			
Add host	$N(625000, 93750)$	$N(562500, 84375)$	$N(500000, 75000)$	$N(437500, 65625)$
Delete host	$N(625000, 93750)$			
Total tasks	512	1024	2048	4096
Init. tasks	384	768	1536	3072
Workload	$N(250000000, 43750000)$			
Interarrival	$E(7812.5)^{\dagger}$	$E(3906.25)$	$E(1953.125)$	$E(976.5625)$
Activation	Resource_and_time_interval(250000)			
Reschedule	True			
Host select	All			
Task select	All			
Number of runs	30			

Table 13: Makespan values (±s.d)for dynamic instances (s.d.: standard deviation)

Strategy	Small	Medium	Large	Very Large
GA - Elitist Generational	4116952.350	4071583.284	4093643.875	4101176.899
	(± 88997.075)	(±82755.099)	(±65754.879)	(±19663.984)
GA - Steady State	4116828.792	4086775.533	4080009.908	4098693.322
	(±88003.621)	(±94409.376)	(±78694.349)	(± 49044.581)
HGS - Elitist Generational	4041720.386	4049342.089	**4074756.208**	4102796.820
	(±64604.330)	(±53679.081)	(±41417.522)	(±27351.630)
HGS - Steady State	**4029602.691**	**4036987.551**	4079790.083	**4096857.780**
	(±62158.628)	(±57950.032)	(±28385.311)	(±26850.714)

(from 32 to 256 machines) were defined using the same simulator as in the static case described in the Subsec. 5.3.1. The settings for the applied dynamic grid version are presented in Table 12.

The number of hosts initially activated in the Grid environment is defined by the parameter *Init.hosts*. The parameters *Max.hosts* and *Min.hosts* specify the range of changes in the number of active hosts during the simulation process. The frequency of appearing and disappearing resources is defined by the normal distributions given by *Add host* and *Delete host*, while the initial number of tasks is given by *Init. tasks*. New tasks can arrive at the system with the frequency *Interarrival* until *Total tasks* is reached. The *Activation* parameter establishes the activation policy according to an exponential distribution. The already assigned tasks which have not been executed yet cannot be rescheduled if the value of the boolean parameter *Reschedule* is false.

Similarly as in the static case the GA- and HGS-based schedulers were plugged into the simulator. The generated problem instance is passed to the particular scheduler by the execution of the `schedule` procedure in the simulator. The scheduler finds a solution, which is then sent back to the simulator to allocate tasks to machines. Tasks assigned to machines not available in the Grid will be re-scheduled in next batch. The results of experiments performed separately for each meta-heuristic were averaged over 30 independent runs of the simulator (see Tables 13 and 14).

It can be observed from the obtained results that HGS-Sched with Steady State replacement

Table 14: Flowtime values (±s.d.) for dynamic instances (s.d.: standard deviation)

Strategy	Small	Medium	Large	Very Large
GA - Elitist Generational	1088624728.827	2161958805.835	4325774055.778	8667695620.357
	(±20327874.283)	(±27840056.635)	(±53264525.057)	(±59203131.427)
GA - Steady State	1089239424.050	2161600149.176	4326061767.612	8663803028.647
	(±20401980.079)	(±21185288.886)	(±53514052.118)	(±57499052.221)
HGS - Elitist Generational	**1071455164.177**	2149393085.396	4309915896.495	8662288096.550
	(±15440799.154)	(±22459040.577)	(±34474855.407)	(±24537584.957)
HGS - Steady State	1072962417.032	**2147265387.563**	**4308229866.603**	**8658644141.606**
	(±18783553.345)	(±19506908.647)	(±34146974.186)	(±22299941.828)

method outperforms other three meta-heuristics in all but two instances in the minimization of both makespan and flowtime metrics. In two cases - small grid for flowtime and large grid for makespan – the HGS-Sched with Elitist Generational replacement holds the best results. It should be also noticed that similarly as in the case of the large static instances (see Subsec. 5.3.1), as the instance size is doubled, the makespan value increases slowly while the flowtime value is doubled.

5.4. Statistical analysis of the results

The results sampled in Tables 10- 14 give a general information about the schedulers' performance. Additionally we employed two statistical methods, namely the coefficient of variation and the Student's t-test for means, to analyze the statistical significance of the results.

Coefficient of variation. The *coefficient of variation (CV)* [17] is defined as the statistical measure of the dispersion of data around the average value. It expresses the variation of the data as a percentage of its mean value, and is calculated as follows:

$$CV(x) = \frac{s.d.}{mean(x)} \cdot 100\% \tag{14}$$

The CV statistic is a useful in the analysis of the data series. It can also provide a general analysis of the performance of the method used for generating the data. CVs calculated for stable heuristic methods should not be greater than 5%. We present in Table 15 the coefficients of variation calculated for all makespan and flowtime results achieved by four applied genetic-based schedulers in the cases of the large size static and dynamic benchmarks (see Tables 10– 14).

It can be observed that in all instances the values of CVs are in the range $0-2\%$ which confirms the stability of the applied algorithms. Each meta-heuristic achieved the best results in the case of very large Grid with regard to both makespan and flowtime measures.

Student's t–test for means. Following the stability analysis of the applied meta-heuristics we performed a simple statistical comparison analysis of the methods on the same sample data, in order to confirm whether the methods provide similar analytical results or not. We used for that study the Student's t–test for the comparison of two means [17]. The outcome of this test is the acceptance or rejection of the null hypothesis ($H0$), which states that any differences in results are purely due to random. An erroneous rejection of the null hypothesis constitutes a Type 1 error.

Table 15: Comparison of the Coefficients of Variation (C.V.) for flowtime and makespan values in the large size static and dynamic instances

Large Size Static Instances								
Strategy	**Makespan**				**Flowtime**			
	Small	**Medium**	**Large**	**Very Large**	**Small**	**Medium**	**Large**	**Very Large**
GA - Elitist Generational	1.80	1.25	1.04	0.61	1.86	1.25	1.02	0.65
GA - Steady State	1.80	1.24	1.02	0.62	1.87	1.24	1.02	0.64
HGS - Elitist Generational	1.95	1.25	1.12	0.64	1.87	1.24	1.02	0.64
HGS - Steady State	1.94	1.43	0.99	0.62	1.85	1.25	1.02	0.64
DYNAMIC INSTANCES								
GA - Elitist Generational	2.16	2.03	1.60	0.47	1.86	1.28	1.23	0.68
GA - Steady State	2.13	2.31	1.92	1.19	1.87	0.98	1.23	0.66
HGS - Elitist Generational	1.59	1.31	1.01	0.66	1.44	1.04	0.79	0.28
HGS - Steady State	1.54	1.43	0.69	0.65	1.75	0.90	0.79	0.25

Table 16: Comparison of the two-tailed P-values for flowtime and makespan results in the large size static and dynamic instances

Large Size Static Instances								
Strategy	**Makespan**				**Flowtime**			
	Small	**Medium**	**Large**	**Very Large**	**Small**	**Medium**	**Large**	**Very Large**
GA - Elitist Generational	0.999	0.243	0.826	< **0.001**	0.984	0.967	0.882	0.914
GA - Steady State	0.999	0.933	1	1	0.987	0.986	1	1
HGS - Elitist Generational	1	1	**0.037**	0.292	1	1	0.905	0.974
HGS - Steady State	0.222	< **0.001**	< **0.001**	0.154	0.924	0.938	0.899	0.969
DYNAMIC INSTANCES								
GA - Elitist Generational	< **0.001**	0.065	0.188	0.480	**0.0005**	**0.021**	0.134	0.436
GA - Steady State	< **0.001**	**0.016**	0.747	0.857	**0.0003**	**0.008**	0.129	0.648
HGS - Elitist Generational	0.462	0.395	1	0.399	1	0.696	0.849	0.549
HGS - Steady State	1	1	0.585	1	0.735	1	1	1

We compared the best results for the makespan and flowtime values in the static and dynamic instances (shown in bold in Tables 10–14) with the remained results for the same Grid type. We applied t-test within the predefined confidence level *C.L.*, which was 95 % in our approach. In table 16 we report the probabilities of Type 1 errors (P-values) [17], which give us the adequate information to judge the acceptance or the rejection of the null hypothesis. The difference in results is not statistically significant if the P-value is not grater than 0.05 (P-value is 1 for the base (best) results to which the remained results are referred).

It can be observed that in the static instances HGS- based schedulers are not significantly better than GA meta-heuristics. In the case of 'very large' Grid the values of makespan achieved by GA-Steady State algorithm are much lower (statistically) than those achieved by the other meta-heuristics.

An interesting conclusion can be drawn from the analysis of the results in the dynamic instances. It can be seen that HGS-based schedulers outperform GAs in the cases of small and medium Grids, while in the larger systems the differences in results are not statistically significant. It seems that HGS-Sched is the most effective in scheduling of small to medium size batches of tasks in the case where the structure of the Grid is dynamically changed. It can make this method very useful in the real-life scenario, in which the scheduling process can be activated with a high frequency for small pools of tasks.

6. Conclusions

In this work we have presented the implementation and evaluation of Hierarchic Genetic Strategies (HGS) for Independent Batch Job Scheduling on Computational Grids for which both makespan and flowtime parameters are simultaneously minimized. Achieving high quality planning of jobs into machines is crucial for Grid systems due to their highly heterogenous and dynamic nature. The work is motivated by a series of works on the literature on the use of Genetic Algorithms (GAs) for the problem. The interest here was, first to explore if the capability of HGS of concurrently searching the search space, as opposed to single population GAs, to achieve fast reductions in makespan and flowtime. Second, the objective has been to analyze the HGS scheduler under a dynamic environment using a Grid simulator.

In our HGS implementation, we have examined several variations of HGS operators in order to identify a configuration that works best for the problem. We also provide some enhancements of the HGS algorithms such as the use of hash techniques for an efficient branch comparison operator. A three-fold experimental studied was carried out to analyze the performance of the HGS-based scheduler. Initially, we used a static benchmark of rather small size instances that has reported in the literature. For the sample of instances selected from the benchmark, the HGS-based scheduler showed to outperform the state-of-the-art GA-based schedulers. Second, we used a Grid simulator to evaluate the HGS-based scheduler in a more realistic setting. In this case, two scenarios were considered, namely the number of tasks and hosts is kept constant and the tasks and machines are dynamically changing their availability. Finally, in both cases, the Grid scenarios ranged from small to very large size instances in order to see the scalability of HGS-based scheduler. The experimental results showed a very good performance, whose validity was also confirmed by statistical analysis of coefficient of variance and Student t-test methods.

References

[1] Abraham, A., Buyya, R., and Nath B.: "Natures heuristics for scheduling jobs on computational grids", *Proc. of the 8th IEEE International Conference on Advanced Computing and Communications*, India, 2000.

[2] Ali, S., Siegel, H.J., Maheswaran and Hensgen, D.: "Task execution time modeling for heterogeneous computing systems", *Proceedings of Heterogeneous Computing Workshop*, pp. 185–199, 2000.

[3] Armstrong, R., Hensgen, D., and Kidd, T.: "The relative performance of various mapping algorithms is independent of sizable varie,nces in run-time predictions", *7th IEEE Heterogeneous Computing Workshop (HCW '98)*, Mar. 1998, pp. 79–87.

[4] Braun, T.D., Siegel, H.J., Beck, N., Boloni, L.L., Maheswaran, M., Reuther, A.I., Robertson, J.P., Theys, M.D., Yao, B., Hensgen, D., and Freund, R.F.: "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.

[5] Brucker, P.: *"Scheduling Algorithms*, Springer, 2007.

[6] Buyya, R.: "Economic-based Distributed Resource Management and Scheduling for Grid Computing", *PhD Thesis*, Monash University, Australia, 2002.

[7] Carretero, J. and Xhafa, F.: "Using Genetic Algorithms for Scheduling Jobs in Large Scale Grid Applications", *Journal of Technological and Economic Development –A Research Journal of Vilnius Gediminas Technical University*, 12(1): 11–17, 2006.

[8] Garey, M. R., and Johnson, D. S.: *"Computers and Intractability A Guide to the Theory of NPCompleteness"*, Freeman and Co., 1979.

[9] Hotovy, S.: "Workload evolution on the Cornell Theory Center IBM SP2", in *Job Scheduling Strategies for Parallel* Proc. Workshop, IPPS'96, 27–40, 1996.

[10] Kołodziej, J., Gwizdała, R., and Wojtusiak, J.: "Hierachical Genetic Strategy as a Method of Improving Search Efficiency", *Advances in Multi-Agent Systems, R. Schaefer and S. Sędziwy eds.*, UJ Press, Cracow 2001, Chapter 9, 149–161.

[11] Kołodziej, J., Jakubiec, W., Starczak, M., and Schaefer R.: "Hierarchical Genetic Strategy Applied to the Problem of the Coordinate Measuring Machine Geometrical Errors", *Proc. of the IUTAM'02 Symposium on Evolutionary Methods in Mechanics*, 24-27 September 2002, Cracow, Poland, Kluver Ac. Press, 22–30.

[12] Kołodziej, J., Xhafa, F., Kolanko, Ł.: "Hierarchic Genetic Scheduler of Independent Jobs in Computational Grid Environment", *Proc. of 23rd ECMS, Madrid, 9-12.06.2009, in J. Otamendi, A. Bargieła, J.L. Montes and L.M. Doncel Pedrera eds.*, IEEE Press, Dudweiler, Germany, 2009, pp. 108–115.

[13] Kołodziej, J., and Rybarski, M.: "An Application of Hierarchical Genetic Strategy in sequential scheduling of permutated independent jobs", Warsaw University of Technology, Lectures on Eletronics, vol. 1, 2009 *Evolutionary Computation and Global Optimization, Jarosław Arabas Ed.*; WUT Press, 2009, pp. 95–103.

[14] Lim, D., Ong, Y.-S., Jin, Y.: "Efficient Hierarchical Parallel Genetic Algorithms Using Grid Computing", *Future Generation Computer Ssytem*, 23(4): 658–670, 2007.

[15] Liu, H., Abraham, A., and Hassanien, A.E.: "Scheduling jobs on computational grids using a fuzzy particle swarm optimization algorithm", *Future Generation Computer Ssytem*,doi:10.1016/j.future.2009.05.022.

[16] Maheswaran, M., Ali, S., Siegel, H. J., Hensgen, D., and Freund, R. F.: "Dynamic mapping of a class of independent tasks onto heterogeneous computing systems", *Journal of Parallel and Distributed Computing*, 59 (1999), pp. 107- 131.

[17] Mann, P. S.: *Introductory Statistics*, 7th ed., Wiley, 2010.

[18] Michalewicz, Z.: *"Genetic Algorithms + Data Structures = Evolution Programs"*, Springer, 1992.

[19] Ritchie, G., and Levine J.: "A fast effective local search for scheduling independent jobs in heterogeneous computing environments",*TechRep*, Centre for Intelligent Systems and Their Applications, University of Edinburgh, 2003.

[20] Schaefer, R. and Kołodziej J.: "Genetic Search Reinforced by The Population Hierarchy", *Foundations of Genetic Algorithms VII* , Morgan, 2003, pp. 369–385.

[21] Xhafa, F., Carretero, J., Barolli, L., and Durresi, A.: "Requirements for an Event-Based Simulation Package for Grid Systems, *Journal of Interconnection Networks*, 8(2): 163–178, 2007.

[22] Xhafa, F., Carretero, J.,and Abraham, A.: "Genetic Algorithm Based Schedulers for Grid Computing Systems", *International Journal of Innovative Computing, Information and Control*, 3(5): 1053–1071, 2007.

[23] Xhafa, F., Barolli L. and Durresi, A.: "Batch Mode Schedulers for Grid Systems", *International Journal of Web and Grid Services*, 3(1): 19–37, 2007.

[24] Xhafa, F., Barolli, L.,and Durresi, A.: "An Experimental Study On Genetic Algorithms for Resource Allocation On Grid Systems", *Journal of Interconnection Networks.* 8(4): 427–443, 2008.

[25] Xhafa, F., Duran, B., Abraham, A., and Dahal, K. P.: "Tuning Struggle Strategy in Genetic Algorithms for Scheduling in Computational Grids", *Neural Network World*, 18(3): 209–225, 2008.

[26] Xhafa, F., Gonzalez, J. A., Dahal, K. P., Abraham, A.: "A GA(TS) Hybrid Algorithm for Scheduling in Computational Grids", *HAIS 2009*, pp. 285–292.

[27] Xhafa, F., and Carretero, J.: "Experimental Study of GA-Based Schedulers in Dynamic Distributed Computing Environments", Chapter 24, In *Alba et al. Eds. Optimization Techniques for Solving Complex Problems*, Wiley, 2009.

[28] Yarkhan, A., and Dongarra, J.: "Experiments with scheduling using simulated annealing in a grid environment", *Proc. of the 3rd International Workshop on Grid Computing*, pp. 232–242, 2002.

[29] Zomaya, A. Y., and Teh, Y. H.: "Observations on using genetic algorithms for dynamic load-balancing", *IEEE Transactions On Parallel and Distributed Systems*, 12(9): 899–911, 2001.

Dr. Joanna Kołodziej graduated in Mathematics from the Jagiellonian University in Cracow in 1992, where she also obtained the PhD in Computer Science in 2004. She joined the Department of Mathematics and Computer Science of the University of Bielsko-Biała as an Assistant Professor in 1997. She serves as PC Co-Chair of PPSN 2010 as a PC member of many international conferences including CISSE 2006, ECMS 2007–2010 CEC 2008, IACS 2008–2009, ICAART 2009-2010.

Dr. Fatos Xhafa is currently a Visiting Professor at the Department of Computer Science and Information Systems, Birkbeck, University of London, UK. He is Associate Professor (with tenure) at the Technical University of Catalonia, Spain. His research interests include parallel and distributed algorithms, combinatorial optimization, distributed programming, Grid and P2P computing. He has widely published in international journals, books and conference proceedings of the research area. He serves the EB of nine peer-reviewed international journals and has also guest co-edited in several international journals. He has served and is currently serving as PC Co-Chair/General Co-Chair of several international conferences and workshops.