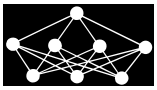

Tuning Struggle Strategy in Genetic Algorithms for Scheduling in Computational Grids

Fatos Xhafa[*], Bernat Duran[*], Ajith Abraham[†], Keshav Dahal[‡]

Abstract:

Job Scheduling in Computational Grids is gaining importance due to the need for efficient large-scale Grid-enabled applications. Among different optimization techniques addressed for the problem, Genetic Algorithm (GA) is a popular class of solution methods. As GAs are high level algorithms, specific algorithms can be designed by choosing the genetic operators as well as the evolutionary strategies such as Steady State GAs and Struggle GAs. In this paper we focus on Struggle GAs and their tuning for scheduling of independent jobs in computational grids. Our results showed that a careful hash implementation for computing the similarity of solutions was able to alleviate the computational burden of Struggle GA and perform better than standard similarity measures. This is particularly interesting for the scheduling problem in Grid systems, which due to changeability over time, has demanding time restrictions on the computation of the planning of jobs to resources.

Key words: *Genetic Algorithms, Scheduling, Grid Computing, Struggle Strategy, Similarity Measure, Tuning.*

Received: ??
Revised and accepted: ??

1. Introduction

With the emerging paradigm of Grid Computing and the development of Grid infrastructures, Grid-based applications are becoming a common approach for solving many complex problems. A key issue in this kind of applications is scheduling jobs into Grid resources efficiently, which is known to be computationally hard and much more difficult than its standard version for sequential or LAN computation environments.

[*]Department of Languages and Informatics Systems, Technical University of Catalonia, Campus Nord, Ed. Omega, C/Jordi Girona 1-3, 08034 Barcelona, Spain. E-mail: fatos@lsi.upc.edu, bduran@lsi.upc.edu

[†]Center of Excellence for Quantifiable Quality of Service, Norwegian University of Science and Technology, Trondheim, Norway ajith.abraham@ieee.org

[‡]School of Informatics, University of Bradford, Bradford BD7 1DP, UK k.p.dahal@Bradford.ac.uk

Job Scheduling in Computational Grids is gaining importance due to the need for efficient large-scale Grid-enabled applications, e.g. in Optimization (Casanova *et al.* [8], Goux *et al.* [13] and Wright [30]), Linderoth *et al.* [19]), Collaborative/eScience Computing (e.g. Newman *et al.* [22], Paniagua *et al.* [24]), Data-Intensive Computing (e.g. Beynon al. [3]) and many applications arising from concrete types of Grids such as Science Grids, Access Grids, Knowledge Grids, etc. Scheduling is a challenging problem in a Grid environment due its dynamic nature and the large number of resources to be managed and jobs to be scheduled. Furthermore, resources can have their own local policies (regarding access, cost etc.) to be taken into account. The problem is multi-objective in its general definition, as there are several optimization criteria to be matched, such as makespan, flowtime, and resource utilization.

Several approaches are being addressed in the literature for the problem aiming to obtain schedulers capable of delivering fast planning of jobs to computational resources of the grid system. On the one hand there many *ad hoc* methods such as immediate and batch mode methods [37, 36]. Such methods distinguish for their simplicity and efficiency. However, these methods fail to produce high quality planning of jobs to Grid resources; for instance the immediate method of Opportunistic Load balancing assigns a job to the machine have the smallest workload, which in a very heterogenous Grid environment could perform poorly. Moreover, such methods can handle only one objective at a time (usually the makespan, workload, etc.) and in Grid systems usually there are more requirements on scheduling. Given the large scale of the Grid systems as well as periodic submissions of large quantity of jobs, researchers are seeking for ways to design more efficient Grid schedulers. In particular, Genetic Algorithms (GA) [16] have proved to be a good alternative for solving a wide variety of hard combinatorial optimization problems and are therefore appropriate for job scheduling in Grids. GAs are a population-based approach where individuals represent possible solutions, which are successively evaluated, selected, crossed, mutated and replaced by simulating the Darwinian evolution found in nature. Genetic Algorithms for Grid scheduling problems have been addressed by Abraham *et al.* [1], Braun *et al.* [5], Zomaya and Teh [41], Martino and Mililotti [9], Page and Naughton [23], Carretero and Xhafa [7], Gao *et al.* [12], Xhafa *et al.* [35, 34].

The research work on GAs has shown that a key issue in GAs is the convergence of the algorithm: a fast convergence of the population would stagnate the search to local optima whereas slower convergence would require a considerably longer time towards sub-optimal solutions. The convergence of GAs is achieved by means of selection and replacement strategies and it is, therefore, very important to carefully tune these strategies. In particular, the selective pressure directly affects the tradeoff between the exploration and exploitation of the search space. Indeed, if the population converges rapidly GA would give more priority to the exploitation and, vice-versa, when the population is kept diverse, other regions of the search space would be explored aspiring thus to find better solutions. GAs represent thus an interesting family of algorithms for Grid scheduling since in many practical Grid-enabled applications we are interested to compute a reasonably good planning of jobs in a very short time rather than an optimal planning. In such case, GAs are useful since we can "burst up" the convergence of the algorithm. Yet, we

are interested to avoid a very premature convergence of the algorithm.

In this work we focus on the importance of tuning the replacement mechanism of GA for scheduling in computational grids. The interest in investigating this aspect is motivated by the need to design efficient schedulers that will be able to deliver fast and quality planning of jobs to resources rather optimal solutions in a dynamic environment. More precisely, we study the tuning of the Struggle strategy (Grueninger [14]; see also [28]). According to this strategy, a new individual replaces the individual that is most similar to it only in case the new individual obtains a better fitness value than the one to be replaced. The aim is to preserve the optimization velocity but delaying its tendency to converge in order to reach a better convergence point. This strategy is known for its effectiveness but suffers from a high computational cost. More precisely, given a new individual, finding a similar individual to it requires comparing against all individuals of the current generation. Efficient computation of the similarity would therefore alleviate the computational burden of the Struggle GA.

The rest of the paper is organized as follows. Some related work to the scheduling problem as well as GA-based work that, as in the case of Struggle GA, use similarity measures to maintain the diversity of the population during the evolution process are given in Section 2. The problem of scheduling of independent jobs considered in this work is presented in Section 3. The Struggle strategy together with similarity measures are introduced in Section 4. The experimental study and some computational results are given in Section 5. We conclude in this work with some remarks and indications for future work in Section 6.

2. Related work

In this section we briefly review some related work in computational intelligence techniques applied to the scheduling problem. Also, other GA-based work that, as in the case of Struggle GA use similarity measures to maintain the diversity of the population during the evolution process, are also indicated.

Genetic Algorithms for Grid scheduling problems have been addressed by Abraham et al. [1], Braun et al. [5], Zomaya and Teh [41], Martino and Mililotti [9], Page and Naughton [23], Carretero and Xhafa [7], Gao et al. [12], Xhafa et al. [35, 34].

Finding a good trade-off between the exploration and exploitation, which is closely related to the diversity of population, has been explored in the GA literature [4, 6, 10, 38]. An interesting recent approach to the tradeoff between exploration and exploitation in evolutionary algorithms is based on the entropy concept [20].

Multi-objective GAs (such as NSGA, SPEA) use some similarity measures to maintain the diversity of the population during the evolution process. Thus, Sato et al. [26], proposed a method for NSGA II (Non-dominated Sorting Genetic Algorithm II) in which similar individuals are eliminated in the process of evolution by using the distance between individuals in objective space. Ishibuchi and Narukawa [17] examined the relation between the performance of the NSGA-II algorithm and the similarity of recombined parent solutions for flowshop scheduling problems. The authors examined the effect of increasing the selection pressure on

the similarity of recombined parent solutions. Wildman and Parks [29] presented a comparative study of selective strategies in Multi-objective GAs through different pairing strategies for combining parents. Ishibuchi and Shibata [18] proposed a new mating scheme in which similarity-based tournament selection is used for choosing a pair of parents among the candidate solutions aiming to maintain the diversity of solutions.

Recently, Memetic Algorithms (MAs) [21] –a relatively new class of population-based methods– which combine the concepts of evolutionary search and local search have been proposed for Grid scheduling problem. Xhafa [32] applied unstructured MAs and Xhafa et al. [33] proposed Cellular MAs (structured MAs) for the independent scheduling problem under ETC model.

Other approaches include Reinforced Learning, Neural Networks, Fuzzy Logic, etc. Some authors have used reinforced learning techniques for scheduling in Grid systems. Perez *et al.* [25], proposed to implement a Reinforcement Learning based scheduling approach for large Grid computing systems. Vengerov [27] presented a utility-based framework for making repeated scheduling decisions dynamically; the observed information about unscheduled jobs and system's resources is used for this purpose. Yu et al. [39] used Fuzzy Neural Networks to develop a high performance scheduling algorithm. The algorithms uses Fuzzy Logic techniques to evaluate the Grid system load information, and adopt the Neural Networks to automatically tune the membership functions. Hao *et al.* [15] presented a Grid resource selection based on Neural Networks aiming at offering QoS on distributed, heterogeneous resources. To this end, the authors propose to select Grid resources constrained by QoS criteria. The resource selection problem is solved using a novel neural networks. Zhou et al. [40] used Fuzzy Logic techniques to design an adaptive Fuzzy Logic scheduler, which utilizes the Fuzzy Logic control technology to select the most suitable computing node in the Grid environment.

3. Problem definition

The job scheduling problem in Grids has many characteristics in common with the traditional scheduling problems. The objective is to efficiently map jobs to resources; however, in a global, heterogenous and dynamic environment, such as Grid environment, we are interested to find a *practically* good planning of jobs very fast.

In this work we deal with the scheduling independent jobs to resources. We describe this version next and then give a formal definition of an instance of the problem. Jobs have the following characteristics: are originated from different users/applications, have to be completed in unique resource (*non-preemptive*), are independent and could also have their requirements over resources. This last characteristic is important if we would like to classify jobs originated from data intensive or computing intensive applications. On the other hand, resources could dynamically be added/dropped from the Grid, can process one job at a time and have their computing characteristics.

This version arises in many Grid-based applications, such as in simulations, massive data processing, which can be divided into independent parts, which are mapped to different Grid resources.

Expected Time to Compute simulation model In order to formalize the instance definition of the problem, we use the ETC (Expected Time To Compute) matrix model, see e.g. [5]. This model is used for capturing most important characteristics of job and resources in distributed heterogeneous environments. In a certain sense, a good planning jobs to resources will have to take into account the characteristics of jobs and resources. More precisely, the Expected Time to Compute matrix, ETC, has size $nb\_jobs \times nb\_machines$ and its components $ETC[i][j]$ are defined as the expected execution time of job i in machine j. ETC matrices are then classified into consistent, inconsistent and semi-consistent according to the consistency of computing of resources: (a) *consistency* means that if a machine m_i executes a job faster than machine m_j, then m_i executes all the jobs faster than m_j. If this holds for all machines participating in the planning, the ETC matrix is considered consistent ; (b) *inconsistency* means that a machine is faster for some jobs and slower for some others; and, (c) *semi-consistency* is used to express the fact that an ETC matrix can have a consistent sub-matrix. In this case the ETC matrix is considered semi-consistent. Notice that the variability in characteristics of jobs and resources yields to different ETC configurations allowing thus to simulate different scenarios from real life distributed applications.

Problem definition Under the ETC simulation model, an instance of the problem consists of:

- A *number* of independent (user/application) *jobs* to be scheduled.
- A *number* of heterogeneous *machines* candidates to participate in the planning.
- The *workload* of each job (expressed in millions of instructions).
- The *computing capacity* of each machine (expressed in *mips* –millions of instructions per second).
- Ready time $ready[m]$ –when machine m will have finished the previously assigned jobs.
- The Expected Time to Compute matrix, ETC.

Optimization criteria. Several objective criteria can be established for a given schedule. We consider the minimization of makespan, that is, finishing time of latest job (S denotes a possible schedule):

$$\min_{S} \max\{F_j : j \in Jobs\}.$$

where F_j is the finishing time of job j.

Makespan can be expressed in terms of the completion time of a machine, as follows:

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

where for a machine m:

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

4. Struggle strategy in GAs and similarity measures

In Struggle GAs [14, 28] (hereafter, SGA), a new generation of individuals is created by replacing only a portion of the population with the new individuals. The struggle genetic algorithm works similarly as the steady-state GAs. However, istead of replacing the worst individual, in SGA a new individual replaces the individual that is most similar to it only in case the new individual obtains a better fitness value than the one to be replaced. This is done in order to adaptively maintain certain diversity among the population. The aim is to preserve the optimization velocity but delaying its tendency to converge in order to reach a better convergence point.

The design of the struggle replacement operator requires the definition of appropriate similarity measures. A similarity measure indicates how similar are two individuals (solutions of the problem). The definition of a similarity measure could be done in different ways, for instance by using the structure of the solution (combinatorics properties)

It has been shown in GA literature that in the long run Struggle GA and Steady State GA converge to a single solution. The interest in using Struggle GA, as opposed to Steady State-like GAs is that in Struggle GAs population evolves by maintaining different solutions long after a basic or steady-state algorithm would have converged. This is a desired property for the case of the scheduling problem in Computational Grids given that we can fine tune the scheduler to "converge" to a good solution depending on available time (for instance, scheduler's time activation interval). Further, Struggle GAs are simple to implement and require no additional parameters to fine tune but the struggle operator. It should be noted however that the performance of Struggle GA, despite of good diversity of the population, depends also on the rest of genetic operators; thus, cross-over operators feeding the population with good individuals and low mutation rate would yield a very good performance of the Struggle GA.

4.1 Computational complexity of struggle operator

This strategy has shown to be very effective for several problems [14, 2]; yet, there is an efficiency issue here: the computational cost of this replacement strategy is very high. Indeed, in order to find which individuals should leave the population, any new individual of the intermediate population has to be compared and its similarity measured against all the individuals of the current population. Obviously, this leads to a quadratic order computational time, which could be very large, if large size populations were to be considered. In fact, this is precisely the case of scheduling independent jobs in computational grids; their large scale and scalability are critical factors since not only the number of resources and jobs submitted to the Grid system are expected to be large or very large but also they could increase over time. It is clear that similarity measures which are not efficient could consume much of the GA running time in detriment to the proper search time.

4.2 Standard similarity measures

In order to compare the similarity between solutions, a measure of similarity or distance function has to be defined and used. Standard similarity measures include:

- *Hamming distance*: given two individuals S_1 and S_2 encoding two schedulings of N jobs, let $g[i] = 1$, iff $S_1[i] = S_2[i]$ and $g[0] = 0$, otherwise. Similarity is then calculated as:

$$Sim_h(S_1, S_2) = \frac{\sum_{i=1}^{N} g[i]}{N}.$$

- *Euclidian distance*: This similarity is based on the Euclidean distance. Given two vector solutions S_1 and S_2, by considering them as two points in N-dimensional space, the similarity is then computed as the Euclidean distance between them:

$$Sim_e(S_1, S_2) = \sqrt{\sum_{i=1}^{N} (S_1[i] - S_2[i])^2}.$$

- *Cosine distance*: In this case, the similarity is measured using the angle of the two vector solutions S_1 and S_2 of the N-dimensional space. Cosine values close to 1 would mean more similarity.

$$Sim_c(S_1, S_2) = \frac{\sum_{i=1}^{N} S_1[i] \cdot S_2[i]}{\sqrt{\sum_{i=1}^{N} S_1^2[i]} \cdot \sqrt{\sum_{i=1}^{N} S_2^2[i]}}.$$

4.3 Hash-based similarity measure

The standard similarity measures given above has linear time computational cost in number of jobs. Therefore for a population of *pop_size* the standard struggle strategies would take $O(intermediate\_pop\_size \times pop\_size) \times N$, where N is the number of jobs. Reducing the quadratic factor of $O(intermediate\_pop\_size \times pop\_size)$ to a linear time factor would be very desirable in this case since in each replacement step it would take a considerable time in detriment to the proper search time of the GA. In order to achieve this, we propose the use of hash techniques so that given a new individual of the intermediate population we can find in constant time the individual most similar to it.

In order to design the hash table, we have to first define the key to identify the individuals of the population. The *key* information is the basis for computing the degree of similarity of the *struggle genetic operator*: the more accurate its definition the better the performance of the operator. In fact, a poor definition of the key would simply reduce the struggle operator to a random replacement. In our definition of the key the context is crucial: the key value should resume as much as possible the genetic information encoded in an individual; hence, if two key values are similar then their respective individuals are genetically similar. The following are three possible definitions:

a) *Fitness-based key*: consists in using the fitness value, which is transformed, using a hash function, into the key value. Certainly this is a very simplistic approach by simply looking at makespan and flowtime values and clearly no genetic information is taken into account (we refer to this as `'a' key`).

b) *Position-based key*: having the permutation vector of task-resource allocation, in which tasks are sorted according to the resource they are assigned to, the key is defined as the sum of number of cells a component of the vector would move to the right as indicated by its value, when the vector is read in a circular way (we refer to this as `'b' key`). For instance, for the vector of 7 tasks in Fig. 1 below, $key = 2 + 4 + 1 + 0 + 2 + 5 + 0 = 14$.

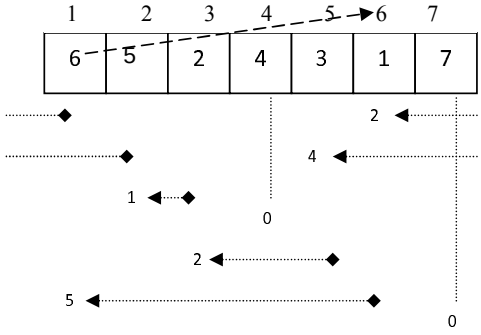

Fig. 1 *Example position-based key calculation.*

Note that this definition uses the genetic characteristics of the solution; however, the relation task-resource is not explicitly taken into account, i.e., to which resource is assigned a task.

c) *Task-resource allocation key*: In this case both information on tasks and resources is used. The key value is now the sum of the absolute values of the subtraction of each position and its precedent in the vector of task-resource allocation (reading the vector in a circular way); we refer to this as `'c' key`.

We give in Fig. 2 the graphical representation of the hash table design as well as the formulae definition of the hash function. Note that the corresponding position is obtained from a solution from the key value k; k_{min} and k_{max} correspond respectively to the key with smallest and largest value in the population.

The hash table has the same size as the population in order to obtain constant time access (in average). If an access fails, a few individuals in the population are randomly chosen and the most similar to the new one is considered for the replacement. Hence, the constant access is always ensured even if a failed access occurs. Therefore, the computational cost of the hash-based struggle operator is $O(pop\_size + intermediate\_pop\_size)$.

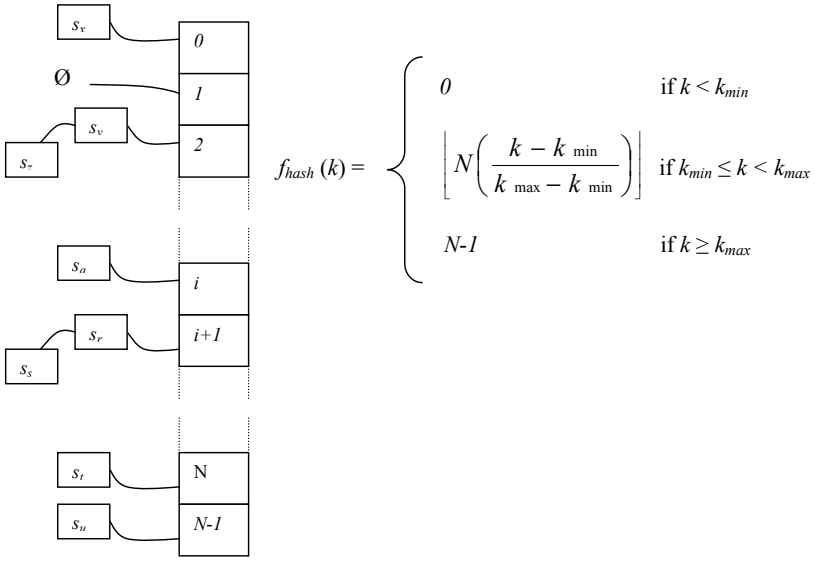

Fig. 2 *Representation of the hash table and the hash function definition.*

5. Experimental study

In this section we present the experimental study of the proposed hash-based Struggle GA. Initially, we generated a set of instances according to ETC matrix model in order to study the performance of the three key definitions and also to fine tune the rest of the parameters of Struggle GA. The best resulting configuration was then used for studying the performance of the SGA on a set of known instances from Braun *et al.* [5].

5.1 Performance comparison of struggle hash operators

The performance of the three struggle operators resulting from `a key`, `b key` and `c key` definitions were measured for makespan value of the schedule. For each of them, the same configuration of parameters (see Table I) was used; reported values are averaged over 10 independents runs.

In Table I, MCT (Minimum Completion time) and LJFR-SJFR (Longest Job to Fastest Resource - Shortest Job to Fastest Resource) are two methods used in initializing the population; rebalance-both is a mutation operator based on load balancing of resources. MCT method [11] assigns a job to the machine yielding the earliest completion time (the ready times of the machines are used). When a job arrives in the system, all available resources are examined to determine the resource that yields the smallest completion time for the job. On the other hand, LJFR-SJFR [1] tries to simultaneously minimize both makespan and flowtime values: LJFR (Longest Job to Fastest Resource) tries to minimize makespan and SJFR (Shortest Job to Fastest Resource) tries to minimize flowtime.

We show in Fig 3, the makespan value computed by the SGA algorithm with

Tab. I *Parameter configuration of Struggle GA*

nb_evolution_steps	(max 90s)
pop_size	10
intermediate_pop_size	6 (60%)
cross_probability	0.9
mutate_probability	0.2
start_choice	MCT and LJFR-SJFR
select_choice	N tournament
cross_choice	Fitness Based Crossover
Mutate_choice	rebalance-both

Hamming distance measure (denoted SGA struggle quadratic) of similarity against the SGA + `a key`, `b key` and `c key` implementations.

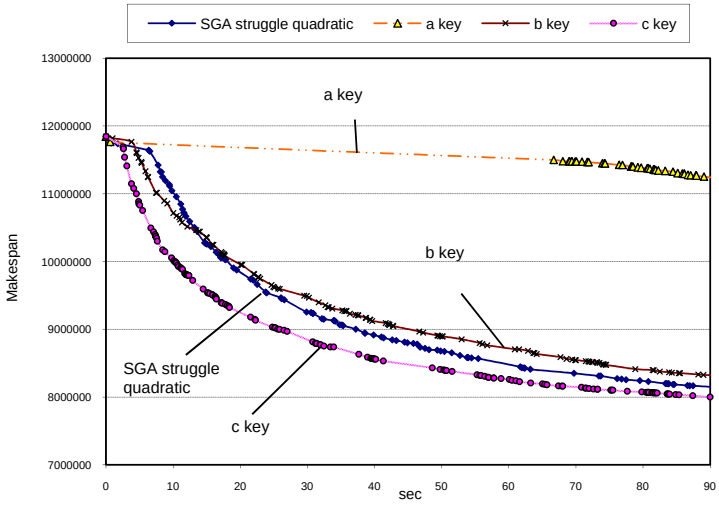

Fig. 3 *Comparison of makespan values (in arbitrary time units) obtained with four versions of SGA.*

As can be seen from Fig 3, the `a key` implementation of SGA performs very poorly -it doesn't even mimic the behavior of SGA struggle quadratic- recall that the measure of similarity related to the `a key` is just the fitness. On the other hand, `b key` and `c key` behave coherently. As expected, the `c key` outperforms both the SGA struggle quadratic and the `b key` implementation. It's worth observing that the `c key` implementation achieves a faster reduction in makespan value –which is certainly desirable for Grid schedulers running in short periods of time in case when we just need a feasible (reasonably good) solution rather than an optimal solution.

In order to better understand the behavior of the three hash-based SGA implementations, we monitored the similarity value computed by each of them (see Table II). It can be seen that, except the `a key` case, the rest used the same simi-

Tab. II *Similarity value of the studied Struggle GAs.*

SGA version	Similarity value (averaged)
SGA struggle quadratic	0,975
`a key` SGA	0,736
`b key` SGA	0,958
`c key` SGA	0,965

larity value. The best performance of `c key` implementation could be explained on the one hand due to its time efficiency (especially when compared with SGA struggle quadratic) and due to the fact that it achieves to introduce a new individual of better genetic information in the population.

Another benefit of using the hash-based struggle implementation is the scalability. Indeed, since the computational time now is scaled down to a linear order, we could increase the size of population and that of intermediate population without affecting the search time of the GA. We observed this experimentally (hereafter, the `c key` SGA is used); we show this effect by considering a population of 30 individuals and intermediate population of 50% of population size and then by increasing the population size to 80 individuals. The graphical representation is shown in Figs. 4 and 5.

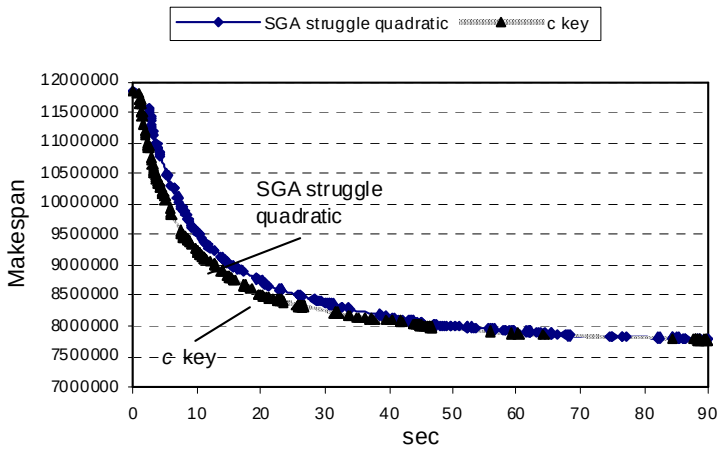

Fig. 4 *Makespan reduction of* `c key` *SGA vs. SGA struggle quadratic using a population of 30 individuals and intermediate population size of 50%.*

Effectively, we can see from Figs. 4 and 5 that, while for a population of size 30 the two SGA perform almost equally, by enlarging the population size to 80 individuals, the makespan reduction is much faster for the `c key` SGA than SGA struggle quadratic.

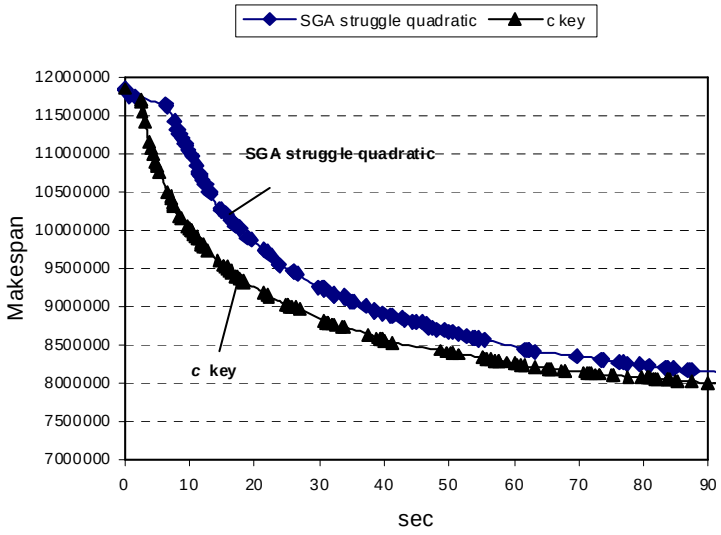

Fig. 5 *Makespan reduction of* `c key` *SGA vs. SGA struggle quadratic using a population of 80 individuals and intermediate population size of 50%.*

5.2 Computational results for a static benchmark

We present here some computational results for the `c key` SGA implementation for 12 instances from the static benchmark or Braun *et al.* [5].

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

- u means uniform distribution (used in generating the matrix);
- x means the type of consistency (c–consistent, i–inconsistent and s means semi-consistent). An ETC matrix is considered consistent when, if a machine m_i executes job t faster than machine m_j, then m_i executes all the jobs faster than m_j. Inconsistency means that a machine is faster for some jobs and slower for some others. An ETC matrix is considered semi-consistent if it contains a consistent sub-matrix;
- yy indicates the heterogeneity of the jobs (hi means high, and lo means low); and,
- zz indicates the heterogeneity of the resources (hi means high, and lo means low).

Each instance consists of 512 jobs and 16 machines.

The parameter configuration used is that of Table I. The results are averaged over 10 independents runs, each of them running for at most 90 sec. on the same machine of standard configuration.

Tab. III *Makespan values for Braun et al. instances obtained with SGA (struggle quadratic) and SGA with* ***c key*** *hash implementation.*

Instance	SGA (struggle quadratic)	SGA (c key Hash)
u_c_hihi.0	**7722057,537**	7752689,084
u_c_hilo.0	157009,284	**156680,578**
u_c_lohi.0	**253843**	253926,055
u_c_lolo.0	5271,765	**5251,1462**
u_i_hihi.0	3269481,011	**3161104,915**
u_i_hilo.0	76413,844	**75598,481**
u_i_lohi.0	116981,385	**111792,174**
u_i_lolo.0	2634,997	**2620,7218**
u_s_hihi.0	4473513,333	**4433792,275**
u_s_hilo.0	98782,703	**98560,043**
u_s_lohi.0	132971,473	**130425,852**
u_s_lolo.0	3565,905	**3534,306**

The computational results are shown in Table III. In the table, the first column indicates the instance name. Note that considered instances are representing different heterogeneous computing environments (four instances for consistent, semi-consistent and inconsistent, respectively). The second column indicates the makespan value obtained with the SGA (struggle quadratic) and the third one the makespan value obtained with SGA (`c key` hash implementation). Values in bold face show the best makespan of the two implementations.

As can be seen from Table III, SGA with `c key` hash implementation outperforms SGA struggle quadratic implementation. Moreover, the SGA with `c key` hash implementation shows to be robust, it performs well for three types of instances (consistent, inconsistent and semi-consistent computing environments).

5.3 Discussion

From the experimental results presented in the previous subsections, it is clearly observed that tuning hash-based similarity measures yielded to an efficient struggle genetic operator. In particular, it is remarkable from Figs. 4 and 5 that the approach scales very well when the population size is increased. Indeed, when passing from a population of 30 individual to 80 individuals, the hash-based GA clearly outperforms the basic GA. This behavior is interesting for Grid scheduling in which the fluctuation in the instance size has to be taken into account; in particular, the number of jobs could unexpectedly increase and thus the size of the population should respectively considered of larger size.

It should also be noticed that although we have presented the study for GAs, the similarity measures considered in this work are also applicable for other population-based scheduling algorithms, so it would be interesting to extend this work for other classes of population-based algorithms.

6. Conclusion and future work

In this paper, we have presented some results on tuning Struggle GAs for the problem of scheduling independent jobs in computational grids. This version of the scheduling requires fast reduction of makespan due to the changeability of the computational environment. In this work we have proposed hash-based implementations of the struggle genetic operator for the GAs for the problem. The resulting struggle operator showed several good properties:

- *It is efficient*: the quadratic computational time (in terms of the population time) is reduced to a linear time and hence the overall proper search time of the GA is increased.
- *It is scalable*: large scale scheduling problems can be efficiently tackled by considering larger size populations, due to the linear factor computational time.
- *It is effective*: new individuals of better genetic information are introduced into new generation.
- *It is robust*: the resulting Struggle GA performed very well on almost all considered instances representing different heterogeneous computing environments.

In our future work we plan to consider other similarity measures based on binary representations of task-resource allocation. Also, we would like to consider clustering-like techniques, for instance, *K-clustering*, to increase the performance of the struggle operator. Finally, we have considered only the makespan objective. It would be interesting to consider the more general multi-objective case of using similarity measures in spirit of those presented in this study for NSGGII, SPEA and other multi-objective approaches for the scheduling problem.

Acknowledgements

We are grateful to the reviewers for their useful comments that helped us to improve the paper. This research is partially supported by Projects ASCE TIN2005-09198-C02-02, FP6-2004-ISO-FETPI (AEOLUS) and MEC TIN2005-25859-E and FORMALISM TIN2007-66523.

References

[1] Abraham, A., Buyya, R., and Nath, B. Nature's heuristics for scheduling jobs on Computational Grids. In *The 8th IEEE International Conference on Advanced Computing and Communications, India*, 2000.

[2] Bartschi Wall, M. A Genetic Algorithm for Resource-Constrained Scheduling PhD Thesis, Massachusetts Institute of Technology June 1996

[3] Beynon, M.D., Sussman, A., Catalyurek, U., Kure, T. and Saltz, J. Optimization for data intensive grid applications. In *Third Annual International Workshop on Active Middleware Services*, 97–106, California, 2001.

[4] Brameier, M. and Banzhaf, W. Explicit control of diversity and effective variation distance in linear genetic programming, In *Genetic Programming, 5th European Conference, EuroGP 2002, Kinsale, Ireland, April 3-5, 2002*, vol. 2278 of *Lecture Notes in Computer Science*, 37–49. Springer, 2002.

[5] Braun, T., Siegel, H., Beck, N., Boloni, L., Maheswaran, M., Reuther, A., Robertson, J., Theys, M., and Yao, B. 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.

[6] Burke, E.K., Gustafson, S.M., Kendall, G. and Krasnogor, N. Advanced population diversity measures in genetic programming, In *Parallel Problem Solving from Nature - PPSN VII, 7th International Conference, Granada, Spain*, vol. 2439 of *Lecture Notes in Computer Science*, 341–350. Springer, 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, ISSN 1392-8619, Vol. 12, No. 1, p. 11-17, 2006.

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

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

[10] Ekárt, A. and Németh, S.Z. Maintaining the diversity of genetic programs, In *Genetic Programming, 5th European Conference, EuroGP 2002, Kinsale, Ireland*, vol. 2278 of *Lecture Notes in Computer Science*, 162–171. Springer, 2002.

[11] Freund, R., Gherrity, M., Ambrosius, S., Campbell, M., Halderman, M., Hensgen, D., Keith, E., Kidd, T., Kussow, M., Lima, J., Mirabile, F., Moore, L., Rust, B. and Siegel, H. Scheduling resources in multi-user, heterogeneous, computing environments with SmartNet. In *Seventh Heterogeneous Computing Workshop*, 184–199, 1998.

[12] Gao, Y., Rong, H., and Huang, J. Z. Adaptive Grid job scheduling with genetic algorithms. *Future Gener. Comput. Syst.* 21, 1 (Jan. 2005), 151-161, 2005.

[13] Goux, J.P., Kulkarni, S., Linderoth, J. and Yoder, M. An enabling framework for master-worker applications on the computational grid. In *9th IEEE Int. Symposium on High Performance Distributed Computing (HPDC'00)*, 2000.

[14] Grueninger, T.: Multimodal optimization using genetic algorithms. Technical report. Department of Mechanical Engineering, MIT, Cambridge, MA, 1997.

[15] Hao, X., Dai, Y., Zhang, B., Chen, T. and Lei Yang: QoS-Driven Grid Resource Selection Based on Novel Neural Networks. GPC 2006: 456-465, 2006.

[16] Holland, J.H. Adaptation in Natural and Artificial Systems, University of Michigan Press, Ann Arbor, MI, 1975.

[17] Ishibuchi, H. and Narukawa, K. Recombination of Similar Parents in EMO Algorithms, In Evolutionary Multi-Criterion Optimization, 265-279, 2005.

[18] Ishibuchi, H. and Shibata, Y. A Similarity-Based Mating Scheme for Evolutionary Multiobjective Optimization. Lecture Notes in Computer Science, Springer, Volume 2723/2003, Genetic and Evolutionary Computation GECCO 2003.

[19] Linderoth, L. and Wright, S. Decomposition algorithms for stochastic programming on a Computational Grid. *Computational Optimization and Applications*, 24:207–250, 2003.

[20] Liu, Sh., Mernik, M. and Bryant, B.R. Entropy-driven exploration and exploitation in evolutionary algorithms, In Bogdan Filipič and Jurij Šilc, editors, *Proceedings of the 2nd International Conference on Bioinspired Optimization Methods and their Applications (BIOMA 2006)*, 15–24, Ljubljana, Slovenia, October 2006. Jožef Stefan Institute.

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

[22] Newman, H.B., Ellisman, M.H. and Orcutt, J.A. Data-intensive e-Science frontier research. *Communications of ACM*, 46(11):68–77, 2003.

[23] Page, J. and Naughton, J. Framework for task scheduling in heterogeneous distributed computing using genetic algorithms. *AI Review*, 24:415-429, 2005.

[24] Paniagua, C., Xhafa, F., Caballé, S. and Daradoumis, T. A parallel grid-based implementation for real time processing of event log data in collaborative applications. In *Parallel and Distributed Processing Techniques (PDPT2005)*, 1177–1183, Las Vegas, USA, 2005.

[25] Perez, J., Kégl, B. and Germain-Renaud, C. Reinforcement learning for utility-based Grid scheduling. At NIPS07 (Twenty-First Annual Conference on Neural Information Processing Systems) Workshops, in Vancouver, Canada, 2007.

[26] Sato, M., Aguirre, H.E. and Tanaka, K. Effects of -Similar Elimination and Controlled Elitism in the NSGA-II Multiobjective Evolutionary Algorithm. In IEEE Congress on Evolutionary Computation, 1164-1171, Vancouver, BC, Canada, 2006

[27] Vengerov, D. Adaptive Utility-Based Scheduling in Resource-Constrained Systems. In AI 2005: Advances in Artificial Intelligence, pp. 477-488, Springer Verlag, 2005

[28] Nicola Senin, Roberto Groppetti and David R. Wallace. Concurrent assembly planning with genetic algorithms. *Robotics and Computer-Integrated Manufacturing*, Vol. 16, Issue 1, pp. 65-72, 2000

[29] Wildman, A. and Parks, G. A Comparative Study of Selective Breeding Strategies in a Multiobjective Genetic Algorithm. In C.M. Fonseca et al. (Eds.): EMO 2003, LNCS 2632, pp. 418432, 2003.

[30] Wright, S. (2001). Solving optimization problems on Computational Grids. *Optima*, Vol. 65, 2001.

[31] Xhafa, F. A Hyper-heuristic for Adaptive Scheduling in Computational Grids, International Journal on Neural and Mass-Parallel Computing and Information Systems, 17(6), 639-656, 2007

[32] Xhafa, F. A Hybrid Evolutionary Heuristic for Job Scheduling in Computational Grids. Springer Verlag Series: Studies in Computational Intelligence , Vol. 75 2007, Chapter 10, ISBN: 978-3-540-73296-9. September 2007.

[33] Xhafa, F., Alba, E., Dorronsoro, B. and Duran, B. Efficient Batch Job Scheduling in Grids using Cellular Memetic Algorithms, Accepted, *Journal of Mathematical Modelling and Algorithms*, Published Online DOI: http://dx.doi.org/10.1007/s10852-008-9076-y

[34] Xhafa, F., Barolli, L. and Durresi, A. An Experimental Study On Genetic Algorithms for Resource Allocation On Grid Systems, *Journal of Interconnection Networks*, Volume: 8, Issue: 4 (December 2007), 427 - 443, World Sci. Pub.

[35] Xhafa, F. Carretero, J. and Abraham, A. Genetic Algorithm Based Schedulers for Grid Computing Systems. International Journal of Innovative Computing, Information and Control, Vol. 3, No.5, pp. 1-19, 2007.

[36] F. Xhafa, L. Barolli and A. Durresi. Batch Mode Schedulers for Grid Systems. International Journal of Web and Grid Services, Vol. 3, No. 1, 19-37, 2007.

[37] F. Xhafa, J. Carretero, L. Barolli and A. Durresi. Immediate Mode Scheduling in Grid Systems. International Journal of Web and Grid Services, Vol.3 No.2, 219-236, 2007.

[38] Yan, W. and Clack, Ch. D. Behavioural GP diversity for dynamic environments: an application in hedge fund investment, In *GECCO '06: Proceedings of the 8th annual conference on Genetic and evolutionary computation*, 1817–1824, New York, NY, USA, 2006. ACM Press.

[39] Yu, K.M., Luo, Zh.J., Chou, Ch.H., Chen, Ch.K., and Zhou, J. A Fuzzy Neural Network Based Scheduling Algorithm for Job Assignment on Computational Grids. NBiS 2007: 533-542, Lecture Notes in Computer Science, Springer Verlag, 2007

[40] Zhou, J., Kun-Ming Yu, K-M. Chou, Ch-H., Yang, L-A., and Luo, Zh-J.: A Dynamic Resource Broker and Fuzzy Logic Based Scheduling Algorithm in Grid Environment. ICANNGA (1) 2007: 604-613, 2007.

[41] 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.