ISSN 1392-8619 print/ISSN 1822-3613 online
ÛKIO TECHNOLOGINIS IR EKONOMINIS VYSTYMAS
TECHNOLOGICAL AND ECONOMIC DEVELOPMENT OF ECONOMY
http://www.tede.vgtu.lt
2006, Vol XII, No 1, 11–17

USE OF GENETIC ALGORITHMS FOR SCHEDULING JOBS IN LARGE SCALE GRID APPLICATIONS

Javier Carretero, Fatos Xhafa

Department of LSI, UPC, Polytechnic University of Catalonia,
Campus Nord - Ed. Omega, Jordi Girona Salgado 1-3, 08034 Barcelona, Spain
E-mail: fatos@lsi.upc.edu

Received 3 October 2005; accepted 4 January 2006

Abstract. In this paper we present the implementation of Genetic Algorithms (GA) for job scheduling on computational grids that optimizes the makespan and the total flowtime. Job scheduling on computational grids is a key problem in large scale grid-based applications for solving complex problems. The aim is to obtain an efficient scheduler able to allocate a large number of jobs originated from large scale applications to grid resources. Several variations for GA operators are examined in order to identify which works best for the problem. To this end we have developed a grid simulator package to generate large and very large size instances of the problem and have used them to study the performance of GA implementation. Through extensive experimenting and fine tuning of parameters we have identified the configuration of operators and parameters that outperforms the existing implementations in the literature for *static* instances of the problem. The experimental results show the robustness of the implementation, improved performance of static instances compared to reported results in the literature and, finally, a fast reduction of the makespan making thus the scheduler of practical interest for grid environments.

Keywords: Genetic algorithms, job scheduling, computational grid, large size instances, makespan, flowtime.

1. Introduction

The emerging paradigm of grid computing and the construction of computational grids [1] are making the development of large scale applications possible from optimization and other fields. The development or adaptation of applications for grid environments is being challenged by the need of scheduling a large number of jobs to resources efficiently. Moreover, the computing resources may vary in regard to their objectives, scope, structure as well as to their resource management policies such as access and cost.

Job scheduling on computational grids is gaining importance due to the large scale applications of the grid, e.g. optimization [2–4], collaborative/eScience computing [5, 6], data-intensive computing [7, 8] that need efficient schedulers. This scheduling task is much more complex than its version in traditional computing environments. Indeed, the grid environment is dynamic and, also the number of resources to manage and the number of jobs to be scheduled are usually very large making thus the problem a complex large scale optimization problem. This problem is multiobjective in its general definition, as several optimization criteria such as makespan, flowtime and resource utilization are to be matched.

Due to its theoretical and practical relevance, the evolutionary computing research community has started to examine this problem [9–13]. However, the existing approaches in the literature show several limitations: in some works [9, 10, 13] just the uniobjective case is considered and usually either concrete grid environments [9] or the static version of the problem are considered [12, 13]. Moreover, the performace of resulting schedulers has been studied for small size instances; dynamic aspects of this problem have not been addressed so far to grid environments.

In this work we present the implementation of Genetic Algorithms for job scheduling on computational grids that optimizes the makespan and the flowtime. We consider two

versions: the first one is a hierarchic structure of objectives first (optimizes the makespan and next optimizes the flowtime); the second one, both objectives are optimized simultaneously.

We provide several variations for each GA's operator in order to identify which works best for the problem. We have done extensive experimenting and fine tuning of parameters and have thus identified the configuration of operators and parameters that outperforms existing implementations [13] for static instances of the problem. Moreover, in order to deal with realistic large scale instances of the problem, we have developed a grid simulator that simulates realistic grid environments by generating resources and jobs with their own characteristics. We used the simulator to generate large and very large size instances of the problem and have used them to study the performance of GA implementation.

The experimental results show that GA converges to good solutions even though the initial populations are generated by *LJFR-SJFR* (Longest Job to Fastest Resource – Smallest Job to Fastest Resource) which performs worse than other *ad hoc* heuristics such as *Min-Min*. Finally, the resulting scheduler from GA implementation shows a fast reduction on the makespan making thus the scheduler of practical interest for grid environments.

The paper is organized as follows. We present the definition of the problem in Section 2. In Section 3 GA for the problem is presented. The fine tuning of parameters is given in Section 4. In Section 5 we give the results of the performance of the algorithm for static instances as well as for large and very large size instances (both static and dynamic settings). We end up in Section 6 with some conclusions and point out further directions of this research.

2. Problem definition

Job Scheduling in computational grids is a multi-objective optimization problem. In this work we are concerned with 2-objective case. More precisely, we consider the problem formulation in which an instance of the problem consists of the following.

- A *number of independent (user/application) tasks* that must be planned/scheduled. Any task has to be processed entirely in unique resource.
- A *number of heterogenuous machines* candidates to participate in the planning.
- The *workload* of each task.
- The *computing capacity* of each machine (in mips).
- The *expected time to compute*, *ETC*, a matrix of size *number_tasks ´ number_machines*, where *ETC[i][j]* indicates the expected execution time of task *i* in machine *j*. This matrix is either computed from the information on workload and mips or can be explicitely provided.

We aim to minimize the completion time (makespan and flowtime) and utilize the resources effectively. Note that the makespan and flowtime are the most important parameters of the scheduling problem.

3. GA algorithm for job scheduling

Population, solution, fitness

These key components of GA are described below.

- *Initial Population:* Three methods are considered for generating initial populations, the first, Longest Job to Fastest Resource - Shortest Job to Fastest Resource (*LJFR-SJRF*) described in [10], the second, Minimum Completion Time (*MCT*) given in [13] and, the third one, *Random*.

Solution

Feasible solutions are encoded in a vector, called *schedule*, of size *number_tasks*, where *schedule*[*i*] indicates the machine where task *i* is assigned by the schedule. Thus, the values of this vector are natural numbers included in the range [0, *number_tasks-1*]. Moreover, the completion time information, the makespan, flowtime and resource utilization defined next are associated with a solution.

The permutation-based representation is also applicable to the problem. In order to transform the combination-based representation into a permutation-based one the following steps are applied: first, starting from the vector of task-machine allocations, we transform it into bidimensional representation which first dimension represents candidate machines and the other one represents, for any machine, the set of jobs assigned to that machine. Then, we concatenate the sequences of tasks of each machine. The information on how many tasks a machine is assigned to is kept separately as a vector of size *number_machines*.

- *Solution Properties:* A feasible solution of the problem is associated with the following information:
- *completion* is a vector of size *number_machines*, where *completion*[*m*] indicates the time in which machine *m* will finalize the processing of the previous assigned tasks as well as of those already planned for the machine. The value of *completion*[*m*] is calculated as follows:

$$completion[m] = ready\_times[m] + \Sigma_{\{j \in Tasks/schedule[j]=m\}} ETC[j][m]$$

where *ETC*[*j*][*m*] is the value from the expected time to compute.

- *local-makespan* denotes the makespan among the machines included in the *schedule*:

$$local\_makespan = max\ \{completion[i] / \forall i \in Machines\}$$

- *flowtime* denotes the flowtime of the *schedule* computed

by increasingly sorting the tasks assigned to each machine regarding their *ETC*. By letting F_t the time in which the task *t* finalizes, flowtime is then:

$$flowtime = \Sigma_{\{j \in Tasks\}} F_j$$

- *local_avg_utilization* is a parameter used to indicate the quality of a solution with respect to the utilization of resources involved in the *schedule*.

• *Representing Fitness:* The concept of fitness for job scheduling on computational grids has to consider many optimization criteria. One can adapt either a hierarchic or a simultaneous approach. In the former the criteria are sorted by their importance. In the latter approach optimal planning is such in which any improvement with respect to a criterion, causes deterioration in respect to another criterion. In this work both approaches are considered. In the hierarchic approach the objective is to find a schedule of the minimum flowtime given the optimized value of makespan. Two forms of this approach are considered: (a) the criterion with more priority is *local_makespan/local_avg_utilization* and the second criterion is *flowtime*; (b) the criterion with more priority is *local_makespan*, and the second criterion is *flowtime*. In the simultaneous approach both criteria are optimized simultaneously.

GA Operators

We will briefly give here the operators and their versions we have implemented.

Selection operators: The following selection operators were investigated.

- *Select Random*: this chooses the individuals for the pool of pairs to cross uniformly at random.
- *Select Best*: this chooses only individuals with better fitness.
- *Linear Ranking Selection*: Each individual will be chosen with probability linearly proportional to its rank as in [14].
- *Binary* and *N-Tournament Selection.*

Crossover Operators (Vector representation): The crossover operators [15] used for vector representation were:

- *One-point crossover*
- *Two-point crossover*
- *Uniform crossover*
- *Fitness-based crossover*

The last of these considers the value of the fitness function of each father while generating the crossing mask (the genes of individual with larger difference in their fitness are more probable to be interchanged).

Crossover Operators (Permutation representation): Permutation-based crossover operators [11] can be applied to this problem. The crossover operators described above for the vector representation are not valid since they often lead to illegal representations. We considered the operators below:

- *Partially Matched Crossover (PMX)*
- *Cycle Crossover (CX)*
- *Order Crossover (OX-1)*

Mutation operators: We have considered the following operators applied to vector representation.

- *Move*: This operator moves a task from a resource to another one, so that the new machine that is assigned is different.

- *Swap*: Considering of the movements of tasks between machines effective, but often it turns out more useful to make interchanges of the allocations of two tasks.

- *Both (move&swap)*: The mutation by swap hides a problem: the number of jobs assigned to any processor remains inalterable by mutation. A combined operator avoids this problem.

- *Rebalancing*: This operator first improves somehow the solution (by rebalancing the machine loads) and then mutates it.

Replacement operators: The new generation of children *P'* from the generation of parents *P*, is obtained by the choice of the following replacement operators:

- *Steady-state*
- *Generational replacement*
- λ–μ *replacement*
- λ+μ *replacement*
- *Replace always*
- *Replace if better*

Above, we let μ be the size of *P* and λ the size of *P'*.

4. Fine tuning of parameters and operators

We have implemented GA in C++ by adapting the algorithmic skeletons defined in [16]. Though dynamic scheduling is our eventual aim, using static instances we are able to compare the quality of the schedule produced by our GA implementation with that of known schedulers in the literature [13]. Moreover, it was very useful in finding an appropriate combination of operators and parameters that work well in terms of robustness.

The input instances used in this study are generated from the grid simulator for an environment of 16 resources, with computation capacity follows a normal distribution $\mathbf{N}(\mu = 950, \sigma = 75)$, in which a total of 512 tasks of work load that follows a normal distribution $\mathbf{N}(\mu = 250500000, \sigma = 2455000)$.

We give below the tables of values for the parameters and for the case of mutation graphical representation is given (similar representations were done for the rest of operators; we omit them here).

Mutation operators: We obtained the following values for the parameters (see Table 1 and Fig 1).

Table 1. Values of parameters for comparing mutation operators

nb_evolution_steps	*2500*
cross_choice	*CrossTwoPoints*
cross_prabability	*0.7*
pop_size/ intermediate_pop_size	*80 / 78*
select_choice	*SelectBest+shuffle*
mutate_probability	*0.2*
replace_generational	*false*
replace_only_if_better	*false*
start_choice	*StartLJFRSJFR*

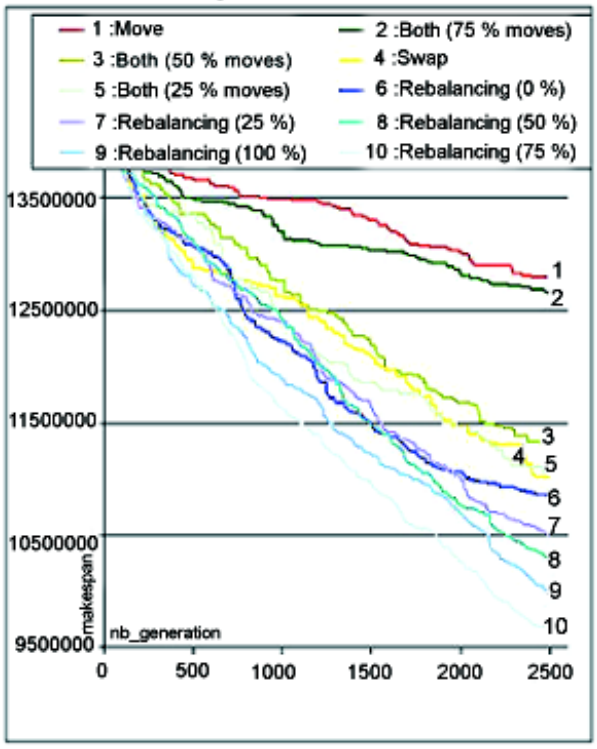

Fig 1. Comparison of different mutation operators (corresponding curves numbered)

Fig 1 clearly indicates that the worse operator of mutation is *Move* and the best one is *Rebalancing.*

Crossover operators: For crossover operators we obtained the values given in Table 2.

From the comparison study operator CX offers the best reduction and with large difference in makespan. In fact, actually the most effective crossover operators in the literature are permutation-based and use precisely this operator [11, 17].

Selection and Replacement operators: For selection we obtained the values given in Table 3 under which the tournament selection works best.

Finally, in regard to the replacement operators we obtained the values given in Table 4. The simple method *Generational* provides the worst results and *Elitist Generational* works best.

Table 2. Values of parameters for comparing crossover operators

nb_evolution_steps	*2500*
cross_prabability	*0.80*
pop_size/ intermediate_pop_size	*80 / 78*
select_choice	*SelectBest+shuffle*
mutate_choice	*MutateRebalancing (pm' = 0.75)*
mutate_probability	*0.2*
replace_generational	*false*
replace_only_if_better	*false*
start_choice	*StartLJFRSJFR*

Table 3. Values of parameters for comparing selection operators

nb_evolution_steps	*2500*
cross_choice	*CrossCX*
cross_probability	*0.80*
mutate_choice	*MutateRebalancing (pm' = 0.75)*
mutate_probability	*0.2*
pop_size / ntermediate_pop_size	*80 / 78*
replace_generational	*false*
replace_only_if_better	*false*
start_choice	*StartLJFRSJFR*

Table 4. Values of parameters for comparing replacement operators

	Simple	*Elitist*	*Replace If Better*	*Steady State*
start_choice	*StartLJFRSJFR*			
select_choice	*SelectLinearRanking*			
nb_evolution_steps	*2500*			*16500*
cross_choice	*CrossCX*			
cross_prabability	*0.80*			*1.0*
mutate_choice	*MutateRebalancing (0.75)*			
mutate_probability	*0.3*			
population_size	*80*			*30*
intermediate_pop_size	*80*	*78*	*80*	*10*
replace_generational	*true*	*false*	*false*	*false*
replace_only_if_better	*false*	*false*	*true*	*false*
W =	***220000***	***214500***	***220000***	***214500***

5. Computational results

We summarize here some of the computational results obtained from our GA implementation. First, we present the results obtained for static instances from [13] where a comparative study of several heuristics for static planning for heterogenous systems is presented. Then, we show the results obtained for large and very large instances in both static and dynamic setting. All the executions were done in Pentium III 500Mhz, 128Mb RAM.

5.1. Small / moderate size instances

The simulation model in [13] is based in *ETC* matrix (see Problem definition). The instances used in [13] are classified into 12 different types, each of them consisting of 100 instances, according to three metrics: task heterogeneity, machine heterogeneity and consistency. This set of in-

stances is actually considered as the most difficult one for the scheduling problem in heterogeneous environments and it is the main reference in the literature. Note that all instances consist of 521 tasks and 16 machines.

GA implementation described in this paper optimizes makespan and flowtime while GA implementation of [13] minimizes only the makespan, therefore we will present and compare the results just for the makespan. Note, however, that optimizing also the flowtime has a negative impact on the overall excecution time of our implementation. The population size used in [13] is 200 and their GA finalizes when either 1000 iterations have been executed, or, when the chromosome elite has not varied during 150 iterations. The executions were done in Pentium III 400Mhz, 1Gb RAM.

In order to compare the results of both GA implementations work *W* made by both implementations is taken as indicative reference. Thus, in [13], W = (0.6 + 0.4) * 200 * 1000 = 200,000 and in our implementation W = (0.8 + 0.4) * 68 * 2500 = 204,000, assuring a comparable amount of work.

Table 5 summarizes the computational results averaged over twenty runs.

Table 5. Comparison of the computational results for static instances

Instance	Min-Min	MCT + LJFR_SJFR	Braun et al GA	Our GA-hierarchic	Our GA-simultaneous
u_c_hihi.0	8460675.00	14665600.26	**8050844.50**	**7747775.57**	**7752349.37**
u_c_hilo.0	164022.44	213423.33	**156249.20**	157457.70	**155571.80**
u_c_lohi.0	275837.34	485591.12	**258756.77**	**253097.28**	**250550.86**
u_c_lolo.0	5546.26	7112.79	**5272.25**	5330.43	**5240.14**
u_i_hihi.0	3513919.25	4193476.36	**3104762.50**	3174335.80	**3080025.77**
u_i_hilo.0	80755.68	92003.30	**75816.13**	77608.98	76307.90
u_i_lohi.0	120517.71	145157.28	**107500.72**	111254.70	**107294.23**
u_i_lolo.0	2779.09	3296.48	**2614.39**	2693.69	**2610.23**
u_s_hihi.0	5160343.00	6510165.67	**4566206.00**	4760014.62	**4371324.45**
u_s_hilo.0	104540.73	121170.49	**98519.40**	101823.10	**983334.64**
u_s_lohi.0	140284.48	190442.12	**130616.53**	138336.54	**127762.53**
u_s_lolo.0	3867.49	4438.42	**3583.44**	3675.73	**3539.43**

Table 5 shows that our implementation outperforms the results of [13] for all but one instance, *u_i_hilo.0*. It is worth mentioning here that our initial populations are worse than the ones used in [13] due to the heuristics used (Min-Min performs much better than LJFR/SJFR). In fact, we observed that our GA spends roughly 55-70% of the total number of iterations to reach the solution of the quality of Min-Min. We would thus expect even better performance of our GA, if initialisation were conducted using Min-Min. On the other hand, our version of hierarchic optimization obtains the results similar to those of [13] but does not outperform them (except two instances).

Notice that GA with hierarchic optimization criteria obtains better results only for instances with consistent *ETC* matrices having a high level of heterogeneity of resources. For the rest of instances, as shown in Table 5, the deviation with respect to [13] is 2.9 % in average (5.91 % in the worst case).

5.2. Large and very large size instances

Any grid scheduler should be able to allocate a large number of tasks to resources. Indeed, the scheduler will be planning tasks originated from many applications running in the grid, hence it is resonable to expect large and very large size instances over time. In order to study the performance of our GA implemenation for this scenario, we developed a grid simulation package with the aim of testing the implementation in a dynamic environment. At this stage of GA we used the simulator for generating large and very large instances in both settings and studied the performance on these instances (detailed results for the dynamic setting will be presented in the full version of the paper). In Fig 6 we give the parametrization used for GA.

The descriptions of instances (for completeness we include also small and average size instances) are given in Table 7 and Table 8 for static and dynamic setting, respectively.

For the dynamic setting we schedule an average of 16 tasks for each resource. The scheduler reschedules tasks that

Table 6. Configuration of GA parameters for large and very large size instances

	Elitist Generational	Steady State
nb_evolution_steps	5 * nbr_tasks	20 * nbr_tasks
pop_size	$\lceil(\log_2(\text{nbr\_tasks}))^2 - \log_2(\text{nbr\_tasks})\rceil$	$4(\lceil\log_2(\text{nbr\_tasks})\rceil - 1)$
intermediate_pop_size	pop_size - 2	pop_size / 3
select_choice	SelectLinearRanking	
cross_choice	CrossCX	
cross_probability	0.80	1.00
mutate_choice	MutateRebalancing (p_m=0.60)	
mutate_probability	0.40	
replace_only_if_better	false	
replace_generational	false	
start_choice	LJFR-SJFR + MCT + Random	
max_time_to_spend	40 secs (static) / 25 secs (dynamic)	

Table 7. Description of static instances

	Small	Average	Large	Very Large
Init./Total hosts	32	64	128	256
Mips	N(1000,175)			
Init./Total tasks	512	1024	2048	4096
Workload	N(250000000,43750000)			
Host selection	all			
Task selection	all			
Local policy	shortest processing time first (sptf)			
Number runs	30			

Table 8. Description of dynamic instances

	Small	Average	Large	Very Large
Init. host	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)	e(3906.25)	e(1953.125)	e(976.5625)
Activation	resource_and_time_interval(250000)			
Reschedule	true			
Host select	all			
Task select	all			
Local policy	shortest processing time first (sptf)			
Nbr runs	15			

have not begun their execution in order to better adapt the planifications. Task interarrival has been set so that once the scheduler activates, there is an average of 1 new task per resource; the scheduler activates approximately in time intervals equal to the average execution time of a task, or when resources availability changes. Finally, new machines are added to the Grid every time an average of 2.5, 2.25, 2.00 and 1.75 tasks have finished their execution (per resource).

The results obtained for the makespan for both cases are given in Table 9 and 10, respectively. We also show the confidence interval.

Table 9. Computational results for static setting

Makespan ± %C.I	Small	Average	Large	Very Large
Elitist Generational (hierarchical)	3975630.27 ±0.5714%	3986741.95 ±0.7950%	4006153.30 ±0.9351%	**4038090.10 ±1.1843%**
Steady State (hierarchical)	**3972614.96 ±0.6421%**	**3979528.76 ±0.7714%**	**3986350.17 ±0.8781%**	3999442.95 ±1.0614%
Elitist Generational (simultaneous)	3999566.76 ±0.7442%	4007250.05 ±0.8810%	4021509.31 ±1.1750%	**4057448.27 ±1.5132%**
Steady State (simultaneous)	**3989635.18 ±0.7250%**	**3993724.92 +0.8905%**	**4001873.91 ±1.0833%**	**4014965.07 ±1.3244%**

Table 10. Computational results for dynamic setting

Makespan ± %C.I	Small	Average	Big	Very Big
Elitist Generational (hierarchical)	4048152.90 ±0.7560%	3988204.13 ±0.8501%	3992023.62 ±1.0724%	**4016478.20 ±1.7805%**
Steady State (hierarchical)	4051858.60 ±0.8109%	3979957.58 ±0.7216%	3981408.54 ±0.9240%	**3978104.73 ±1.4477%**
Elitist Generational (simultaneous)	4062331.15 ±0.8102%	3999261.61 ±0.9912%	4015066.05 ±1.4350%	**4041820.13 ±1.9363%**
Steady State (simultaneous)	**4063425.46 ±0.8225%**	**3994804.90 ±0.8905%**	**3995161.98 ±1.2805%**	**4009852.02 ±1.8390%**

6. Conclusions and Further Work

We have presented new GA implementation coupled with a flexible simulation system to support the job scheduling on computational grids. The results obtained from our GA implementation are very promising; we have identified a setting of parameters and operators that outperform the current best GA Job Scheduling system on a static sceduling benchmark. We plan to extend this work as follows:

- To use other ad hoc heuristics, such as *Min-Min* or *Stratified Min-Min*, for initialisation of population.
- To investigate why certain operators perform better.
- To include experimental results using the simulator over a period of time and study their statistical significance.
- To incorporate our scheduler in real grid-based applications.

7. Acknowledgments

The research work of Fatos Xhafa is partially supported by the Spanish MCYT project TIC2002-04498-C05-02 (TRACER).

References

1. Foster, I. and Kesselman C. The Grid - Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers, 1998.
2. Casanova, H. and Dongarra, J. NetSolve: Network enabled solvers. *EEE Computational Science and Engineering*, Vol 5, No 3, 1998, p. 57–67.
3. Goux, J. P. and Leyffer, S. Solving large MINLPs on computational grids. *Optimization and Engineering*, Vol 3, 2002, p. 327–346.
4. Wright, S. J. Solving optimization problems on computational grids. *Optima*, Vol 65, 2001.
5. Newman, H. B. and Ellisman, M. H. and Orcutt, J. A. Data-intensive e-science frontier research, *Commun. ACM*, Vol 46, No 11, ACM Press, 2003, p. 68–77.
6. 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: Proceedings of Int. Conference on Parallel and Distributed Processing Techniques and Applications. Las Vegas, USA, 2005, p. 1177–1183.
7. Beynon, M.D.; Sussman, A.; Çatalyürek, Ü; Kure, T. and Saltz, J. Optimization for Data Intensive Grid Applications. In: Third Annual International Workshop on Active Middleware Services. California, 2001, p. 97–106.
8. Lenz, C. J., Majewski, D. World-wide Local Weather Forecasts by Meteo-GRID Computing Quarterly Report of the Operational NWP-Models of the Deutscher Wetterdienst, No 28, Offenbach, 2002.
9. Di Martino, V. and Mililotti, M. Sub optimal scheduling in a grid using genetic algorithms. *Parallel Computing*, Vol 30, 2004, p. 553–565.

10. Abraham, A.; Buyya, R., and Nath, B. Nature's heuristics for scheduling jobs on computational grids. In: The 8th IEEE Int. Conference on Advanced Computing and Communications, India, 2000.
11. Zomaya, A.Y.; Teh, Y. H. Observations on Using Genetic Algorithms for Dynamic Load-Balancing. *IEEE Transactions On Parallel and Distributed Systems*, Vol 12, No 9, 2001.
12. Meijer, M. Scheduling parallel processes using Genetic Algorithms. Master thesis. Universitat van Amsterdam, Faculteit der Natuurwetenschappen, Wiskunde en Informatica, 2004.
13. Braun, T. D.; Siegel, H. J.; Beck, N.; Bölöni, L. L.; Maheswaran, M.; Reuther, A. I.; Robertson, J. P.; Theys, M. D. 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*, Vol 61, No 6, 2001, p. 810–837.
14. Whitley, D. The GENITOR Algorithm and Selective Pressure: Why Rank-Based Allocation of Reproductive Trials is Best. In: Proceedings of the 3rd International Conference on Genetic Algorithms. D. Schaffer, ed., Morgan Kaufmann, 1989, p. 116–121.
15. Goldberg, D. E. and Deb, K. A. Comparative analysis of selection schemes used in genetic algoritms. Morgan Kauffman Ed., 1991, p. 69–93.
16. Alba, E.; Almeida, F.; Blesa, M.; Cabeza, J.; Cotta, C.; Dķaz, M.; Dorta, I.; Gabarró, J.; León, C. and Luna, J.; Moreno, L.; Pablos, C.; Petit, J.; Rojas, A.; Xhafa, F. MALLBA: A library of skeletons for combinatorial optimisation. EuroPar2002, LNCS, Vol 2400, Springer, 2002, p. 927–932.
17. Page, A. J. and Naughton, Th. J. Dynamic task scheduling using genetic algorithms for heterogeneous distributed computing. 8th International Workshop on Nature Inspired Distributed Computing. In: Proceedings of the 19th International Parallel & Distributed Processing Symposium, Denver, Colorado, USA, IEEE Computer Society, April 2005.

GENETINIŲ ALGORITMŲ NAUDOJIMAS KOMPIUTERIŲ TINKLUOSE IR KALENDORINIS DARBŲ PLANAVIMAS

J. Carretero, F. Xhafa

Santrauka

Aprašoma, kaip genetinis algoritmas taikomas darbų trukmėms optimizuoti kalendoriniam darbų planavimui, naudojant kompiuterių, sujungtų į tinklą, išteklius. Kalendorinis darbų planavimas, naudojant kompiuterių tinklą, yra aktuali problema, sprendžiant kompleksines, didelio masto problemas. Autorių tikslas – sukurti tokį algoritmą, kuris efektyviausiai paskirstytų teikiamų skaičiuoti darbų srautą į kompiuterių tinklą. Ištirti keli algoritmai, išrinktas geriausias. Sukurtas kompiuterių tinklo darbą imituojantis programinis paketas, jis patikrintas, sprendžiant konkrečius uždavinius. Eksperimentuojant rastas geriausias operatorių ir parametrų derinys, o eksperimento rezultatai atskleidė, jog darbų planavimo laikas sutrumpėjo.

Pagrindiniai žodžiai: genetinis algoritmas, kalendorinis darbų planavimas, kompiuterių tinklas, pavyzdžiai, darbų trukmė, laikas.

Fatos XHAFA. Associate Professor. The Polytechnical University of Catalonia, Barcelona (Spain). He graduated from the Faculty of Natural Sciences, Tirana (Albania) in 1988 and received his PhD in Computer Science, in 1998 from the Department of Languages and Informatics Systems of the Polytechnic University of Catalonia. Research interests: combinatorial optimization, approximation algorithms, mathematical programming, meta-heuristics (especially Tabu Search and Evolutionary Algorithms), and parallel and distributed programming, Grid Computing paradigm and its use in solving large-scale combinatorial optimization problems.

Javier CARRETERO. Master student, the Faculty of Informatics, the Polytechnical University of Catalonia, Barcelona (Spain). Research interests: combinatorial optimization, meta-heuristics and design and implementation of simulation environments for Computational Grids, heuristic methods for solving Job Scheduling on Computational Grids and their use in solving large scale optimization problems.