HIERARCHIC GENETIC SCHEDULER OF INDEPENDENT JOBS IN COMPUTATIONAL GRID ENVIRONMENT

Joanna Kołodziej
University of Bielsko-Biała
Department of Mathematics and Computer Science
ul. Willowa 2, Bielsko-Biała, Poland
Email: jkolodziej@ath.bielsko.pl

Fatos Xhafa
Department of Languages and Informatics Systems
Technical University of Catalonia
Campus Nord, Ed. Omega, C/Jordi Girona 1-3
08034 Barcelona, Spain.
Email: fatos@lsi.upc.edu

Łukasz Kolanko
University of Bielsko-Biała
Faculty of Mechanical Engineering and Computer Science
ul. Willowa 2, Bielsko-Biała, Poland
Email: lkolanko@ath.bielsko.pl

KEYWORDS

Scheduling, Computational grids, Genetic algorithms, Hierarchic genetic strategies, ETC matrix model.

ABSTRACT

In this work we present an implementation of Hierarchic Genetic Strategy (HGS) for Independent Job Scheduling on Computational Grids. In our formulation of the scheduling problem, makespan and flowtime parameters are simultaneously optimized. The efficient assignment of jobs to machines that optimizes both objectives is crucial for many Grid systems. The objective of this work is to examine several variations of HGS operators in order to identify a configuration of operators and parameters that works best for the problem. Differently from classical GA algorithms, which maintain only an unstructured population of individuals, HGS performs by many small populations enabling a concurrent search in the optimization domain. From the experimental study we observed that HGS implementation outperforms existing classical GA schedulers for most of considered instances of a static benchmark for the problem.

INTRODUCTION

Grid technologies have primarily emerged for scientific and technical work, where geographically distributed computers, linked through Internet, are used to create virtual supercomputers of vast amount of computing capacity which enables to solve complex problems from e-Science domain. During the past years Grid Computing has shown its feasibility to achieve breakthroughs in meteorology, physics, medicine and other computing-intensive fields. Grid computing is still in the development stage, and most projects are still from academia and large IT enterprises; it has however developed very quickly and more and more scientists are currently engaged to solve many challenges in Grid Computing.

One important challenge in Grid computing is the design and implementation of efficient Grid schedulers as a basis for the development of high performance Grid-enabled applications. Scheduling problem in Grid systems can be seen as a family of problems, depending on the way how the problem is formalized (realistic vs. simulated), the number of objectives to optimize (single vs. multi-objective), the environment (static vs. dynamic), processing mode (immediate vs. batch), job dependencies (independent vs. dependent), etc.

In this work we consider the scheduling problem formulation, referred to as Independent Job Scheduling, in which: (a) expected time to compute values of jobs in machines are known; (b) two objectives, namely makespan and flowtime are optimized (bi-objective optimization); (c) the environment is dynamic; (d) jobs are processed in batch mode; and, (d) jobs are independent (or loosely coupled).

Independent job scheduling appears usually in data intensive computing such as data mining and massive processing applications. For this class of Grid-enabled applications, batch mode is appropriate given that such application manage voluminous data, could run periodically and also because large amount of data have to be transferred, replicated and accessed by the applications. In such case, jobs or applications are grouped in *batches* and scheduled as a group. Batch mode scheduling methods are simple and yet powerful heuristics that are distinguished for their efficiency by taking better advantage of job and resource characteristics since they dispose of the time interval between two successive activations of the batch scheduler.

The objective in this work is to design and implement batch schedulers based on a family of dependent genetic processes enabling a concurrent search in the optimization domain. We defined a *Hierarchic Genetic Scheduler (HGS-Sched* which is based on the Hierarchical Genetic Strategy (HGS) defined by (Kołodziej et al., 2001).

HGS is an effective tool for solving continuous global

Proceedings 23rd European Conference on Modelling and Simulation ©ECMS Javier Otamendi, Andrzej Bargiela, José Luis Montes, Luis Miguel Doncel Pedrera (Editors)
ISBN: 978-0-9553018-8-9 / ISBN: 978-0-9553018-9-6 (CD)

optimization problems with multimodal and weakly convex objective functions. It was successfully applied to some practical engineering problems, for example for the estimation of the geometric errors of the Coordinate Measure Machine (Kołodziej et al., 2004). The successful applications of HGS make it a candidate algorithm for the design of efficient batch schedulers in Grid systems for which time efficiency and high quality planning are crucial.

The rest of the paper is organized as follows. We introduce the scheduling problem and its definition in Section "STATEMENT OF THE PROBLEM". Definitions of HGS-Sched procedures and a short description of applied genetic operators are given in Section "HIERARCHIC GENETIC SCHEDULER FOR GRID SYSTEMS". Section "EXPERIMENTAL STUDY FOR STATIC SCHEDULING" contains an experimental analysis of the performance of HGS-Sched and some selected singe-population genetic algorithms for the benchmark of static scheduling. The paper ends in Section "CONCLUSIONS" with some final remarks.

STATEMENT OF THE PROBLEM

Scheduling of jobs to Grid resources is indispensable for Computational Grids to ensure high performance Grid-enabled applications. Notice that the scheduling impacts, on the one hand, on Grid resource utilization, the efficiency of Grid applications and also on the QoS of the Grid system. Unlike traditional schedulers of conventional distributed systems, centralized Grid (global) schedulers have to manage the inherent heterogeneous structure of the Grid, the existence of local schedulers in different organizations or resources and the possible restrictions of the jobs to be scheduled (restrictions on the resources needed to complete the job, transmission cost, etc.).

In this work we consider the problem formulation based on the Expected Time to Compute matrix model. In this model (Ali et al., 2000), it is assumed that we know:

- The estimation or prediction of the computational load of each job (e.g. in millions of instructions). This information is usually given from job specification or is extracted from execution traces.
- The computing capacity of each resource (e.g. in millions of instructions per second, MIPS).
- The estimation of the prior load of each one of the machines. Machines in the system could be idle (in such case the prior load is zero) or could be running some job and have a pending queue of jobs. In this later case, the prior load is equal to the current load of the machine plus the computing load due to the execution of the pending jobs in the queue.
- The entries of the *ETC* matrix, denoted by $ETC[\tilde{j}][\tilde{m}]$ indicates the expected time to compute job $\tilde{j}$ in resource $\tilde{m}$. A simple way to compute $ETC[\tilde{j}][\tilde{m}]$ entry is by dividing the computing load of a job $\tilde{j}$ by the computing capacity of a machine $\tilde{m}$.

Further, the bi-objective optimization model is considered for the problem, namely the minimization of *makespan* and *flowtime*, where $F_{\tilde{j}}$ denotes the time when job $\tilde{j}$ finalizes and $Sched$ is the set of all possible schedules; the two objectives are integrated into a single objective function.

$$makespan = \min_{S_i \in Sched} \{ \max_{\tilde{j} \in Jobs} F_{\tilde{j}} \} \tag{1}$$

$$flowtime = \min_{S_i \in Sched} \{ \sum_{\tilde{j} \in Jobs} F_{\tilde{j}} \} \ . \tag{2}$$

Makespan expresses the performance of a Grid systems; the smaller the values of makespan, the closer to the optimal is the scheduling of jobs to Grid resources, and hence, the faster is the executions of jobs in the Grid system. On the other hand, flowtime, expresses the response time to the submitted job execution requests of Grid users and therefore, flowtime is usually considered as QoS criterion in Grid systems.

HIERARCHIC GENETIC SCHEDULER FOR GRID SYSTEMS - (HGS-Sched)

Many kinds of genetic algorithms defined for the scheduling problems (see Braun et al. (2001), Xhafa et al. (2007b)) can be implemented as *the law of evolution*, which governs the progression from one generation to the next. GA's objective is to explore as much as possible the solution space using a population of individuals and to find the best solution for the problem at hand. Due to the large size of the solution space, an intelligent space exploration is thus critical for GAs and is usually achieved through the use of genetic algorithms. It should be noted that classical GAs running in sequential setting define a single evolutionary process.

The main idea of HGS-Sched is running a set of dependent evolutionary processes. This dependency relation has a tree structure with the restricted number of levels. Every single process "creates" a branch in this structure and can be defined as a sequence of evolving populations. The core of the structure governs the search procedures and it is active until the stopping criterion for the whole strategy is satisfied. The HGS-Sched mechanism is different than the mechanism of other hierarchical and branching schedulers presented in the literature (see (Brucker , 2007) for example).

For each branch we define a *degree* parameter, denoted by $j \in \mathbb{N}$, the value of which corresponds to the accuracy of the search in this branch. There is a unique branch of the lowest degree 0 called *root*. The processes of lower degree (close to the root) represent chaotic search with a low accuracy. They detect the promising regions in

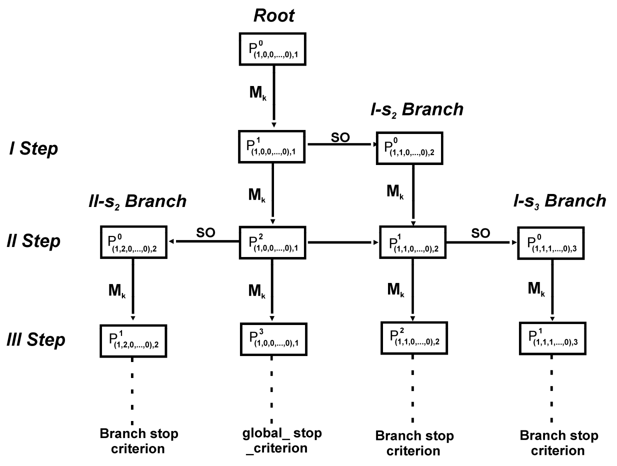

Figure 1: An example structure of HGS-Sched after execution of 3 metaepochs

the optimization landscape, in which more accurate processes of higher order are activated. A new branch could be created after running a metaepoch in the parental one.

Let us denote by $P^e_{i,j}$ populations evolving in branches of the different degrees, where $e \in \mathbb{N}$ is the global metaepoch counter, j, $(j \in \{1, \dots, M\}, M \in \mathbb{N})$ is a degree of the branch, M is the maximal degree of a branch and i –is the unambiguous branch identifier, $i = (i_1, \dots, i_M)$, $i_h = 0$ for $h > j$, which describes the "history of creation" of the given branch (Schaefer, and Kołodziej).

A k –periodic metaepoch $M_k, (k \in \mathbb{N})$ is a discrete evolution process terminating after k generations by the selection of the best adapted individual. The outcome of the metaepoch started from the population $P^e_{i,j}$ may be denoted by:

$$M_k\big(P^e_{i,j}\big) = \big(P^{e+l}_{i.j}, \widehat{x}\big); \qquad (3)$$

for some i, j, where $P^{e+l}_{i,j}$ is the resulting population and $\widehat{x}$ is the best adapted individual in the metaepoch.

An example structure of HGS-Sched is shown in Figure 1.

The algorithm starts from an initial population generated (usually randomly) for the root. A procedure of activation of a new process of higher degree is called a *Sprouting Operation*. It is performed conditionally, according to the outcome of a *Branch Comparison Operation*. The definitions of these operations together with the description of all operators implemented in the genetic engine of HGS-Sched are presented in the rest of this section.

Genetic Engine in HGS-Sched Branches

We used in our implementation a genetic algorithm template, defined in Algorithm1 below, for the description of a general framework of the genetic engine in HGS-Sched branches.

Similar template can be found in (Xhafa et al., 2007b).

Algorithm 1 Genetic engine template

```
Generate the initial population $P^0$ of size $\mu$;
Evaluate $P^0$;
while not termination-condition do
    Select the parental pool $T^t$ of size $\lambda$; $T^t := Select(P^t)$;
    Perform crossover procedure on pars of individuals in $T^t$ with probability $p_c$; $P^t_c := Cross(T^t)$;
    Perform mutation procedure on individuals in $P^t_c$ with probability $p_m$; $P^t_m := Mutate(P^t_c)$;
    Evaluate $P^t_m$ ;
    Create a new population $P^{t+1}$ of size $\mu$ from individuals in $P^t$ and/or $P^t_m$ ; $P^{t+1} := Replace(P^t; P^t_m)$
    $t := t + 1$;
end while
return Best found individual as solution;
```

Representations of Individuals and Genetic Operators

We used two methods of encoding of individuals for the representations of solutions of scheduling problems: *direct* and *permutation-based* representation (Xhafa et al., 2007b). The advantage in using two representations is that different families of operators can be defined for each of them, increasing thus the possibilities for operator configurations in GAs.

In **direct representation** solutions are encoded into a schedule vector $x = [x_1, \dots, x_n]^T$, where $x_i \in [1, m]$ for $i = 1, \dots, n$, n is the number of jobs and m is the number of machines. Coordinate i indicates the number of machine, to which job i is assigned.

The following vector $[1, 2, 1, 1, 3, 1, 2, 2, 3, 3]^T$ is a simple example of the schedule for 3 machines and 10 jobs.

In **permutation-based** representation first we define for each machine a sequence of jobs assigned to it, then we concatenate all obtained sequences into one ordered sequence (a permutation) of coordinates of job-machine allocation vector. It is also necessary to define an additional vector of size m with number of jobs assigned to the machines. An individual's chromosome is defined as a pair of vectors:

$$x = (u; v), u = [u_i, \dots, u_n]^T, v = [v_1, \dots, v_m]^T \qquad (4)$$

where $u_i \in [1, \dots, n]$ and $v_j \in [1, \dots, n]$.

The permutation-based representation for the schedule from the above example is as follows: $([1, 3, 4, 6, 2, 7, 8, 5, 9, 10]^T; [4, 3, 3]^T$. As can be seen, to machine 1 are assigned 4 jobs (1,3,4,6), to machine 2 are assigned 3 jobs (6,2,7), and so on.

Note that in the first representation many coordinates can have the same value, what does not cause in u in the permutation-based encoding. We applied both encoding methods in our work. We used direct representation for the individuals in base populations of the GA algorithm denoted by P^t in Algorithm 1. The permutation-based representation is necessary for running some crossover procedures defined below.

As the fitness function we define the bi-objective for scheduling given by the following formula:

$$fitness := makespan + flowtime, \qquad (5)$$

which is minimized in our work.

The initial population for the root of the HGS-Sched structure is selected randomly. The processes of higher degrees start from the populations obtained as the outcomes of the *Sprouting Operator* defined in the next section.

The genetic procedures used in a main loop of Algorithm 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:** Simple Generational, Elitist Generational.

All those operators are defined in GA algorithm for the problem in (Xhafa et al., 2007b). We briefly describe here the mutation operators whose definition is more problem dependent; the selection and replacement operators are defined as usually in GAs. Three definitions are considered for the mutation operator. `Move` moves a job from one machine to another one. Although the job can be appropriately chosen, this mutation strategy tends to unbalance the number of jobs per machine. The mutation *Swap*, keeps the number of jobs per machines unchanged by swapping two jobs in two different machines. Finally, *Rebalancing* makes use of the load balancing technique.

HGS-Sched Branching Operators

To extend the tree structure of HGS-Sched we introduced two branching operators: *HGS-Sched Sprouting Operator (SO)* and *Branch Comparison Operator (BC)*.

The *SO* operator, which is responsible for creation of new branches of higher degree, is defined by the following formula:

$$SO(P^e_{i,j}) = \big(P^e_{i,j}, P^0_{i',j+1}\big), \tag{6}$$

where $\widehat{x}$ is the best adapted individual found in the parental branch $P^e_{i,j}$ after execution of e-th metaepoch, $P^0_{i',j+1}$ is an initial population for a new branch of degree $j+1$ and $i' = \big(i_1, \ldots, i_{j-1}, 1, 0, \ldots, 0\big)$.

The individuals in population $P^0_{i',j+1}$ are selected from the s_j-neighborhood ($1 \leq s_j \leq n$) of the solution $\widehat{x}$ defined in the following way. Let A_{s_j} be the operator which "cuts out" a s_j-length prefix from a given chromosome x. It is defined as follows:

$$A_{s_j}(x) = \tilde{x}, |\tilde{x}| = s_j, |x| \geq s_j, \tag{7}$$

where $|x| = n$ denotes the length of x. The s_j-neighborhood of solution x contains all individuals which can differ from x by the $\big(n - s_j\big)$-length suffixes in their genotypes. It can be achieved by the permutation of jobs in the suffix in the permutation-based representation or job transfers to another machines in the suffix in the case of direct representation of individuals.

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

$$s_j = s^j \cdot n, \tag{8}$$

where $s \in [0,1]$ is a global strategy parameter called *neighborhood parameter*, j- the branch degree, n- the number of jobs.

The sprouting operation can be activated or not, depending on the outcome of *Branch Comparison* operator $BC : Q \to \{0,1\}$ defined below:

$$BC(X,Y,s_j) = \begin{cases} 1, & \exists x \in X, \exists y \in Y : \\ & A_{s_j}(x) = A_{s_j}(y) \\ 0, & \text{otherwise,} \end{cases} \tag{9}$$

where $Q = \{(X,Y,s_j)\}$ and X, Y- are the populations in branches of degrees j and $j+1$ respectively. In the case presented in Figure 1 the value of *BC* is 1. The value 0 means that the parental branch cannot sprout a branch, which is similar to existing branches of the same degree. This operator is activated after execution of at least two metaepochs in the root.

The codes of procedures for those two operators are very similar to the codes of *SO* and *PC* procedures for HGS, which can be found in (Kołodziej et al., 2001).

EXPERIMENTAL STUDY FOR STATIC SCHEDULING

We have conducted a first experimental evaluation of the HGS implementation for the problem. Although our ultimate goal is to test and evaluate the HGS-based scheduler in a dynamic environment, we found it useful to initially use a static benchmark for evaluating the HGS implementation and compare it with state-of-the-art GAs results for the scheduling problem.

Our experiments were performed for benchmark of static instances for scheduling problem classified into 12 types of ETC matrix, according to three parameters: job heterogeneity, machine heterogeneity and consistency. Each instance consists of 512 jobs and 16 machines and is labelled by *u_x_yyzz.0* like in (Braun et al., 2001) and (Xhafa et al., 2007b):

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

- x means the type of consistency (c–consistent, $\tilde{i}$–inconsistent and s means semi-consistent). An ETC matrix is considered consistent when, if a machine $m_{\tilde{i}}$ executes job $\tilde{j}$ faster than machine $m_{\tilde{j}}$, then $m_{\tilde{i}}$ executes all the jobs faster than $m_{\tilde{j}}$. Inconsistency means that a machine is faster for some jobs and

Table 1: HGS-Sched global parameters 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

slower for some others. An ETC matrix is considered semi-consistent if it contains a consistent submatrix.

- yy indicates the heterogeneity of the jobs (hi means high, and lo means low).

- zz indicates the heterogeneity of the resources (hi means high, and lo means low).

The benchmark tries to capture different real Grid scenarios by combining different characteristics of the Grid systems such as computing consistency, heterogeneity of resources and jobs.

The values of HGS-Sched parameters for all tests are given in Table 1.

The parameter $nb\_of\_metaepochs$ defines a global stop criterion for the whole strategy, which is the maximal number of metaepochs run in the root of the structure. We denoted by $mut\_prob(j)$ the probability of mutation in the branch of degree j ($mut\_prob(0)$ and $mut\_prob(1)$ are the probabilities of mutation for the root and the branches of degree 1 respectively). We assume that the probability of crossover, denoted by $cross\_prob$, takes the same value in the branches of all degrees. The initial population of the root was created by a uniform random generator and linear ranking method was used as selection procedure.

We conducted the experimental study into two steps. First, we tried to find an appropriate combination of genetic operators for HGS-Sched. Then, in the second step, we made a short comparison analysis of the performances of our algorithm and GA-based schedulers presented in (Braun et al., 2001) and (Xhafa et al., 2007b).

Tuning of Genetic Operators for HGS-Sched

A simple tuning process of genetic operators for HGS-Sched was performed for one instance of ETC matrix labeled by u_c_hihi.0. The sizes of populations in root and sprouted branches of degree 1 were 50 and 18, respectively. The intermediated populations consisted of 48 and 16 individuals in the case of Elitist replacement and 38 and 14 in the case of Simple Generational replacement. Each experiment was repeated 30 times under the same configuration of operators and parameters and the average makespan and flowtime values were computed.

Table 2 shows the results obtained for four types of crossover operators and Rebalancing mutation. CX and OnePoint methods are more effective than OX and PMX, while the differences in makespan and flowtime values calculated for them are minor. We decided to choose the CX operator for our strategy engine, based on the results of similar analysis presented in (Xhafa et al., 2007b).

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

Operator	**Average Makespan**	**Average Flowtime**
PMX	8147446.46338	1096287320.223
OX	9283953.76642	1249786030.004
CX	**7696260.50065**	10413560968.452
OnePoint	7696439.32214	**10413530955.129**

Table 3: 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**

The results of the comparison analysis of the performance of three mutation operators are reported in Table 3. We combined them with CX procedure. Rebalancing method outperforms significantly Move and Swap mutations.

The values of fitness components calculated for two considered replacement procedures are given in Table 4. The appropriate combination of genetic operators for HGS-Sched is completed with the Elitist Generational replacement method due its better performance.

Comparison Analysis of Genetic-Based Schedulers

Once fine tuned, the operators and parameters were used for a comparison analysis of three genetic algorithms, namely our implementation of HGS-Sched with GA schedulers defined in (Braun et al., 2001) and (Xhafa et al., 2007b). It should be noted that the GA in (Xhafa et al., 2007b) outperformed the GA in (Braun et al., 2001) and, it was thus interesting to see the performance of our HGS-Sched with respect to the two GA algorithms.

All 12 instances of ETC matrix were used in our experimental study. In fact, the objective was not only to study the performance of the HGS-Sched but also to see its behavior regarding three group of instances: consistent, semi-consistent and inconsistent. Conclusions in

Table 4: Comparison of applied replacement operators for makespan and flowtime values.

Operator	**Average Makespan**	**Average Flowtime**
Generational	7744052.84523	1047480700.341
Elitist Generational	**7696260.50065**	**10413560968.452**

this regard are interesting when the scheduler is to be designed for certain type of Grid infrastructures.

In order to make a fair comparison we considered that it is important to use not only the execution time, which is an arbitrary stopping condition, but also the amount of work done by the algorithm. The objective was thus to observe the performance of the algorithms, within the same amount of time, even though they use different genetic operators and parameter values.

We defined the work of GA in Eq. (10).

$$\begin{aligned} W = \quad & (mut\_prob + cross\_prob)\times \\ & \times pop\_size \times nb\_of\_generations \end{aligned} \tag{10}$$

For HGS-Sched parameter W is calculated as given in Eq. (11).

$$\begin{aligned} W = \quad & \Big[(mut\_prob(0) + cross\_prob) \times pop\_size(0) + \\ & + \textstyle\sum_{j=1}^{M} \big((mut\_prob(j) + cross\_prob)\times \\ & \times pop\_size(j) \times nb\_br(j)\big)\Big] \\ & \times period\_of\_metaepoch \times \\ & \times nb\_of\_metaepochs \end{aligned} \tag{11}$$

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. (10) and (11), the amount of work calculated for analyzed schedulers are as follows:

- for GA in (Braun et al., 2001): $W = (0.4 + 0.6) * 200 * 1000 = 200000$,
- for GA in (Xhafa et al., 2007b): $W = (0.4 + 0.8) * 68 * 2500 = 240000$, and
- for HGS-Sched: $W = W = ((0.2 + 0.8) * 8 * 12 + (0.4 + 0.8) * 28) * 200 * 10 = 259200$.

As can be seen from the above values, the three algorithms perform a similar amount of work, ensuring thus a fair comparison between them.

Comparison of makespan values. We present in Table 5 the makespan values obtained by the three genetic algorithms under study.

From the results in Table 5 we can observe that HGS-Sched outperforms the GA by Braun et al. for all but one instance and the GA by Xhafa et al. for 75% of considered instances. Notice that HGS-Sched performs better for consistent and semi-consistent ETC matrices. This observation is useful in considering HGS-Sched as a basis for Grid scheduler that have consistent computing features. It is also interesting to observe that HGS-Sched performed consistently and robustly, that is, it achieved high quality schedulers for the same type of examples in all different executions carried out for this experimental study.

Table 5: Comparison of best makespan values for three genetic algorithms. Best results are shown in bold

Instance	Braun et al.	Xhafa et al.	HGS-Sched
u_c_hihi	8050844.500	7610176.437	**7606401.235352**
u_c_hilo	156249.200	155251.196	**155063.78363**
u_c_lohi	258756.770	248466.775	**248059.043238**
u_c_lolo	5272.250	5226.972	**5215.044607**
u_i_hihi	3104762.500	**3077705.818**	3077989.170045
u_i_hilo	**75816.130**	75924.023	76122.941623
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	**127065.766276**
u_s_lolo	3583.440	**3515.526**	3575.692663

Table 6: Best flowtime values for benchmark instances (in arbitrary time units). Best results are shown in bold.

Instance	**Xhafa et al.** Steady State GA	**Xhafa et al.** Struggle GA	**HGS-Sched**
u_c_hihi	1048333229	**1039048563**	1040162825.157
u_c_hilo	27687019.4	27620519,	**27578766.61354**9
u_c_lohi	34767197.1	34566883,8	**34530865.9457**
u_c_lolo	920475.17	917647,31	**916896.056194**
u_i_hihi	378010732	379768078	**358850693.7164**
u_i_hilo	12775104.7	12674329,1	**12665322.029**
u_i_lohi	13444708.3	13417596,7	**12880192.58278**
u_i_lolo	446695.83	**440728,98**	441201.6681
u_s_hihi	526866515	524874694	**522084156.1341**
u_s_hilo	16598635.5	**16372763,2**	16419442.90678
u_s_lohi	15644101.3	15639622,5	**15370602.53526**
u_s_lolo	605375.38	**598332,69**	601220.872348

Comparison of flowtime values. We present in Table 6 the flowtime values obtained with the Steady State GA by Xhafa et al., Struggle GA presented in (Xhafa et al., 2008) and HGS-Sched algorithm (Braun et al. did not report values for flowtime parameter).

From the results in Table 6 we conclude that HGS-Sched outperforms Steady State GA for all types of ETC matrices and it is better than Struggle GA in 8 instances. In the case of flowtime, HGS-Sched performed consistently for all three groups of instances (consistent, semi-consistent and inconsistent ETC matrices).

Results of Summative Evaluation. The results of the experimental study showed that HGS-based scheduler performs better that GA-based schedulers reported for the problem in the literature for both makespan and flowtime minimization. Although the experimental study has been conducted on a small set of instances and of rather small sizes, HGS algorithms can be considered suitable for the design of Grid schedulers. Regarding the effect of Grid characteristics (features of ETC matrix) such as consistency of computing and heterogeneity, the most significant differences in the obtained results for HGS-Sched and GAs were observed for the instances of inconsistent ETC matrices, where the reduction of makespan by the hierarchic scheduler was not so effective.

CONCLUSIONS

In this work we have presented the implementation of Hierarchic Genetic Strategies (HGS) for Independent Job Scheduling on Computational Grids for which both makespan and flowtime parameters are simultaneously optimized. Achieving high quality planning of jobs into machines is crucial for Grid systems due to their heterogenous and dynamic nature. To this end, we have examined several variations of HGS operators in order to identify a configuration of operators and parameters that works best for the problem aiming at the design of efficient batch schedulers. The HGS implementation was experimentally studied using a known benchmark of static instances for the problem and the obtained results were compared with those of two other GA schedulers from the literature. From the experimental study we observed that HGS implementation outperforms existing classical GA implementations in the literature for most of considered instances of a static benchmark for the problem. In particular, HGS performed very well for instances representing features of consistent and semi-consistent computing environments.

In our future work we plan to study the performance of the HGS scheduler using the Grid simulator in Xhafa et al. (2007a). We would also like to design an interface for our HGS scheduler for its application to real Grid environments.

Acknowledgments

This research is partially supported by $10/I/GW/2008$ ATH Grant.

REFERENCES

S. Ali, H.J. Siegel, M. Maheswaran and D. Hensgen. Task execution time modeling for heterogeneous computing systems. Proceedings of Heterogeneous Computing Workshop, 185 - 199, 2000. (HCW 2000)

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. (2001). *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):810837.

Brucker, P. (2007). *Scheduling Algorithms, Fifth Ed..* Springer Vlg., Berlin-Heidelberg 2007.

Kołodziej, J., Gwizdała, R., and Wojtusiak, J.(2001). *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.

Kołodziej, J., Jakubiec, W., Starczak, M., and Schaefer R. (2004). *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.

Ritchie, G., and Levine J. (2003). *A fast effective local search for scheduling independent jobs in heterogeneous computing environments. Technical Report.* Centre for Intelligent Systems and Their Applications, School of Informatics, University of Edinburgh.

Schaefer, R. and Kołodziej J. (2002). *Genetic Search Reinforced by The Population Hierarchy. Foundations of Genetic Algorithms VII .* Morgan Kaufmann, 369–385.

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

Xhafa, F., Barolli, L.,and Durresi, A. (2008). *An Experimental Study On Genetic Algorithms for Resource Allocation On Grid Systems. Journal of Interconnection Networks.* vol. 8, issue 4, 427–443, World Sci. Pub.

Xhafa, F., Carretero, J.,and Abraham, A. (2007). *Genetic Algorithm Based Schedulers for Grid Computing Systems. International Journal of Innovative Computing, Information and Control.* vol. 3, number 5, 1–19.

AUTHOR BIOGRAPHIES

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 served as a PC member of ECMS 2007, CEC 2008, IACS 2008.

Dr. Fatos Xhafa received his PhD in Computer Science from the Technical University of Catalonia –UPC– (Barcelona, Spain) in 1998. He is currently Associate Professor at UPC. He served as Organizing Chair of ARES 2008, PC chair of CISIS 2008, General co-chair of HIS 2008 and Workshops chair of CISIS 2009 conferences. Presently, he is General co-chair of INCoS-2009 international conference.

Łukasz Kolanko is an undergraduate student of computer science at the Faculty of Mechanical Engineering and Computer Science at the University of Bielsko-Biała. The topic of his diploma thesis is the analysis of the hierarchic genetic algorithms in distributed heterogeneous environments.