Alea – Grid Scheduling Simulation Environment

Dalibor Klusáček, Luděk Matyska, and Hana Rudová

Faculty of Informatics, Masaryk University,
Botanická 68a, 602 00 Brno, Czech Republic
{xklusac,ludek,hanka}@fi.muni.cz

Abstract. This work concentrates on the design of a system intended for study of advanced scheduling techniques for planning various types of jobs in a Grid environment. The solution is able to deal with common problems of the job scheduling in Grids like heterogeneity of jobs and resources, and dynamic runtime changes such as arrivals of new jobs.
Our new simulator called Alea is based on the GridSim simulation toolkit which we extended to provide a simulation environment that supports simulation of varying Grid scheduling problems. To demonstrate the features of the GridSim environment, we implemented an experimental centralised Grid scheduler which uses advanced scheduling techniques for schedule generation. By now local search based algorithms and some dispatching rules were tested.
The scheduler is capable to handle both static and dynamic situation. In the static case, all jobs are known in advance while the dynamic situation means that jobs appear in the system during simulation. In this case generated schedule is changing through time as some jobs are already finished while the new ones are arriving. Comparison of FCFS, local search and dispatching rules is presented for both cases and we demonstrate that the new local search based algorithm provides the best schedule while keeping the running time acceptable.

Key words: Grid scheduling, Local search, Dispatching rules, Simulation with GridSim

1 Introduction

Grid [9] is generally understood as a distributed and heterogeneous computing system with various types of different resources. First of all it is important to maximise their usage, on the other hand it is also desirable to provide nontrivial quality of service to the users and their applications.

We are proposing a complex and *extensible* simulation environment unlike to frequently used ad-hoc simulators [2, 12, 24, 3] that is able to model various situations such as different types of jobs and applications or different Grid topology and then evaluate proposed solutions and algorithms. The goal is to use the simulator to design Grid schedulers with differing scheduling techniques and test their behaviour in different but fully controlled conditions.

We apply known advanced scheduling techniques [11, 5, 22] to the Grid scheduling problem [8]. Currently we are focusing on local search based algorithms [11]

and dispatching rules [13, 22]. Various techniques were applied in related problems such as job scheduling on single machine [20, 17], identical parallel machines [24, 2] or heterogeneous parallel machines [21, 3]. Unfortunately, these solutions are usually tested only in static situation, i.e., when all the jobs are known to the scheduler before execution. One of our main interest is the dynamic aspect of scheduling [25, 4, 16], i.e., scheduling of jobs arriving during the system run [3, 12], sometimes referred to as incremental scheduling [16] or on-line reasoning [4].

Using the extended GridSim environment we can simulate scheduling and execution of different types of non preemptive jobs in both static and dynamic fashion on resources composed of parallel and heterogeneous machines. Administrator's demands on resource utilisation can be satisfied by makespan minimisation and user requirements can be handled through optimisation of the total tardiness of all jobs. The simulation environment allows us an easy comparison of the scheduling algorithms. Among other algorithms we have implemented local search algorithms [15, 14] for both the static and dynamic problems. The typical use of the local search for the static problems is related with a high computational costs. Our experiments show that this cost can be significantly reduced for dynamic problems while its optimisation performance is preserved. This seems to be very interesting since we are not aware of any relevant work in the area of local search for dynamic problems[1].

The structure of this paper is following. The next section describes the simulation toolkit we used as the basis for the Alea simulator, the extensions we have done, and it presents design description of the scheduler. The following section describes the algorithms used to create and optimise the schedule. Next we present some experimental results, and the last section concludes our research and discuss the future work.

2 Characteristics of the Alea Simulator

There are numerous Grid simulators that provide various functionality. Bricks [23] is designed for simulations of client–server architectures in Grids, SimGrid [18] is used for the simulation and development of distributed applications in heterogeneous and distributed environment. Simbatch [10] allows to evaluate scheduling algorithms for batch schedulers and MicroGrid [19] can be used for systematic study of the dynamic behavior of applications, middleware, resources, and networks. As the basis of our simulator we use Java based simulation toolkit GridSim [6]. This toolkit is flexible and universal and it has a very good documentation. It provides functionality to simulate the basic Grid environment and its behaviour. GridSim provides simple implementation of common entities such as computational resources or users and also allows to simulate simple jobs, network topology, data storage and other useful functionalities. However, provided

[1] Our investigation and questions addressed to the Institute on Resource Management and Scheduling of the European CoreGRID Network of Excellence to get information about previous applications of local search-based algorithms on dynamic Grid scheduling problems did not give any relevant response.

implementations are too simple and it was necessary to extend these entities to fit more complex requirements. This has been done by implementing new Java class which inherits from the existing GridSim class. Then new functions can be implemented or modified and new parameters can be added to provide the desired functionality of this new entity. Since all important components like Grid resource, job, etc., are defined in separate classes, it is very easy to modify them or create a new one.

We have developed new specialised entities such as the centralised scheduler, the jobs with special parameters or the submission system with dynamically appearing jobs that allows us to build complex dynamic simulations of Grid environment. There exists decentralised scheduler implemented in GridSim [1], but it does not allow dynamic behaviour of the system and, due to the older version of GridSim, it is not capable of simulating network topology and other recent features. The reason is the incompatibility between older and new GridSim versions. The following text describes our solution and its main features.

2.1 Description

The Alea simulator[2] is modular, composed of independent entities which correspond to the real world. It consists of the centralised scheduler, the job submission system, and the Grid resources. These entities communicate together by message passing. Currently Grid users are not directly simulated but a job generator attached to the job submission system is used to simulate job arrivals.

The *Job submission system* stores jobs before and after they are executed and communicates with the scheduler to get a scheduling strategy, which it further uses to select a resource to execute a job.

The *Job generator* is attached to the job submission system and it is used to simulate job arrivals. It generates new synthetic jobs that appear during simulation run. The job arrival times correspond to the selected statistical distribution. Currently we support uniform and normal distribution, but it is easy to add new distributions or real workload data.

The *scheduler* is responsible for schedule generation and further optimisation. In the dynamic situation this schedule may change in time as some jobs are already finished while new ones appear. Since it is a standalone entity it has to be able to communicate with other entities, mainly with the job submission system. To keep the scheduler extensible it was designed as a modular entity composed of three main parts. The first part is responsible for communication with the job submission system. The second part stores dynamic information about each Grid resource such as jobs currently being executed or prepared schedule. It also implements functions that approximate makespan, tardiness, and other values important for the scheduling process. These information are used by the third part of the scheduler, i.e., by the scheduling algorithms.

[2] Alea can be downloaded from `http://www.fi.muni.cz/~xklusac/alea`.

Each *Grid resource* is responsible for the job execution. The resource is selected by the scheduler and job is then submitted by the job submission system. Completed jobs are returned to the job submission system.

2.2 Communication Scheme

The figure 1 shows the common communication scheme between the job submission system, scheduler, and one Grid resource. Job submission system submits job descriptions to the scheduler. The scheduler uses the list of all available Grid resources and their parameters such as the number of CPUs and their rating. The scheduler is centralised, therefore it has information about and access to all available resources in the system. On the basis of this information scheduler generates *separate schedule* for each Grid resource. Using these schedules and also information of jobs currently in execution *the scheduler is able to approximate various parameters of the schedule* such as makespan or expected tardiness for the jobs before their execution and completion. This allows the scheduler to compare two different schedules with respect to the optimisation criteria and select the better one.

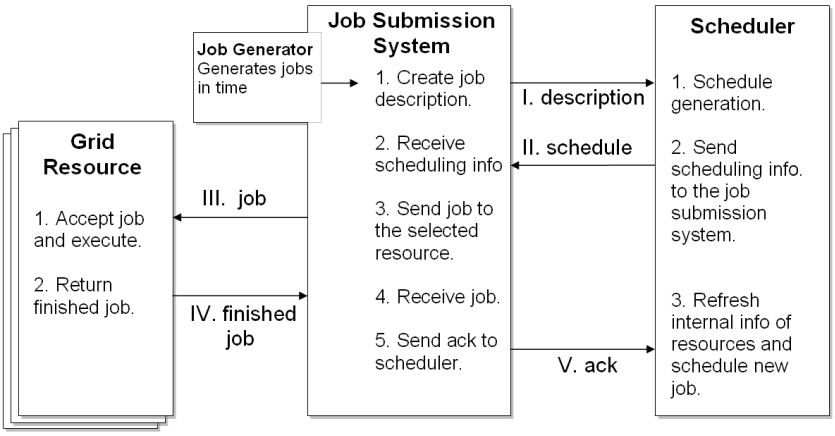

Fig. 1. Communication scheme between the job submission system, the scheduler and the Grid resources.

According to the constructed schedule and the current simulation time the scheduler responds to the submission system with scheduling information, i.e., provides information which resource was selected to execute the job. It is important to notice that this communication is *asynchronous*. Job submission system does not wait for the response from the scheduler and sends the job descriptions as the new jobs are available from the job generator. On the other hand the scheduler sends the scheduling information to the job submission system according to the current load of the resources. The scheduler works with the

job description while the job stays with the job submission system. This helps to save scheduler's network bandwidth and prevents the scheduler to become a bottleneck of the whole system.

Once some job is finished the job submission system sends the acknowledge message to the scheduler. The scheduler then updates its internal information of the current resource load. It is also an impulse to check whether another job should be sent on a resource to prevent the resource from being idle.

2.3 Extensibility

Since the Alea simulator is modular and the main functionality of the scheduler is divided into separate parts, it is easier to simulate different types of job, scheduling algorithms or optimisation criteria by making small changes in existing simulator. For example, if we want to test some new scheduling algorithm we will modify only the appropriate class. If we want to schedule different type of jobs we will only change the job generator and possibly corresponding objective function in the scheduler. The rest of the classes stays intact so the experiments can be repeated with the exactly same setup. The changes are encapsulated and the results are easily comparable. The Alea's behaviour is driven by the events, the active entities like job submission system, scheduler or resources are simulated as independent interacting entities, which makes this solution more closer to the real world. Therefore the simulation environment is ready for a future extensions, e.g., adding network simulation or fault tolerance.

3 Problem Description and Algorithms

The Grid scheduling problem [8] is generally defined by a set of resources (typically machines, storage, memory, network, etc.), a set of tasks, an optimality criterion, an environmental specification and by other constraints. Grid is also a typical example of the dynamic environment. The problem complexity strongly depends on the machine, job, and environmental characteristics. We understand the machine as a computational resource with one or more CPUs. The machine environment may consist of a single machine, identical parallel machines, parallel machines with different speeds, etc. Machines may also become unavailable (breakdown). Job parameters are also very important. Jobs have computational length (processing time), may require one or more CPUs, they may have various release and due dates or priorities (weight). Some jobs may be migrated during the execution from one resource to another (preemption) or may have the precedence constraints typical for workflows or parallel DAG applications. Some jobs require only specific kind of machines (machine – job suitability). Problem complexity rises yet more with additional parameters such as Grid network topology, network bandwidth requirements or fault tolerance. The goal of the scheduling is to satisfy user's and system administrator's demands, e.g., to minimize the total or weighted tardiness of the jobs or to minimize the makespan.

Currently we consider system with no failures such as resource failure or job loss, etc. The system is composed of parallel multiprocessor machines. Different machines may have different CPU rating. We also assume that the computational length of the job is known prior to its execution and job requires one or more CPUs for its run. We simulate the static situation when all jobs are known in advance as well as dynamic situation when new jobs appear in the system during execution so the generated schedule is changing through the time as some jobs are already finished while the new ones are arriving. In such case the scheduler has to create schedule incrementally. Our scheduler does not support job preemption or job migration yet.

We have implemented various algorithms to minimise the makespan, the total tardiness or the number of delayed jobs. Different *queue-based policies* such as First Come First Served (FCFS), Earliest Release Date (ERD) or Earliest Due Date (EDD) [13] were implemented. We also proposed and implemented various algorithms based on the construction of the *global schedule* such as composite dispatching rules or local search based algorithms such as Tabu search [15, 14], Hill climb or Simulated annealing [11]. We will concentrate on some of the interesting results with the total tardiness minimization that demonstrate the variability of the simulator.

3.1 Dispatching Rules and Local Search

We developed composite dispatching rules [14] that select a proper resource in the first step and then use common dispatching rule to put a new job into the existing schedule of this resource. *Minimum Tardiness Earliest Due Date (MTEDD) dispatching rule* selects such resource where the overall tardiness after adding new job will rise minimally. In that resource's schedule the job is placed according to its due date so the jobs with earlier due dates will be executed earlier. Generally it leads to the minimization of the tardiness of this job. Another option is to apply *Minimum Tardiness Earliest Release Date (MTERD) dispatching rule* where the job is placed to the resource's schedule wrt. its release date.

When using local search optimisation an initial schedule is always required. The scheduler uses two steps to create the final schedule. The MTEDD (or MTERD) dispatching rule is used for initial schedule generation then the *Tabu search* [14] algorithm is applied to optimise this schedule. The algorithm moves delayed jobs from one resource's schedule to another and checks if this move is good or not. The job is placed into tabu list to prevent cycling of the algorithm. The Tabu search finishes when there is no job in the schedule expected to be delayed. It also finishes after a fixed number of non improving moves or when the upper bound of iterations is reached. Detailed description of the dispatching rules and local search algorithms can be found at [15, 14].

4 Results and Discussion

The experiments were performed on the Intel Pentium 4 2.6 GHz machine with 512 MB RAM. The tests were run for a different number of available machines

with different CPU rating. Each machine has 4 CPUs. Tests were done for two to eight machines available in the Grid. Single machine situation was not tested because Tabu search only moves jobs between different schedules. We have run tests for 1000 jobs with release dates and due dates. The release date is taken as a hard constraint in the sense that the job cannot start its execution earlier while the due date is a soft constraint which may not be complied. The data sets were generated synthetically and results were averaged from 20 different data sets. The scheduler optimises the total tardiness of all jobs. In the following, we compare the results between the *FCFS*, *MTEDD dispatching rule*, the *MTERD dispatching rule*, and their *Tabu search* optimisation.

The figure 2 shows the total tardiness and the time required to generate the schedule when using FCFS, MTEDD, MTERD, or their Tabu search extensions in the static situation (all the jobs were known to the scheduler before the start of the scheduling process). In the dynamic case the simulated environment was

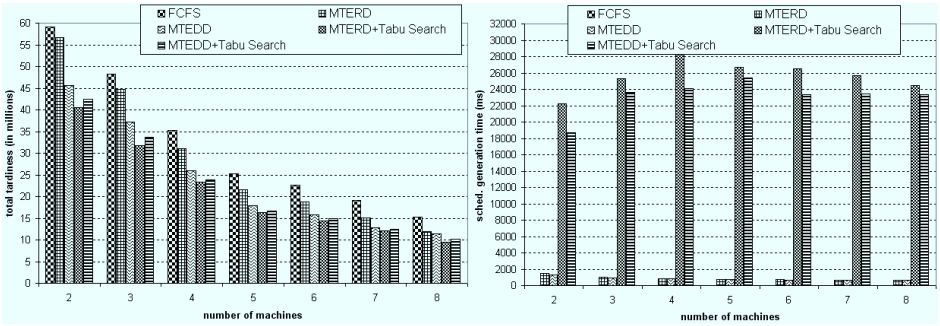

Fig. 2. Total tardiness and total schedule generation time for FCFS, MTEDD, MTERD and Tabu search in static situation.

identical to the static simulation example but the jobs were submitted to the system during the execution. The time intervals were generated with the uniform distribution. In the present experiment, the Tabu search optimisation with the fixed number of iterations was performed in three different ways. In the first scenario the Tabu search was run for each newly arrived job. In the other two cases the Tabu search was run only after each five or ten jobs. The figure 3 shows the total tardiness and the time required to generate the schedule when using FCFS, MTEDD, MTERD and Tabu search applied to initial MTERD.

4.1 Discussion

It is clear that the total tardiness strongly depends on the number of available machines. We can see that the Tabu search makes improving moves in all tested situations. On the other hand the basic methods (FCFC, MTEDD and MTERD) are much faster than the Tabu search in both static and dynamic situation.

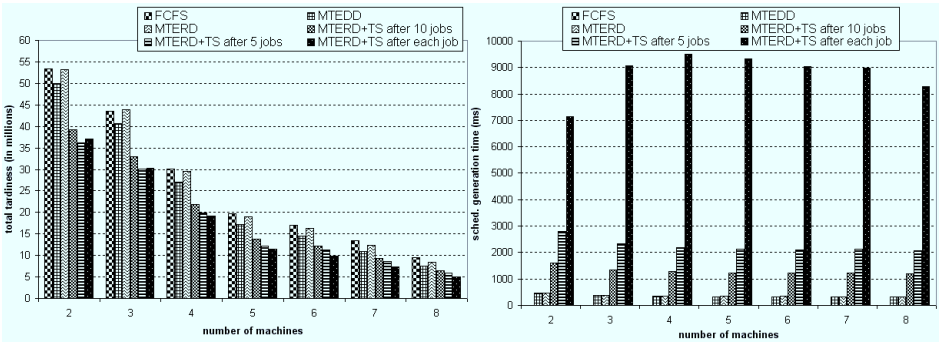

Fig. 3. Total tardiness and total schedule generation time for FCFS, MTEDD, MTERD and Tabu search in dynamic situation.

The static MTERD schedule is not very good but the Tabu search is able to perform more significant improvement in the value of objective function. The figure 3 also shows that the total time required to create schedule (i.e., sum of all times required to include new job into existing schedule) rises when the Tabu search is used for each newly arriving job in contrast to the situation when it is performed only after every five or ten new jobs. The reason why the Tabu search is much slower than MTEDD is because it makes more complex changes in existing schedule and computes the expected total tardiness. This value has to be calculated many times which makes it much more time consuming than MTEDD. The Tabu search combined with MTERD was the best approach wrt. the total tardiness minimisation in comparison with the other algorithms applied. When the Tabu search optimisation is run only after every five or ten jobs, the total time increase is acceptable while the resulting schedule is still very good among all the compared schedules. As expected, the time required to perform Tabu search in the dynamic situation becomes stable with the growing number of available machines. We used the same number of jobs for all the simulations, i.e., with more machines the size of the schedule decreases because jobs previously scheduled are already finished and the algorithm in each run works with the smaller number of jobs. It is natural that in such case the calculation of expected tardiness of yet unfinished jobs on one machine is faster but on the other hand it has to be calculated for higher number of machines.

5 Conclusion and Future Work

In our research we managed to propose an extensible simulation environment Alea usefull to design and test the scheduling algorithms for some typical Grid scenarios. Using this environment, new centralised scheduler with different scheduling algorithms was implemented and its behaviour was evaluated. The tested Tabu search extensions of MTEDD and MTERD methods demonstrated the added value of the local search in dynamic cases. An easy combination of the

scheduling algorithms using our environment allowed to detect promising direction in the development of new efficient local search algorithms applicable for Grid scheduling as a part of the evaluation of the new simulation environment. Further study of local search-based algorithms designed to solve dynamic problems is the main subject of our current work since it is a new unexplored area [15, 14].

Our future work will on one hand focus on extensions of the simulation environment, e.g., addition of network topology, support for different job types including preemptive and priority jobs, workflows, etc. We will also be able to model job migration or an estimated running time of the job. The environment will be further extended to simulate different failures (resource or job crash, network disconnection, etc.) to test robustness of scheduling algorithms. On the other hand, we plan to use this environment to implement and evaluate differing scheduling algorithms and technologies, such as Backfilling or Convergent Scheduling [7].

Acknowledgments. This work was supported by the Ministry of Education, Youth and Sports of the Czech Republic under the research intent No. 0021622419 and by the Grant Agency of the Czech Republic with grant No. 201/07/0205.

References

1. D. Abramson, R. Buyya, M. Murshed, and S. Venugopal. Scheduling parameter sweep applications on global Grids: A deadline and budget constrained cost-time optimisation algorithm. *International Journal of Software: Practice and Experience (SPE)*, 35(5):491–512, 2005.
2. V. A. Armentano and D. S. Yamashita. Tabu search for scheduling on identical parallel machines to minimize mean tardiness. *Journal of Intelligent Manufacturing*, 11:453–460, 2000.
3. R. Baraglia, R. Ferrini, and P. Ritrovato. A static mapping heuristics to map parallel applications to heterogeneous computing systems: Research articles. *Concurrency and Computation: Practice and Experience*, 17(13):1579–1605, 2005.
4. J. Bidot. *A General Framework Integrating Techniques for Scheduling under Uncertainty*. PhD thesis, Institut National Polytechnique de Toulouse, France, 2005.
5. P. Brucker. *Scheduling Algorithms*. Springer Verlag, 1998.
6. R. Buyya and M. Murshed. GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for Grid computing. *The Journal of Concurrency and Computation: Practice and Experience (CCPE)*, 14:1175–1220, 2002.
7. G. Capannini, R. Baraglia, D. Puppin, L. Ricci, and M. Pasquali. A job scheduling framework for large computing farms. In *SC07 International Conference for High Performance Computing, Networking, Storage and Analysis*, 2007. To appear.
8. P. Fibich, L. Matyska, and H. Rudová. Model of Grid Scheduling Problem, Exploring Planning and Scheduling for Web Services, Grid and Autonomic Computing, Papers from the AAAI-05 workshop. Technical Report WS-05-03, AAAI Press, 2005.

9. I. Foster and C. Kesselman. *The Grid 2: Blueprint for a New Computing Infrastructure, second edition*. Morgan Kaufmann, 2004.
10. J.-S. Gay and Y. Caniou. Simbatch: an API for simulating and predicting the performance of parallel resources and batch systems. Research Report 6040, INRIA, 2006.
11. F. W. Glover and G. A. Kochenberger, editors. *Handbook of Metaheuristics*. Kluwer, 2003.
12. X. He, X. Sun, and G. von Laszewski. QoS guided min-min heuristic for Grid task scheduling. *Journal of Computer Science and Technology*, 18(4):442–451, 2003.
13. N. B. Ho and J. C. Tay. Evolving dispatching rules for solving the flexible job-shop problem. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 2848–2855, 2005.
14. D. Klusáček, L. Matyska, and H. Rudová. Local Search for Deadline Driven Grid Scheduling. In *Third Doctoral Workshop on Mathematical and Engineering Methods in Computer Science (MEMICS 2007)*, pages 74–81, 2007.
15. D. Klusáček, L. Matyska, and H. Rudová. Local Search for Grid Scheduling. In *Doctoral Consortium at the International Conference on Automated Planning and Scheduling (ICAPS'07)*, Providence, RI, USA, 2007.
16. W. Kocjan. Dynamic scheduling state of the art report. Technical report, SICS Technical Report T2002:28, 2002.
17. S. G. Kolliopoulos and G. Steiner. On minimizing the total weighted tardiness on a single machine. In V. Diekert and M. Habib, editors, *Symposium on Theoretical Aspects of Computer Science*, volume 2996 of *Lecture Notes in Computer Science*, pages 176–186. Springer, 2004.
18. A. Legrand, L. Marchal, and H. Casanova. Scheduling distributed applications: the SimGrid simulation framework. In *Proceedings of the Third IEEE International Symposium on Cluster Computing and the Grid*, pages 138–145. IEEE Computer Society, 2003.
19. X. Liu, H. Xia, and A. Chien. Validating and scaling the MicroGrid: A scientific instrument for Grid dynamics. *The Journal of Grid Computing*, Volume 2(2):141–161, 2004.
20. R. Loganantharaj and B. Thomas. An overview of a synergetic combination of local search with evolutionary learning to solve optimization problems. In *IEA/AIE '00: Proceedings of the 13th international conference on Industrial and engineering applications of artificial intelligence and expert systems*, pages 129–138. Springer, 2000.
21. M. Mathirajan and A. I. Sivakumar. Minimizing total weighted tardiness on heterogeneous batch processing machines with incompatible job families. *The International Journal of Advanced Manufacturing Technology*, 28:1038–1047, 2006.
22. M. Pinedo. *Planning and Scheduling in Manufacturing and Services*. Springer, 2005.
23. A. Takefusa, S. Matsuoka, K. Aida, H. Nakada, and U. Nagashima. Overview of a performance evaluation system for global computing scheduling algorithms. In *HPDC '99: Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing*, pages 97 – 104, USA, 1999. IEEE.
24. J. M. van den Akker, J. A. Hoogeveen, and J. W. van Kempen. Parallel machine scheduling through column generation: Minimax objective functions. In Y. Azar and T. Erlebach, editors, *European Symposium on Algorithms*, volume 2996 of *Lecture Notes in Computer Science*, pages 648–659. Springer, 2004.
25. G. Verfaillie and N. Jussien. Constraint solving in uncertain and dynamic environments: A survey. *Constraints*, 10(3):253–281, 2005.