A GA(TS) Hybrid Algorithm for Scheduling in Computational Grids

Fatos Xhafa[1], Juan A. Gonzalez[1], Keshav P. Dahal[2], and Ajith Abraham[3]

[1] Department of Languages and Informatics Systems
Technical University of Catalonia, Barcelona, Spain
fatos@lsi.upc.edu

[2] School of Informatics, University of Bradford, UK
k.p.dahal@Bradford.ac.uk

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

Abstract. The hybridization of heuristics methods aims at exploring the synergies among stand alone heuristics in order to achieve better results for the optimization problem under study. In this paper we present a hybridization of Genetic Algorithms (GAs) and Tabu Search (TS) for scheduling in computational grids. The purpose in hybridizing these heuristics is to benefit the exploration of the solution space by a population of individuals with the exploitation of solutions through a smart search of the TS. Our GA(TS) hybrid algorithm runs the GA as the main algorithm and calls TS procedure to improve individuals of the population. We evaluated the proposed hybrid algorithm using different Grid scenarios generated by a Grid simulator. The computational results showed that the hybrid algorithm outperforms both the GA and TS for the makespan value but cannot outperform them for the flowtime of the scheduling.

1 Introduction

Meta-heuristics are the *de facto* approach to cope in practice with the computationally hard optimization problems. Meta-heuristics are in fact *hybrid* in their nature since they consist of a high level algorithm that guides the search using other particular methods. For instance, in population based meta-heuristics, such as Genetic Algorithms, the solution space is explored through a population of individuals and there are used methods for generating the initial population, computing the fitness of individuals as well genetic operators to transmit the genetic information from parents to offsprings.

Besides using meta-heuristics as *stand alone* approaches for solving hard combinatorial optimization problems, during the last years, the attention of researchers has shifted to consider another type of high level algorithms, namely hybrid algorithms. These algorithms do not follow any concrete meta-heuristic, but rather they combine other meta-heuristics and/or other methods (e.g. exact methods) yielding thus hybrid meta-heuristics.

The *rationale* behind the hybridization resides in the "no free lunch theorem" [16] stating that "*... all algorithms that search for an extremum of a cost function perform exactly the same, when averaged over all possible cost functions. In particular, if algorithm A outperforms algorithm B on some cost functions, then loosely speaking there must exist exactly as many other functions where B outperforms A.*" Essentially, the theorem states that there is no any search method for optimization which outperforms all other search methods. This suggests that one can use existing algorithms as components for designing new efficient search algorithms and expect improved performance of the newly obtained algorithm for some cost functions.

There are at least two major issues in designing hybrid meta-heuristics: (a) how to choose the (existing) heuristic methods to combine, and (b) how to combine the chosen heuristic methods into new hybrid approaches. Unfortunately, there are no theoretical foundations for these issues. For the former, different classes of search algorithms can be considered for the purposes of hybridization, such as exact methods, simple heuristic methods (ad hoc methods) and meta-heuristics. Moreover, meta-heuristics themselves are classified into local search based methods, population based methods and other classes of nature inspired meta-heuristics. Therefore, in principle, one could combine any methods from the same class or methods from different classes. Regarding the later, there are some attempts for taxonomies of hybrid meta-heuristics [8, 5]; in fact, the common approach is to *try out* in smart ways, based on domain knowledge of problem at hand and characteristics of heuristics methods, different hybrid approaches and shed light on the performance of the hybrid approach through empirical studies. Frameworks that facilitate the fast prototyping have been also provided in the meta-heuristics literature [2, 4].

In this paper, we present a hybrid algorithm for the problem of scheduling independent tasks in computational grids. A computational grid is a distributed infrastructure of computational resources (hardware, software, data storages, etc.) highly heterogenous, interconnected through heterogenous networks. One key issues in Grids is to design efficient schedulers, which will be used as part of middleware services to provide efficient planning of users' tasks to grid nodes.

Recently, heuristic approaches have been presented for the problem [1, 7, 9, 11, 10, 12], however, proper hybrid approaches are lacking. Our hybrid approach combines Genetic Algorithms (GAs) and Tabu Search (TS) methods. Roughly, our hybrid algorithm runs the GA as the main algorithm and calls TS procedure to improve individuals of the population. Our hybrid algorithms deals with the scheduling problem as a bi-objective optimization problem, in which makespan is considered a primary objective and flowtime a secondary one. Such optimization scheme is usually referred to as hierarchic optimization. The proposed algorithm has been experimentally evaluated and the results are contrasted against both GAs and TS for the problem.

The rest of the paper is organized as follows. In Section 2, we briefly present the scheduling of independent tasks considered as a bi-objective optimization problem in this work. In Section 3, types of hybridizations are presented. The

GAs and TS for the problem as well as the hybrid approach are given in Section 4. The experimental study and some computational results are given in Section 5. We conclude in Section 6 with some remarks and indications for future work.

2 Scheduling of independent tasks in computational grids

Many applications are being developed to be run in computational grids to benefit from the large amount of computational resources in such systems. In simple Grid systems such as enteprise grids or campus grids, the user can use queuing systems such as Condor or Sun Grid Engine; even, manual selection of the appropriate machines for running the application is possible in such grids. In large scale and highly heterogenous grids, however, this tedious task is automatically handled by grid schedulers, which are expected to find planning of users' tasks and applications to most appropriate machines.

One class of grid schedulers are batch schedulers, that is, schedulers that compute a planning of a set of tasks/applications altogether to a set of grid nodes. Meta-heuristic approaches are useful for the design of such schedulers, since they usually provide quality solutions in short times.

In this work we are interested in scheduling of independent tasks to grid resources. The formal definition of the problem is based on the definition of the Expected Time to Compute (ETC) matrix in which $ETC[j][m]$ indicates an estimation of how long will it take to complete task j using resource m. Under the ETC matrix model, the independent scheduling can be defined as follows:

- A number of independent *tasks* to be allocated to grid resources. Each task has to be processed entirely in a single resource and is not preempted (once started, a task runs until completion).
- A number of *machines* candidates to participate in the allocation of tasks.
- The *workload* (in millions of instructions) of each task.
- The *computing capacity* of each machine (in *Mips*).
- The ready times, denoted $ready_m$, indicating when machine m will have finished the previously assigned tasks. At the beginning, usually ready times are considered equal to zero (all machines in the machine set are available for task allocation).
- The ETC matrix of size $nb\_tasks \times nb\_machines$, where $ETC[j][m]$ is the value of the expected time to compute of task j in machine m.

The quality of a schedule can be measured using several optimization criteria, such as minimizing the *makespan* (that is, the finishing time of the latest task), the *flowtime* (i.e., the sum of finalization times of all the tasks), the *completion time* of tasks in every machine (closely related to makespan), or maximizing the resource utilization. In this work we consider that the most important criterion is that of minimizing the *makespan*. Additionally, we consider the minimization of the *flowtime* of the grid system as a secondary criterion. These two criteria are formally defined as follows: *makespan*: $\min_{S_i \in Sched}\{\max_{j \in Tasks} F_j\}$ and, *flowtime*: $\min_{S_i \in Sched}\{\sum_{j \in Tasks} F_j\}$, where F_j denotes the time when task j

finalizes and *Sched* is the set of all possible schedules. Notice that by considering the makespan as the main objective to optimize and the flowtime as a secundary goal, we aim at designing a hierarchical algorithm, in which the value for makespan can not be worsened when optimizing the flowtime.

3 Hybridization of meta-heuristics

As mentioned earlier, the hybridization started as an approach that tries to combine fully or partially two or more algorithms to enhance the performance of stand alone search method for optimization problems. To achieve such goal, the hybridization should be able to embed the best features of the combined algorithms into a new high level algorithm.

Current hybrid models take into account two main aspects: (1) Type of methods to hybridize, and (2) Level of hybridization. The first refers to the type of the methods to be hybridized. Essentially we could consider two cases: (a) meta-heuristics + meta-heuristics and (b) meta-heuristics + specific search method. In the first case the components are meta-heuristics while in the later, a meta-heuristic is combined with another type of search method, which could be an exact algorithm, dynamic programming, constraint programming or other AI techniques. In this work we are considering the first case, being the meta-heuristics the GAs and TS method.

The level of hybridization, on the other hand, refers to the degree of coupling between the meta-heuristics, the execution sequence and the control strategy.

Level of hybridization. *Loosely coupled*: in this case the hybridized meta-heuristics preserve their identity, namely, their flow is fully used in the hybridization. This case is also referred to as *high level of hybridization*. *Strongly coupled*: in this case, the hybridized meta-heuristics inter-change their inner procedures, resulting in a *low level of hybridization.*

Execution sequence. *Sequential* (the meta-heuristics flows are run sequentially) or *Parallel* (the meta-heuristics flows are run in parallel.)

Control strategy. *Coercive*: the main flow is that of one of the meta-heuristics, the other meta-heuristics flow is subordinated to the main flow. *Cooperative*: the meta-heuristics explore the solution space cooperatively (eventually, they can explore different parts of the solution space.)

4 The proposed GA(TS) hybrid approach

For the design of our hybrid approach we consider two well-known meta-heuristics: Genetic Algorithms (GAs) and Tabu Search (TS). Both GAs and TS have been developed for the independent task scheduling in Xhafa et al. [11] and [12] in sequential setting. We have considered the Steady-State GA in this work. The choice of these two meta-heuristics is based on the following observations. First, grid schedulers should be very fast in order to adapt to dynamic nature of computational grids. Therefore, a fast convergence of the main algorithm is preferable in this case, which can be achieved through a good tradeoff between exploration and exploitation of the search. Second, in order to achieve high quality planning in a very short time, it is suggestive to combine the *exploration* of the solution space by a population of individuals with the *exploitation* of neighborhoods of solutions through local search. In such case, GAs and TS are among the best representatives of population based and local search methods, respectively.

We are thus considering the case of hybridization of two meta-heuristics running in sequential environment. We have considered a low level hybridization and the coercive control strategy. Roughly, our hybrid algorithm runs the GA as the main algorithm and calls TS to improve individuals of the population.

The hybridization scheme is shown in Figure 1. It should be noted that in the hybridization scheme in Figure 1, instead of replacing the mutation procedure of GAs by the TS procedure, we have added a new function to the GA Population class (namely `apply_TabuSearch`) for applying the TS. This new function could be applied to any individual of the current population, however, this is computationally costly. In our case, given that we want to run the Grid scheduler in short times, the `apply_TabuSearch` is applied with small probability[4]. In fact, this parameter can well be used to tune the convergence of the GA since TS usually provides substantial improvements to individuals.

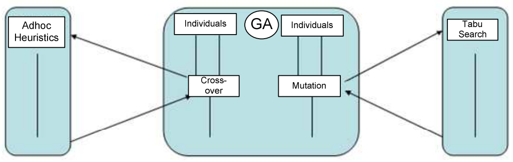

Fig. 1. The hybrid GA(TS) scheme.

We shortly present next both the GA and TS meta-heuristics for independent task scheduling in computational grids (refer to [11] and [12] for details.)

4.1 GAs for the scheduling problem in Grids

GAs are a population-based approaches where individuals represent possible solutions, which are successively evaluated, selected, crossed, mutated and replaced by simulating the Darwinian evolution found in nature. We have implemented the Steady State version of GAs. In Steady State GAs, a few good individuals of population are selected and crossed. Then, the worst individuals of the population are replaced by the newly generated descendants; the rest of the individuals of the population survive and pass to the next generation. The rest of genetic operators and methods are as follows: *Initialization methods* are MCT and LJFR-SJFR implemented in [14, 15]; *Selection operator*: Linear ranking;

[4] This is a user input parameter. For the purposes of this work, `apply_TabuSearch` is applied roughly to 30% of individuals

Table 1. Simulators' configuration.

	Small	Medium	Large
Init./Total hosts	32	64	128
Mips	n(1000, 175)		
Init./Total tasks	512	1024	2048
Workload	n(250000000, 43750000)		
Host selection	All		
Task selection	All		
Local policy	SPTF		
Number of runs	30		

Crossover operator: Cycle Crossover (CX); *Mutation operator*: Mutate Rebalancing. The concrete values for the rest of parameters are given in Section 5.

4.2 Tabu Search for the scheduling problem in Grids

Tabu Search (TS) has shown its effectiveness in a broad range of combinatorial optimization problems and distinguishes for its flexibility in exploiting domain/problem knowledge. The main procedures used in TS are summarized next. The *initial solution* is found using Min-Min method [14]. Regarding *historical memory*, both short and long term memories have been used in TS algorithm. For the *recency* memory, a matrix TL ($nb\_tasks \times nb\_machines$) is used to maintain the tabu status. In addition, a tabu hash table (TH) is maintained in order to further filter the tabu solutions. The neighborhood exploration is done using a steepest descent - mildest ascent method using two types of movements, namely, transfer (moves a task from a machine to another one, appropriately chosen) and swap (two tasks assigned to two different machines are swapped). Further, several *aspiration criteria* are used to remove the tabu status of movements. They are defined using the fitness of solutions as well as information from recency matrix. *Intensification* is implemented using elite solutions while *soft diversification* uses penalties to ETC values, task distribution and task freezing. Finally, *strong diversification* is implemented using large perturbations of solutions.

The concrete values for the rest of parameters are given in Section 5.

5 Experimental study

We have used a Grid simulator [13] to evaluate our hybrid algorithm.

Simulation environment setting. For the evaluation of the GA(TS) hybrid algorithm, we have used three Grid scenarios: small, medium and large size. They consist, respectively, of 32 hosts/521 machines, 64 hosts/1024 machines, and 128 hosts / 2048 machines. Each scenario is generated from the simulator but the number of tasks and machines are kept constant, that is, for both of them, respectively, the number of initial tasks equals the total number of tasks in the system and and the initial number of machines equals the total number of machines.

The configuration of simulator follows the parameters given in Table 1. In the table `n(·,·)` refers to normal distribution; `SPTF` stands for Shortest Processing

Time First local policy. The parameter values of the GA and TS algorithms used in the hybrid algorithm are given in Tables 2 and 3.

Table 2. Parameter values of GA.

Parameter	**Value**
evolution steps	$20 \cdot nb\_tasks$
population size	$4 \cdot (\log_2(nb\_tasks) - 1)$
intermediate pop.	$(pop\ size)/3$
cross probab.	1.0
mutation probab.	0.4

Table 3. Parameter values of TS.

Parameter	**Value**
#iterations	$nb\_tasks \cdot nb\_mach$
max. tabu status	$1.5 \cdot nb\_mach$
#repetitions before activating intensific./diversific.	$4 * \ln(nb\_tasks) \cdot \ln(nb\_mach)$
#iterations per intensific./diversific.	$\log_2(nb\_tasks)$
#iterations for aspiration criteria	$max\_tabu/2 - \log_2(max\_tabu)$

Computational results and evaluation. The simulator is run[5] 30 times for each scenario and computational results for makespan and flowtime are averaged. Standard deviation (at 95% confidence interval) is also reported. The results for makespan and flowtime are given in Table 4 and Table 5, resp.

Table 4. Makespan values.

	Small	**Medium**	**Large**
GA (hierarchic)	2808662.116	2760024.390	2764455.222
	±1,795%	±1,010%	±0,745%
TS (hierarchic)	2805531.301	2752355.018	**2748878.934**
	±1,829%	±1,056%	±0,669%
GA(TS) (hierarchic)	**2805519.428**	**2751989.166**	2812776.300
	±1,829%	±1,058%	±1,176%

Table 5. Flowtime values.

	Small	**Medium**	**Large**
GA (hierarchic)	**709845463.699**	**1405493291.442**	2811723598.025
	±1,209%	±0,655%	±0,487%
TS (hierarchic)	710189541.278	1408001699.550	2812229021.221
	±1,124%	±0,616%	±0,455%
GA(TS) (hierarchic)	711183944.069	1409127007.870	**2811605453.116**
	±1,174%	±0,604%	±0,465%

As can be seen from Table 4, for makespan value the GA(TS) outperforms both GA and TS for small and medium size grid scenarios but achieves worse value for large size scenario. On the other hand, from Table 5, we can see that GA(TS) performs better than both GA and TS for flowtime value only for large size instances. So, GA(TS) performs better for makespan value, which is considered primary objective in hierarchic version, than for flowtime parameter, which is a secondary objective. In fact, close to (sub-)optimal solutions, makespan and flowtime behave as contradictory objectives and thus under our hierarchic model, the improvements of flowtime are difficult to happen.

6 Conclusions

In this paper we have presented a hybrid GA(TS) algorithm for the problem of independent scheduling in computational grids. The hybridization follows a low level approach in which GA is the main flow and TS is subordinated to it. The

[5] AMD Athlon 64 3200+, 2GB RAM.

objective function considered is that of bi-objective in which makespan is primary objective and flowtime is secondary. The experimental evaluation showed that GA(TS) outperforms both GA and TS for makespan values of small and medium size grid scenarios and for flowtime values of large size grid scenarios.

The GA(TS) hybridization scheme is very appropriate for parallel implementation, by running TS method to all individuals of GA population in parallel.

References

1. A. Abraham, R. Buyya, and B. Nath. Nature's heuristics for scheduling jobs on computational grids. In *The 8th IEEE International Conference on Advanced Computing and Communications*, India, 2000.
2. E. Alba, F. Almeida, M. Blesa, C. Cotta, M. Díaz, I. Dorta, J. Gabarró, C. León, G. Luque, J. Petit, C. Rodríguez, A. Rojas, and F. Xhafa. Efficient parallel LAN/WAN algorithms for optimization. The Mallba project. *Parallel Computing*, 32(5-6):415–440, 2006.
3. T. Braun, H. Siegel, N. Beck, L. Boloni, M. Maheswaran, A. Reuther, J. Robertson, M. Theys, and B. Yao. 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).
4. S. Cahon, N. Melab and E. Talbi. Building with paradisEO reusable parallel and distributed evolutionary algorithms. *Parallel Computing* 30, 5-6, 677-697, 2004.
5. L. Jourdan, M. Basseur and E. Talbi. Hybridizing Exact Method and Metaheuristics: A Taxonomy. *European Journal of Operational Research*, (Online, 2008).
6. H.C. Lau, W.C. Wan, M.K. Lim, and S. Halim. A Development Framework for Rapid Meta-Heuristics Hybridization. In *Proc. of the 28th Annual International Computer Software and Applications Conference*, 362-367, 2004.
7. G. Ritchie and J. Levine. A fast, effective local search for scheduling independent jobs in heterogeneous computing environments. TechRep, Centre for Intelligent Systems, University of Edinburgh, 2003.
8. E. Talbi. A Taxonomy of Hybrid Metaheuristics. *J. of Heur.* 8(5), 541-564, 2002.
9. F. Xhafa. A Hybrid Evolutionary Heuristic for Job Scheduling in Computational Grids. Springer Series: Studies in Comp. Intell., Vol. 75, Chap. 10, 2007.
10. F. Xhafa, L. Barolli, and A. Durresi, An Experimental Study on Genetic Algorithms for Resource Allocation on Grid Systems, JOIN 8(4), 427 - 443, 2007.
11. F. Xhafa, J. Carretero, A. Abraham. Genetic Algorithm Based Schedulers for Grid Computing Systems. International Journal of Innovative Computing, Information and Control, Vol. 3, No.5, pp. 1-19, 2007.
12. F. Xhafa, J. Carretero, B. Dorronsoro and E. Alba. Tabu Search Algorithm for Scheduling Independent Jobs in Computational Grids. *Computers and Informatics*, 2009. To appear.
13. F. Xhafa, J. Carretero, L. Barolli and A. Durresi. Requirements for an Event-Based Simulation Package for Grid Systems. *JOIN*, 8(2):163-178, 2007.
14. F. Xhafa, J. Carretero, L. Barolli and A. Durresi. Immediate Mode Scheduling in Grid Systems. *Int. J. of Web and Grid Services*, Vol.3 No.2, 219-236, 2007.
15. 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.
16. D.H. Wolpert, W.G. Macready. No Free Lunch Theorems for Optimization, IEEE Transactions on Evolutionary Computation 1(1), 67-82, 1997.