Scalable Computing: Practice and Experience
Volume 11, Number 2, pp. 173–188. http://www.scpe.org
ISSN 1895-1767

FAST MULTI-OBJECTIVE RESCHEDULING OF WORKFLOWS TO CONSTRAINED RESOURCES USING HEURISTICS AND MEMETIC EVOLUTION

WILFRIED JAKOB, FLORIAN MÖSER, ALEXANDER QUINTE, KARL-UWE STUCKY AND WOLFGANG SÜß*

Abstract. Scheduling of jobs organized in workflows to a computational grid is a permanent process due to the dynamic nature of the grid and the frequent arrival of new jobs. Thus, a permanent rescheduling of already planned and new jobs must be performed. This paper will continue and extend previous work, which focused on the tuning of our **G**lobal **O**ptimising **R**esource **B**roker and **A**llocator GORBA in a static planning environment. A formal definition of the scheduling problem and a classification will be given. New heuristics for rescheduling based on the "old plan" will be introduced and it will be investigated how they contribute to the overall planning process. As an extension to the work published in [22] a simple local search is added to the basic Evolutionary Algorithm (EA) of GORBA and it is examined, whether and how the resulting Memetic Algorithm improves the results within the limited time frame of three minutes available for planning. Furthermore, the maximal possible load, which can be handled within the given planning time, will be examined for a grid of growing size of up to 7000 grid jobs and 700 resources.

Key words: scheduling, multi-objective optimisation, evolutionary algorithms, memetic algorithms, benchmarks, heuristics, computational grid, restricted resources

1. Introduction. A computational grid can be regarded as a virtualised and distributed computing centre [11]. Users describe their *application jobs*, consisting of one or more elementary *grid jobs*, by workflows, each of which may be regarded a directed acyclic graph defining precedence rules between the grid jobs. The users state which resources like software, data storage, or computing power are needed to fulfil their grid jobs. Resources may need other ones. A software tool, for instance, may require a certain operating system and appropriate computer hardware to run on. This leads to the concept of co-allocation of resources. Furthermore, users will give due dates, cost budgets and may express a preference for cheap or fast execution [20]. For planning, execution times of the grid jobs are needed. In case of entirely new jobs, this can be done by estimations or by the use of prediction systems only. Otherwise, values coming from experience can be used. The grid middleware is expected to support this by providing runtimes and adding them to the workflow for further usage. According to the policy of their owners, resources are offered at different costs depending on e.g. time of day or day of the week and their usage may be restricted to certain times. In addition, heterogeneous resources usually differ in performance as well as cost-performance ratios.

To fulfil the different needs of resource users and providers, the following four objectives are considered: *completion time* and *costs* of each application job measured as fulfilment of user-given limits and averaged, and to meet the demands of resource providers, the *total makespan* of all application jobs and the ratio of *resource utilisation*. Some of these criteria like costs and time are obviously conflicting.

As grid jobs are assumed to require computing time in the magnitude of several minutes at the minimum, a certain but limited time frame for planning is available. A time limit of three minutes was regarded reasonable for planning. All grid jobs, which will be started within this time slot according to the old schedule, are regarded *fixed jobs* and will not become subject of rescheduling.

This paper extends work on rescheduling published at the PPAM 2009 conference [22] by adding two left out benchmark scenarios for the investigation of increasing workload. Furthermore, a simple heuristic search mechanism is introduced and incorporated in the basic EA of GORBA in such a way that it improves generated offspring. Thus, the EA is turned into a Memetic Algorithm (MA), an algorithm class, which has already proven its superiority to pure EA in various domains [25, 23, 15, 16, 29, 2]. The investigation presented in [22] is enhanced in this paper by the assessment of the new Memetic Algorithm and its comparison to the EA results. The main problem of creating a Memetic Algorithm is the definition of suited local searchers for the task on hand. We will come back to this in §3.

In §2 a formal definition of the problem, a classification, and a comparison with other scheduling tasks will be given. The section ends with a discussion of related work. Section 3 will describe the used algorithms, especially the new heuristics and the new local searcher, and give a summary of the work carried out so far. The results of the experiments for assessing the effect of the new rescheduling heuristics will be presented in §4, which will also report about first investigations regarding the maximum possible load for a grid, the size of

*Karlsruhe Institute of Technology (KIT), Institute for Applied Computer Science, Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen, Germany, {wilfried.jakob, florian.moeser, alexander.quinte, uwe.stucky, wolfgang.suess}@kit.edu

which is growing proportionally to the amount of grid jobs. In §5 a summary and outlook to future research will be provided.

2. Problem Definition and Classification. A notation common to scheduling literature [6, 7] is used to facilitate comparisons with other scheduling problems. Given are a set $M = \{M_1, \ldots, M_m\}$ of resources, a set $J = \{J_1, \ldots, J_l\}$ of application jobs, and a set O of grid jobs. The n_i grid jobs of application job J_i are denoted $O_{i1}, \ldots, O_{in_i}$. The following functions are given:

(i) a precedence function $p : O \times O \to \{TRUE, FALSE\}$ for the grid jobs

(ii) an assignment function $\mu : O \to \mathcal{P}(\mathcal{P}(M))$ from grid jobs to resource sets. $\mathcal{P}(M)$ is the power set of M. μ_{ij} is the set of all possible combinations of resources from M, which together are able to perform the grid job O_{ij}

(iii) a function $t : O \times \mathcal{P}(M) \to \mathbb{R}$, which gives for every grid job O_{ij} the time needed for the processing on a resource set $R_{ij} \in \mu_{ij}$

(iv) a cost function, $c : \mathbb{R} \times \mathcal{P}(M) \to \mathbb{R}$, which gives for every time $z \in \mathbb{R}$ the costs per time unit of the given resource set

Optimisation is done by choosing suitable start times $s(O_{ij}) \in \mathbb{R}$ and resource allocations $R_{ij} \in \mu_{ij}$. A solution is valid, if the following two restrictions are met:

1. All grid jobs are planned and resources are allocated exclusively:

$$\begin{gathered}\forall O_{ij} : \exists s(O_{ij}) \in \mathbb{R}, R_{ij} \in \mu_{ij} \colon \forall M_j \in R_{ij} : \\ M_j \text{ is in } [s(O_{ij}); s(O_{ij}) + t(O_{ij}, R_{ij})] \text{ exclusively allocated by } O_{ij}.\end{gathered} \tag{2.1}$$

2. Precedence relations are adhered to:

$$\forall i, j \neq k : p(O_{ij}, O_{ik}) \Rightarrow s(O_{ik}) \geq s(O_{ij}) + t(O_{ij}, R_{ij}) \tag{2.2}$$

A violation of the two following soft constraints is treated by penalty functions in such a way that the amount of time and cost overruns is considered as well as the number of application jobs affected.

1. All application jobs J_i have a cost limit c_i, which must be observed:

$$\forall J_i : c_i \geq \sum_{j=1}^{n_i} \int_{s(O_{ij})}^{s(O_{ij})+t(O_{ij},R_{ij})} c(z, R_{ij}) \mathrm{d}z \tag{2.3}$$

2. All application jobs J_i have due dates d_i, which must be adhered to:

$$\forall J_i : d_i \geq s(O_{in_i}) + t(O_{in_i}, R_{in_i}) \text{ where } O_{in_i} \text{is the last grid job of } J_i \tag{2.4}$$

The fitness calculation is based on the above-mentioned four objectives and a auxiliary objective. It measures the average delay of each non-terminal grid job (i. e. a grid job with no successors) relative to the earliest starting time of its application job and it is aimed at rewarding the earlier completion of non-terminal grid jobs. The idea is to support the process of starting grid jobs earlier, such that the final grid job can be completed earlier in the end, which is recognised by the main objective *completion time*. Lower and upper estimations for costs and processing times are calculated in the first planning stage of GORBA, which will be described in the next section. Except for the utilisation rate, the value $crit_{i,val}$ of every $criterion_i$ is calculated relative to these limits based on the actual value $criterion_{i,act}$ as follows:

$$crit_{i,val} = \frac{criterion_{i,act} - criterion_{i,min}}{criterion_{i,max} - criterion_{i,min}} \tag{2.5}$$

This makes the single values $crit_{i,val}$ independent of the task on hand and results in a percentage-like range. These values are weighted and summed up, which yields the *raw fitness*. To avoid unwanted compensation effects, the criteria are sorted singly or in groups according to priorities. The criteria of the highest priority always contribute to the sum, while the others are added, if all criteria of the next higher priority fulfil a given threshold value. Weights and priorities are based on experience and aimed at reaching a fair compromise between users

and resource providers. The tuning of the suggested adjustment is left to the system administrator. If the two soft constraints are violated, the raw fitness is lowered to the *end fitness* by a multiplication by the corresponding penalty function, each of which delivers a factor between 0 and 1. Otherwise, end fitness and raw fitness are identical.

Generalising, this task contains the *job shop scheduling* problem as a special case. The extensions are co-allocation of heterogeneous and alternative resources of different performances and time-dependent availability and costs, earliest start times and due dates, parallel execution of grid jobs, and more than one objective. As our task includes the job shop problem, it is NP-complete. For this reason and because of the three minutes runtime limit, approximated solutions can be expected only.

A comparable problem could not be found in literature, see e.g. [6] and [7] for a comprehensive presentation of scheduling problems. This corresponds to the results of the literature review found in [32]. There, it is concluded that only few publications deal with multiple objectives in scheduling and, if so, they mostly deal with single machine problems. Within the grid domain some papers dealing with multi-criteria optimisation were published recently. In [35] it is reported that most of them deal with two criteria, like e.g. [10], and that in most cases only one criterion is really optimised while the other serves as a constraint, see e.g. [31, 37]. The approach from [37] uses matrix-like chromosomes, which is probably the reason why they can handle about 30 jobs within one hour only. Kurowski et al. [24] use a modified version of the weighted sum for a real multi-criteria optimisation, but do not handle workflows. The same holds for Xhafa et al. [36], who use two criteria, *makespan* and *average flow time*. They can be regarded as less conflicting as time and costs, which are obviously contradicting goals. Their approach and ours have the following points in common: The usage of a Memetic Algorithm, a structured population, the concept of fast and permanent replanning, and experiments based on workloads and ressource pools of comparable sizes. One difference regarding both, EA and MA, is that we use much more heuristics for seeding the initial population and for resource allocation. In [20] we have shown that heuristic resource selection outperforms its evolutionary counterpart if the runtime for planning is strictly limited. GLEAM uses a structured population based on the diffusion model since the early 90's, see [13, 14, 19, 18]. Regarding the balance between exploration and exploitation it has properties comparable to the cellular approach used by Xhafa et al. But cellular neighbourhoods are harder to control and to adjust (cf. [36]) than the ring structure we use. The latter needs only one parameter to steer the ratio between exploration and exploitation and this is the *neighbourhood size*. The cellular approach allows the genetic information to spread in two dimensions, which requires smaller neighbourhood sizes, as also Xhafa et al. observed in their experiments [36]. Assuming the same populaton size and a comparable slow loss of genotypic diversity during the course of evolution the ring topology allows larger neighbourhoods and thereby the establishment of better elaborated niches of individuals. This and the simpler control ability can be regarded as advantages and are arguments in favour for our continuing usage of the ring based diffusion model introduced by Gorges-Schleuter 20 years ago [13, 14].

Summarising, beside the work of Xhafa et al. [36] we did not find any report about a comparable amount of resources and grid jobs organised in workflows and subject to a global multi-criteria optimisation. Of course, a lot of publications focus on partial aspects of this problem. For instance, the well-known Giffler-Thompson algorithm [12, 28] was extended to the given problem, but surprisingly produced inferior results than our heuristics [20] described below.

3. Algorithms of GORBA and Results from the Tuning Phase. GORBA [21, 20, 22] uses advanced reservations and is based on Globus toolkit 4 at present. It executes a two-stage planning process. In the first stage the data of new application jobs are checked for plausibility and a set of heuristics is applied that immediately delivers first estimations of costs and completion times. These results are also used to seed the start population of the subsequent run of the Evolutionary Algorithm GLEAM (**G**lobal **L**earning **E**volutionary **A**lgorithm and **M**ethod) [4, 5].

Firstly, the old heuristics used for the tuning of GORBA reported in [33, 20] are described, followed by the new ones for rescheduling. This is followed by a short introduction of GLEAM and a description of the local search procedure and the adaptive mechanism of its application as a meme of GLEAM.

3.1. Heuristics for Planning and Replanning. In a first step a sequence of grid jobs is produced by the following three heuristic rules taking the precedence rules into account:

1. *Shortest due time*: grid jobs of the application job with the shortest due time first
2. *Shortest working time of grid job*: grid jobs with the shortest working time first

3. *Shortest working time of application job*: grid jobs of the application job with the shortest working time first

In the next step resources are allocated to the grid jobs using one of the following three resource allocation strategies (**RAS**):

RAS-1: Use the fastest of the earliest available resources for all grid jobs
RAS-2: Use the cheapest of the earliest available resources for all grid jobs
RAS-3: Use RAS-1 or RAS-2 for all grid jobs of an application job according to its time/cost preference

As every RAS is applied to each grid job sequence, nine schedules are generated, from which the best one is selected as result. These heuristics are rather simple and, therefore, more sophisticated ones have been investigated. Among them, such well-known ones like the Giffler Thompson Algorithm [12, 28], Tabu Search [3], Shifting Bottleneck [1, 8], and GRASP (Greedy Randomised Adaptive Search Procedure) [30]. To our surprise, they all performed inferior to our best heuristic, which is based on *shortest working time of application job* [20, 27]. Thus, the first planning phase is still based on the three first heuristics, which can be used for both creating a new schedule and replanning an already started one. In the latter case the information of the old plan is ignored, of course.

3.2. New Replanning Heuristics. The new heuristics tailored to the replanning situation use the grid job sequence of the old plan for grid jobs, which are subject to rescheduling, i. e. all grid jobs which have not already been started or will be started within the three minutes time frame. The new grid jobs are sorted according to one of the three heuristic rules already described and added to the sequence of old jobs, yielding three different sequences. Resources are allocated using the three RAS and again, nine more schedules are generated, but this time based on the old plan. The best of these eighteen schedules is the final result of the first planning stage, while all are used to seed the subsequent EA run of the second stage.

3.3. Evolutionary Search. The EA GLEAM already contains some evolutionary operators designed for combinatorial problems. They are summarised here only, due to the lack of space, and the interested reader is referred to [4, 5]. A chromosome consists of a sequence of segments, containing a sequence of genes, whose structure is defined by their associated problem-configurable *gene types*. The definition of a gene type constitutes its set of real, integer, or Boolean parameters together with their ranges of values. The latter are observed by the standard genetic operators of GLEAM so that explicit constraints of parameters are always met. For the problem in hand there are two basic gene types, the *grid job genes* and the RAS-gene described later. Each grid job gene consists of a grid job identifier (*id* in figure 3.1), represents a grid job, and has no parameters. The gene sequence is important, as it determines the scheduling sequence described later. Apart from the standard mutation, which changes the sequence of genes by simply shifting one of them, GLEAM contains genetic operators for arbitrary movements of gene segments and for the inversion of their internal order. As segment boundaries can be changed by some mutations, the segments form an evolvable meta structure over the chromosomes. Segment boundaries are also used for the standard 1- and n-point crossover operators, which include a genetic repair that ensures that no offspring lacks genes in the end. The evolvable segmentation and the problem-configurable gene types as well as their associated operators among others distinguish GLEAM from most standard EAs.

Figure 3.1 illustrates the coding by an example consisting of two simple application jobs or workflows, one containing five grid jobs and the other two. The grid jobs of all workflows are labeled by distinct identifiers *id*:

$$1 \leq id \leq n = \sum_{l=1}^{l} n_i \tag{3.1}$$

Besides the grid job genes, each chromosome contains a special additional gene for the selection of the RAS, the so called *RAS-gene*. It has one integer parameter, the *ras-id*: $1 \leq \textit{ras-id} \leq 3$. This is the only parameterised gene in this coding and thus, the only one which can undergo parameter mutation (in this case a uniform mutation). The position of the RAS-gene has no significance for the interpretation of the chromosome.

A schedule is constructed from the grid job genes in the sequence of their position within the chromosome as follows:

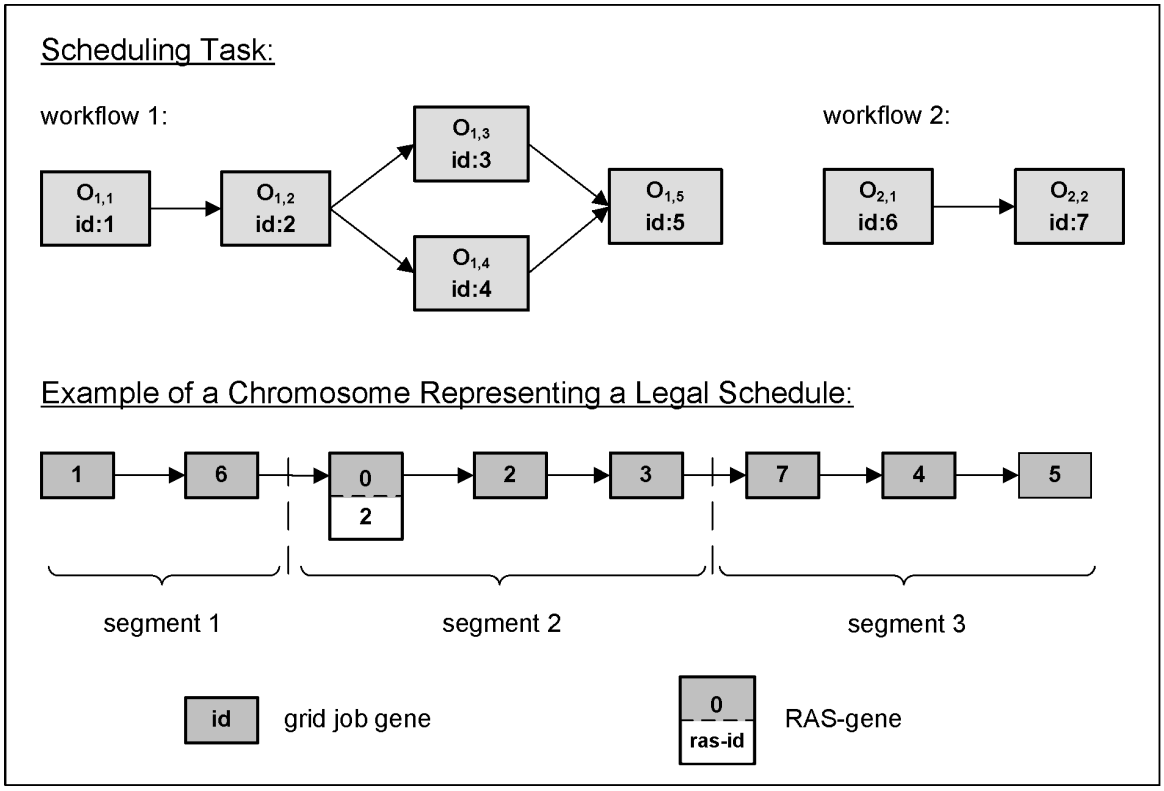

FIG. 3.1. *Example of a scheduling task consisting of two simple workflows and a chromosome representing a legal schedule. It is legal because there is no gene representing a direct predecessor of a grid job, which is located after the gene of that grid job. Thus, all precedence rules will be observed. The example chromosome consists of three segments, the role of which during evolution is described in the text. The parameter part of the RAS-gene is marked by a white background.*

Step 1: The earliest start time of a grid job is either the earliest start time of its application job or the latest end time of its predecessors, if any.
Step 2: According to the RAS selected by the RAS-gene, a list of alternatives is produced for every primary resource.
Step 3: Beginning with the first resources of the lists, the duration of the job is calculated and it is searched for a free time slot for the primary resource and its depending ones, beginning at the earliest start time of step 1. If no suitable slot is found, the resources at the next position of the lists are used.
Step 4: The resources found are allocated to the grid job with the calculated time frame.

The example chromosome shown in figure 3.1 is a legal solution, as all grid jobs are scheduled in such a way that no job is scheduled before its predecessor. In this case the described scheduling mechanism ensures that the precedence rules are adhered to. Alternatively, a phenotypic repair is performed, which is described and assessed in detail in [33, 20]. It retains all grid jobs with missing predecessors from beeing scheduled until their predecessors are scheduled.

3.4. Memetic Search. The basic idea of a Memetic Algorithm is to mimic the learning capability of individuals in nature by an improvement procedure, usually in the form of some local or heuristic search applied to the offspring. This type of enhanced EA is called Memetic Algorithm, a term coined by Moscato [26]. For the problem on hand, the multi-objective nature of the fitness function makes it hard, if not impossible to define a local searcher (LS), for which local assessment is sufficient. This is, because it is hard to decide whether a small local change of e.g. a job sequence is an improvement or not without setting up the complete allocation matrix. For the travelling salesman problem, for instance, which is a common benchmark task for combinatorial optimisation, it is easy to judge whether the swapping of two cities, for example, yields a better result or not. Consequently, we have dropped this desirable property of local assessment and now accept the necessity of elaborating the entire schedule to evaluate a change. The first result is the local searcher described at the end of this section.

```
initialise and evaluate start population
REPEAT                                                    // generational loop
  FOR all individual of the population                    // loop of matings
    choose partner
    FOR all predefined sets of genetic operators          // offspring generation
      generate offspring and evaluate them
      IF best_improvement                                 // best-improvement
        improve best offspring by local search
      ELSE                                                // adaptive all-improvement
        FOR all offspring
          IF offspring is best from evolution OR with probability p_all
            improve offspring by local search
            record effort for adaptation of p_all         // preparation
        calculate and record relative fitness gain for p_all // of adaptation
        IF enough matings                                 // adaptation
          perform adjustments of p_all                    // of p_all
      IF fitness of best offspring is sufficient
        best offspring substitutes parent (individual which chose a partner)
      IF best offspring was locally improved AND
          is accepted for the next generation
        update chromosome according to results of local search
      delete all not accepted offspring
UNTIL termination criterion is satisfied
deliver best individual at minimum as result
```

FIG. 3.2. *Pseudocode of the basic EA GLEAM and its memetic extension (marked by a dark grey background)*

Firstly, the LS is regarded a black box and it will be shown how it is integrated in the EA and what is adjusted adaptively. Figure 3.2 shows both the basic EA and its memetic extension indicated by a dark grey background. Within the generational loop, every individual of the population chooses a partner and produces some offspring as determined by the sets of genetic operators and their predefined probabilities. According to the strategy parameter *best_improvement*, only the best offspring is locally enhanced or its siblings have the chance to undergo local search, too (indicated as *adaptive all-improvement* in figure 3.2). The general adaptation scheme, which is described in detail in [16, 17, 18], controls strategy parameters like termination thresholds or iteration limits of the applied local searchers (LS) as well as their application intensity in case of *adaptive all-improvement*. If more than one LS are used, their selection also is adaptively controlled. The adaptation is based on the costs caused by the local search measured in additional fitness calculations and by the benefit obtained measured in relative fitness gain. The relative measurement of the fitness improvement is motivated by the fact that the same absolute amount may be achieved easily in the beginning of an optimisation run when the overall quality is low and hardly in the end of an optimisation process. For more details about the adaptation see [16, 17, 18].

In this particular case, only one local searcher is applied and it has no strategy parameters to control its search intensity, as will be described later. Thus, figure 3.2 only contains the adaptation of the probability *p_all*, by which the siblings of the best offspring are selected for improvement. When a predefined amount of matings has been processed, the adaptive adjustment of *p_all* is performed based on the observed relative fitness gain and the required evaluations recorded for the matings since the last adjustment. If the quality of the best offspring of a mating is sufficient to substitute its parent and if it was improved by local search, the chromosome is updated according to the LS results. For the details of offspring acceptance and the rationale of chromosome update (Lamarckian evolution), see [4, 5, 18]. The outer loop is performed until a given termination criterion, which is here the time limit of three minutes, is fulfilled.

As mentioned before, the RAS-gene determines the RAS used in the second step of the schedule construction. For crossover operators the RAS-gene of the better parent is inherited to the offspring assuming that it will

perform best for the new child. But what, if not? This question motivates a check-out of all reasonable alternatives in the form of a local search. Reasonable means in this context that alternative RAS are determined for testing based on the actual RAS and the properties of the associated schedule. If, for example, a schedule is too costly and RAS-2 preferring cheap resources was used for its construction, only RAS-3 may perform better. If fast resources were preferred (RAS-1) both other RAS may yield better results. All other situations are treated accordingly.

Due to the fact that this simple local search procedure does not possess any parameters to control its search intensity, the adaptation of the Simple Adaptive Memetic Algorithm (ASMA) is limited to the adjustment of its adaptive all-improvement part. As pointed out in [16, 18], this type of Memetic Algorithm is called simple, because it works with one meme or local searcher only.

3.5. Benchmarks and Results from Earlier Investigations. In a first development phase of GORBA the incorporated algorithms were tested and tuned using our standard benchmark set without co-allocation described in [34]. We use synthetic benchmarks, because it is easier to ensure and steer dissimilarities as described below. The set consists of four classes of application jobs, each class representing different values of the following two characteristics: the degree of dependencies between grid jobs D and the degree of freedom of resource selection R:

$$D = \frac{\sum_{i=1}^{l} \sum_{j=1}^{n_i} p_{ij}}{n(n-1)/2} \tag{3.2}$$

$$R = \frac{\sum_{i=1}^{l} \sum_{j=1}^{n_i} card(\mu_{ij})}{nm} \tag{3.3}$$

where p_{ij} is the number of predecessors of grid job O_{ij}, n is the number of all grid job as defined in (3.1).

The resulting values are in the range between 0 and 1 with small values for few dependencies or a low number of resource alternatives respectively. The benchmarks are constructed in such a way that random values for p_{ij} and $card(\mu_{ij})$ are chosen using a uniform distribution within given ranges. These bounds are adjusted so that either a small degree of about 0.2 or a large one of about 0.8 is achieved. The four combinations of low and high degrees result in four basic benchmark classes, which are abbreviated by *sRsD*, *sRlD*, *lRsD*, and *lRlD*, where *s* stands for small and *l* for large values of R and D.

The duration of grid jobs ranges uniformly distributed between three and seven time units. The same holds for the amount of grid jobs per application job. As the total number of grid jobs is another measure of complexity, benchmarks containing 50, 100, and 200 grid jobs are defined using the same set of resources for every amount. A fourth benchmark set again consists of 200 grid jobs, but with a doubled set of resources available. Time and cost budgets of the application jobs were manually adjusted so that the described heuristics just could not solve them and the exciting question was whether the EA can produce schedules that observe the budgets. The four benchmark classes, together with four different loads, yielded a total of 16 benchmarks [34]. They were used for former investigations, the results of which are summarised at the end of this section.

The benchmarks used for the experiments presented here are based on the described ones with the following peculiarities. In the experiment section four questions are investigated, see §4, which deal firstly with the usability of the old plan for rescheduling and the effectiveness of the new rescheduling heuristics. For these questions the old benchmarks with 100 and 200 grid jobs are used. And secondly, the processible workload and the effect of the Memetic Algorithm compared to the pure evolutionary search is investigated. For that the amount of grid jobs and resources is increased proportionally while preserving the remaining properties of the basic benchmarks based on 200 grid jobs. But in this case, the budgets are no longer adjusted manually to reduce the effort. The new constraints are based on the results of a first heuristic planning with a given percentage of average reduction to make them harder.

The investigations presented here are based on the results reported in [20, 33]. This means that the described coding of the chromosomes is used together with the already mentioned phenotypic repair of possible violations of precedence rules of the grid jobs. The advantage of this approach compared to repair mechanisms on the gene level is that intermediate steps of shifting genes, which themselves may be faulty, are allowed to occur. The idea

is that the resulting schedule is improved after some generations. Furthermore, the well-known OX crossover reported by Davis [9] is added to the set of genetic operators. It preserves the relative order of the parent genes and is aimed at combining of useful gene sequences rather than disturbing them, as (larger) disturbances are provided by the standard crossover operators.

4. Experimental Results for Fast Rescheduling. There are various reasons for rescheduling, of which the introduction of new application jobs is the most likely one. Others are job cancellations or terminations, new resources, resource breakdowns, or changes in the availability or prices of resources. The experiments are based on the most likely scenario of new application jobs and shall answer the following four questions:

1. Does rescheduling benefit from the old plan? If so, to which fraction of finished and new grid jobs?
2. How effective are the old and new heuristics and the subsequent GLEAM run?
3. Up to which amount of grid jobs and resources does the EA or ASMA improve the best heuristically generated schedule?
4. Does the ASMA perform better than the EA and if so, which improvement type is better, *best-improvement* or *adaptive all-improvement*?

As the two benchmark classes based on large degrees of dependencies turned out to be harder than those using small degrees [20, 33], they were used here to answer the questions one, two, and four to limit the effort. In contrast to the work presented at the already mentioned PPAM conference [22], the examination of the third question was now based on all four classes. As pointed out in [20] and [33], the time and cost limits of the application jobs were set so tightly that the heuristics could not solve them without violating these soft constraints. One criterion of the usefulness of the EA run was to find fitting schedules, which was achieved in most, but not all cases. In addition to this criterion, the end fitness values obtained were also compared for the new investigations.

For the experiments reported here, the only EA parameter tuned was the population size varying from 90 to 800 for the runs investigating the first two questions. For the last two ones, smaller population sizes also had to be used, as will be described later on. For every benchmark setting and population size, 50 runs were done, with the results being based on the averages. Confidence intervals and t-tests were used to check the significance of differences at a confidence range of 99%.

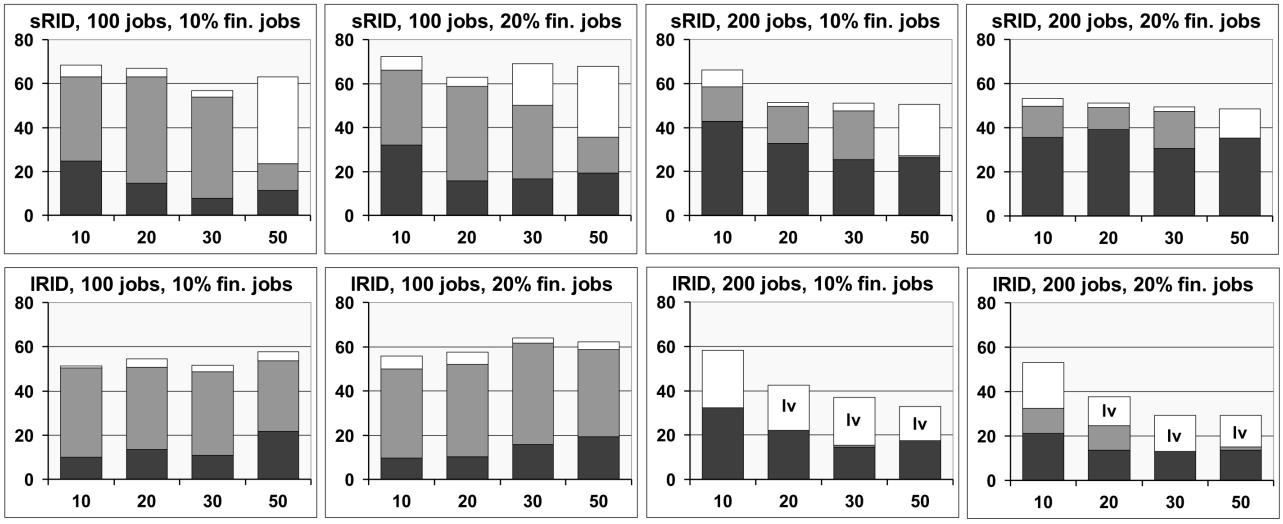

FIG. 4.1. *Comparison of the fitness shares obtained from the basic heuristics (dark grey), rescheduling heuristics (light grey) and GLEAM (white) for all 32 rescheduling settings. X-axis: fraction of new grid jobs in percent relative to the original schedule size, y-axis: normalised end fitness. All GLEAM runs improve the fitness significantly. Even for the smallest improvement of benchmark lRlD, 100 grid jobs, 10% fin. jobs, the best heuristic fitness is clearly outside of the confidence interval of the GLEAM result (three times more than necessary). Abbreviations: fin. jobs: finished grid jobs, lv: limit violations (mostly 1 to 3 application jobs violating the due date), for sRlD and lRlD see previous page.*

For the first two questions, the two basic benchmark scenarios were used for the first planning, with 10 resources and application jobs consisting of 100 and 200 grid jobs, respectively. Eight rescheduling events were compared, which take place when 10 or 20% of the grid jobs are finished and new application jobs with 10, 20,

30, or 50% grid jobs (relating to the original schedule size) are added. This results in 32 benchmark settings. We concentrated on small amounts of already processed grid jobs, because this is more likely in practice and gives a chance for the old schedule to be useful. Otherwise, the situation is coming closer to the already investigated "new planning" situation.

Figure 4.1 compares the results for all 32 benchmark settings. It must be mentioned that a certain variation in the resulting normalised fitness values is not relevant, as the benchmarks are generated with some stochastic variations. Values between 50 and 70 may be considered good results, as the lower and upper bounds for the evaluations are theoretical minima and maxima. Results close to 100% would indicate either a trivial case or a software error. The most important outcome is that for 10% new grid jobs, all eight scenarios perform well. The contribution of the heuristics is clearly situation-dependent and if they tend to yield poor results, GLEAM compensates this in most cases. In other words, if the heuristics can solve the problem well, there is smaller room left for an improvement at all. Another nice result is that this compensation is also done to a certain extent for more new grid jobs, even if the schedules cannot be made free of limit violations (cf. the six columns indicated by *lv*, which stands for limit violations). It can be expected that more new grid jobs will lower the contribution of the replanning heuristics and, in fact, Figure 4.1 confirmes this for the instance of 50% new grid jobs. The case of lRlD, 200, and 10% finished grid jobs is somewhat exceptional, as the replanning heuristics do not work well even in the case of few new jobs.

TABLE 4.1

Comparison of the contributions of all rescheduling heuristics for the different fractions of finished and new grid jobs. The best values of each column are marked dark grey, while values which reach 90% of the best at the minimum are marked light grey. Abbreviations: SDT: shortest due time, SWT: shortest work time, AJ: application job, GJ: grid job, RAS: see §3.

Finished grid jobs:	10%				20%				Ave-
New grid jobs:	10%	20%	30%	50%	10%	20%	30%	50%	rage
RH SDT & RAS-3	0.90	0.96	0.88	0.92	0.86	0.83	0.89	0.92	0.90
RH SDT & RAS-2	0.70	0.44	0.73	0.80	0.64	0.59	0.75	0.61	0.66
RH SDT & RAS-1	0.48	0.44	0.54	0.45	0.59	0.22	0.53	0.45	0.46
RH SWT (GJ) & RAS-3	0.97	0.86	0.81	0.69	0.94	0.78	0.81	0.62	0.81
RH SWT (GJ) & RAS-2	0.74	0.42	0.63	0.53	0.66	0.50	0.68	0.39	0.57
RH SWT (GJ) & RAS-1	0.47	0.41	0.46	0.28	0.57	0.24	0.54	0.26	0.40
RH SWT (AJ) & RAS-3	0.90	0.88	0.82	0.70	0.86	0.83	0.77	0.70	0.81
RH SWT (AJ) & RAS-2	0.70	0.41	0.70	0.56	0.64	0.51	0.57	0.46	0.57
RH SWT (AJ) & RAS-1	0.48	0.44	0.57	0.31	0.59	0.24	0.49	0.43	0.44
SDT & RAS-3	0.45	0.42	0.35	0.56	0.45	0.41	0.47	0.51	0.45
SDT & RAS-2	0.58	0.52	0.38	0.72	0.51	0.43	0.56	0.69	0.55
SDT & RAS-1	0.42	0.42	0.39	0.54	0.39	0.37	0.37	0.45	0.42
SWT (GJ) & RAS-3	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
SWT (GJ) & RAS-2	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
SWT (GJ) & RAS-1	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
SWT (AJ) & RAS-3	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
SWT (AJ) & RAS-2	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02
SWT (AJ) & RAS-1	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02	0.02

Table 4.1 illustrates the contribution of each heuristic. For each of the 32 benchmark settings, the fitness of each heuristic is calculated relative to the best heuristic result for this setting. The four values for both grid job amounts for *sRlD* and *lRlD* are averaged and shown in the table. The right column again averages the values for the different finished and new grid job fractions. The old heuristics based on short working times in the lower half of the table show the same poor behaviour as for planning an empty grid [20], but when taken as a basis for the new rescheduling heuristics, they contribute quite well. According to the table, RAS-3 performs best, but the raw material not shown here has thirteen cases, in which the two other RAS are the best ones. Thus, it is meaningful to use them all for replanning, but it seems to be wise to drop the six old heuristics sorting according to short working times.

To investigate the last two questions concerning the processible load, the rescheduling scenario with 10% finished and 10% new grid jobs is used with proportionally growing numbers for grid jobs and resources. The studies on the behaviour of the EA GLEAM are based on all benchmark classes, while the ASMA is checked with sRlD and lRlD only to limit the effort. The penalty functions showed a weakness when applied to an increasing load, with the penalty factor converging to one for small numbers of penalty causing application jobs. Thus, the two functions penalising the number of too costly or late application jobs were modified to yield a value 0.8 instead of near 1.0 in case of just one job. 0.8 was chosen to have a distinct separation from penalty-free schedules with weaker overall quality. It must be borne in mind that a suitable definition of the fitness function is crucial to the success of every evolutionary search. To achieve comparability, all runs were done with the modified fitness function, so that the results reported in [22] differ in the details of the figures from those reported here. But the general tendencies remain the same as expected.

The comparisons are based on the fitness improvement obtained by the EA compared to the best heuristic result and on the success rate, which is the ratio between violation-free and total runs per benchmark setting. Figure 4.2 shows the results. As expected, success rate and EA improvement decrease with growing load. For large resource alternatives, good success rates can be preserved much longer. This is not surprising, as more resources mean more planning freedom. The relatively large variations of the EA improvement can be explained by the varying ability of the heuristics to produce schedules with more or less limit violations. As limit violations are penalised severely, a small difference already can produce a relatively large fitness alteration.

No violations at all leave little room for improvements like in the two cases of sRlD and lRsD and 200 grid jobs. There is a clear correspondence between the success and the magnitude of the confidence intervals. This is because the penalty function reduces the fitness significantly. For loads of up to 400 grid jobs and 40 resources, GLEAM can always produce violation-free schedules. For up to 1000 grid jobs and 100 resources, this can be achieved for the cases of large resource alternatives. From that on to 1600 grid jobs and 160 resources, the chance of producing successful schedules is slowly diminishing. But improvements of the overall quality and below the level of violation-free schedules are still possible for loads of up to 7000 grid jobs and 700 resources, as shown in the lower part of Figure 4.2. In other words, the amount of application jobs keeping the budgets is still increased compared to the heuristic results.

The fourth question concerns the effect of adding local search to the evolutionary process. The results for sRlD and lRlD exhibit better success rates for an increasing load, as shown in Figures 4.3 and 4.4. For sRlD both improvement types of the ASMA yield a success for 600 grid jobs and 60 resources, which could not be achieved by GLEAM alone. In case of lRlD, the ASMA runs with adaptive all-improvement produce the better results for the three loads between 1200 and 1600 grid jobs. Looking at the improvement rate, it is striking that there are only minor differences, which in most cases are not significant, for all completely or nearly successful runs. Again, the ASMA with adaptive all-improvement works better for the lRlD benchmarks in all cases, but one (5000 grid jobs) and in many cases of the other benchmark series. As there are four sRlD cases and one lRlD case, where GLEAM performs best (900, 1000, 1200, and 2400 grid jobs), more investigations are necessary. For results like sRlD and 3200 or 5000 grid jobs, the t-test shows that the differences are not significant. In summary, there is a clear tendency in favour of the ASMA variant using adaptive all improvement, which is based mainly on the achieved improvements of the success rates.

Finally, the questions for how long improvements can be expected and which population sizes are good shall be studied. The more grid jobs must be planned, the less evaluations can be processed within the three minutes time frame. Figure 4.5 shows that this amount decreases continuously with growing load. The more resource alternatives are available, the more must be checked by the RAS, which lasts longer and explains the lower numbers for the two lR cases. In the long run, the evaluations possible decrease to such an extent that the population size must be reduced to 20 or 30 to obtain two dozens of generations at least in the end. It is obvious that with such small numbers, only poor results can be expected and it is encouraging that even with the largest load there still is a significant improvement. Hence, a faster implementation of the evaluation software or better hardware will enlarge the possible load or deliver better results for the loads investigated.

As mentioned above, the only tuned strategy parameter of the EA and the ASMA is the population size. For GLEAM, sizes between 20 and 400 were observed, if the exceptional cases of 10 and 800 are left out. The values are a little bit lower for the ASMA using best-improvement and about half (20 to 200) for adaptive all-improvement. This means that the range of potential good population sizes is significantly smaller for the ASMA based on adaptive all-improvement. This is due to the more intensive search done when more siblings of a mating are locally improved.

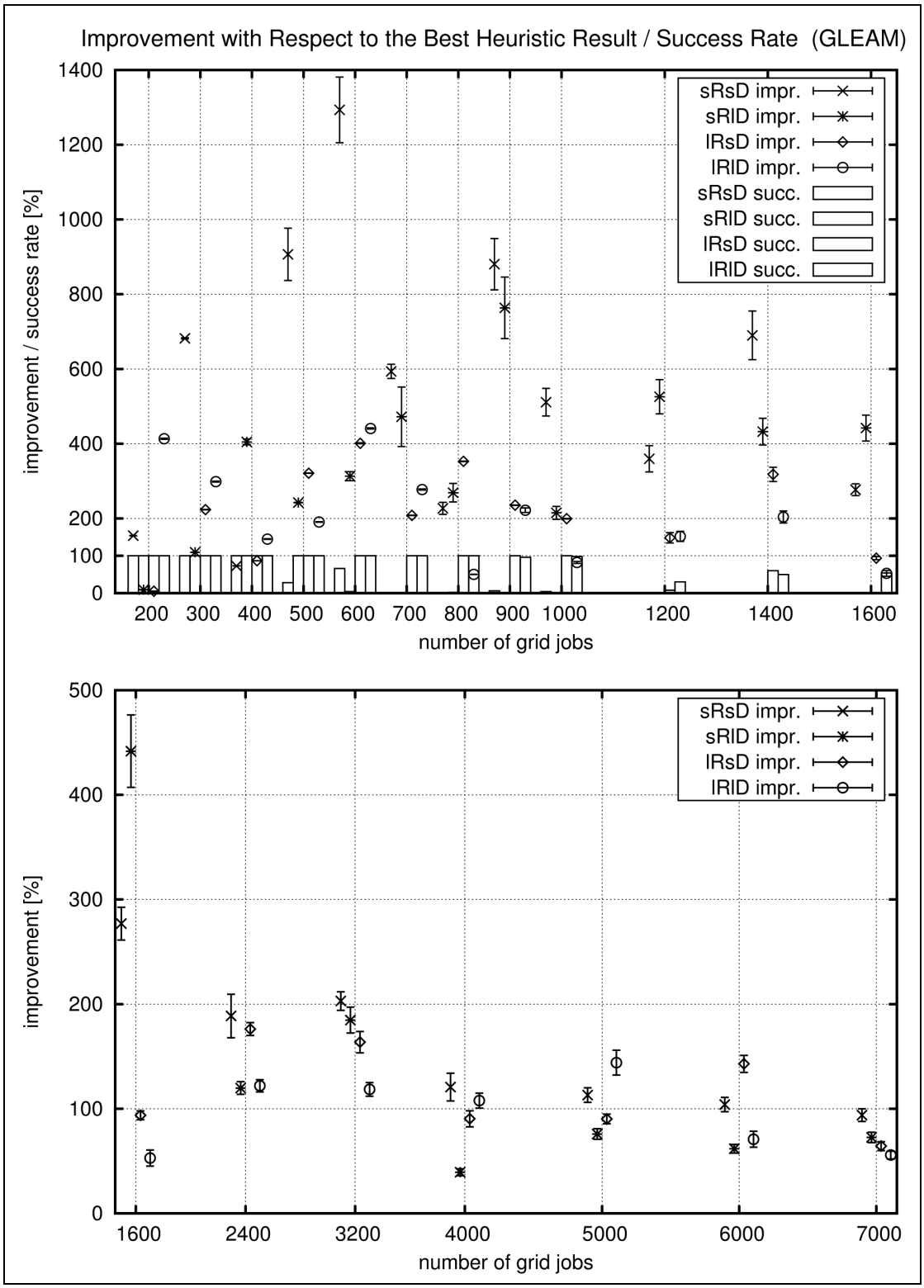

FIG. 4.2. *Success rate and EA improvement compared to the best heuristic at increasing load for all four benchmark classes and 10% finished and new grid jobs. The improvement factors are given with their confidence intervals. The bars of the success rates are not shaded, so that the marks of low success rates remain visible. They are always located in the same column as the corresponding success rate. The number of resources is in each case 10% of the number of grid jobs. For a better visibility the results for 2400 grid jobs and more are plotted in a separate diagram, where success rates already shown in the upper part are left out.*

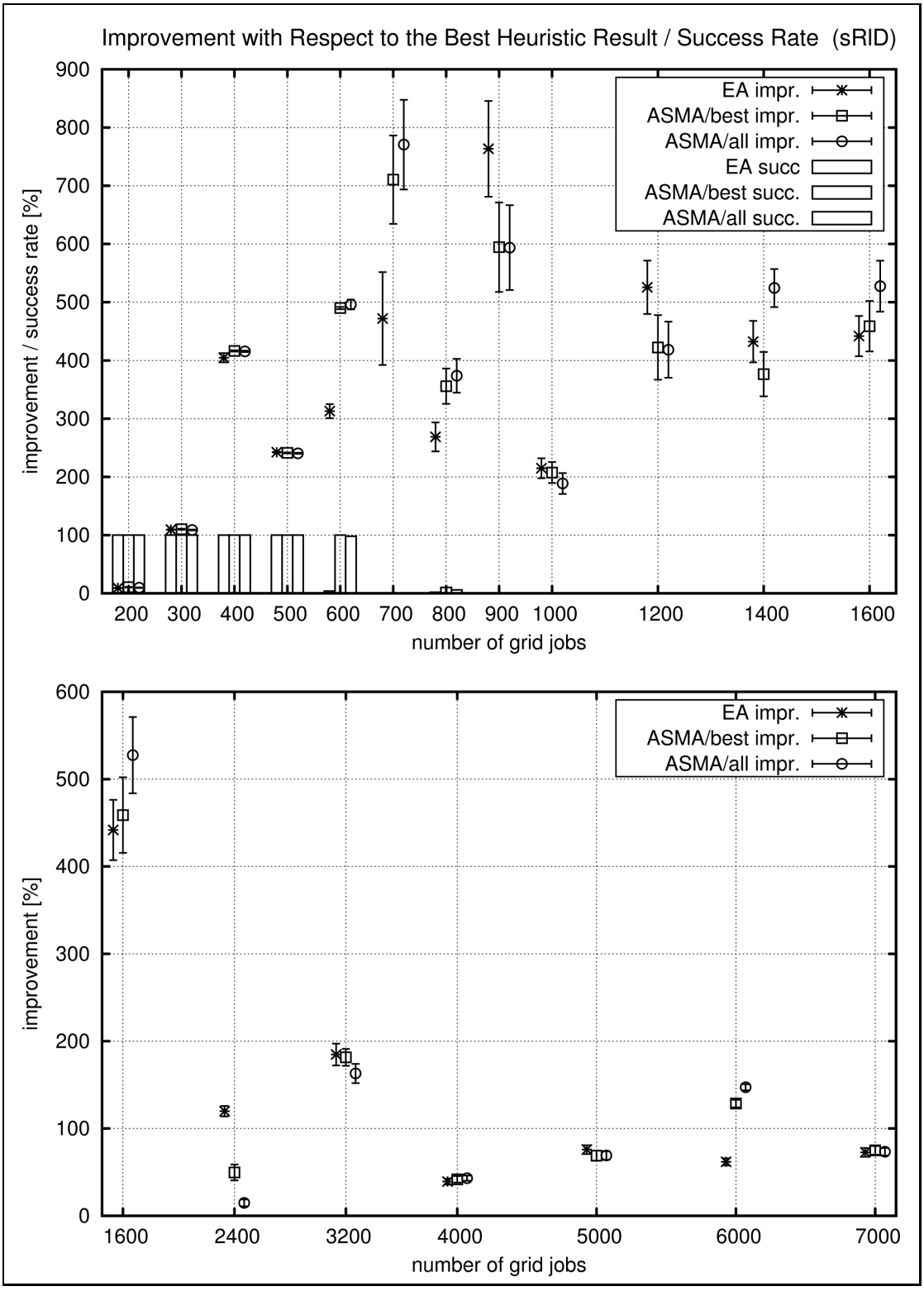

FIG. 4.3. *Success rate and EA improvement as described in Figure 4.2 for the sRlD benchmark series comparing GLEAM with the two ASMA variants described in §3.4.*

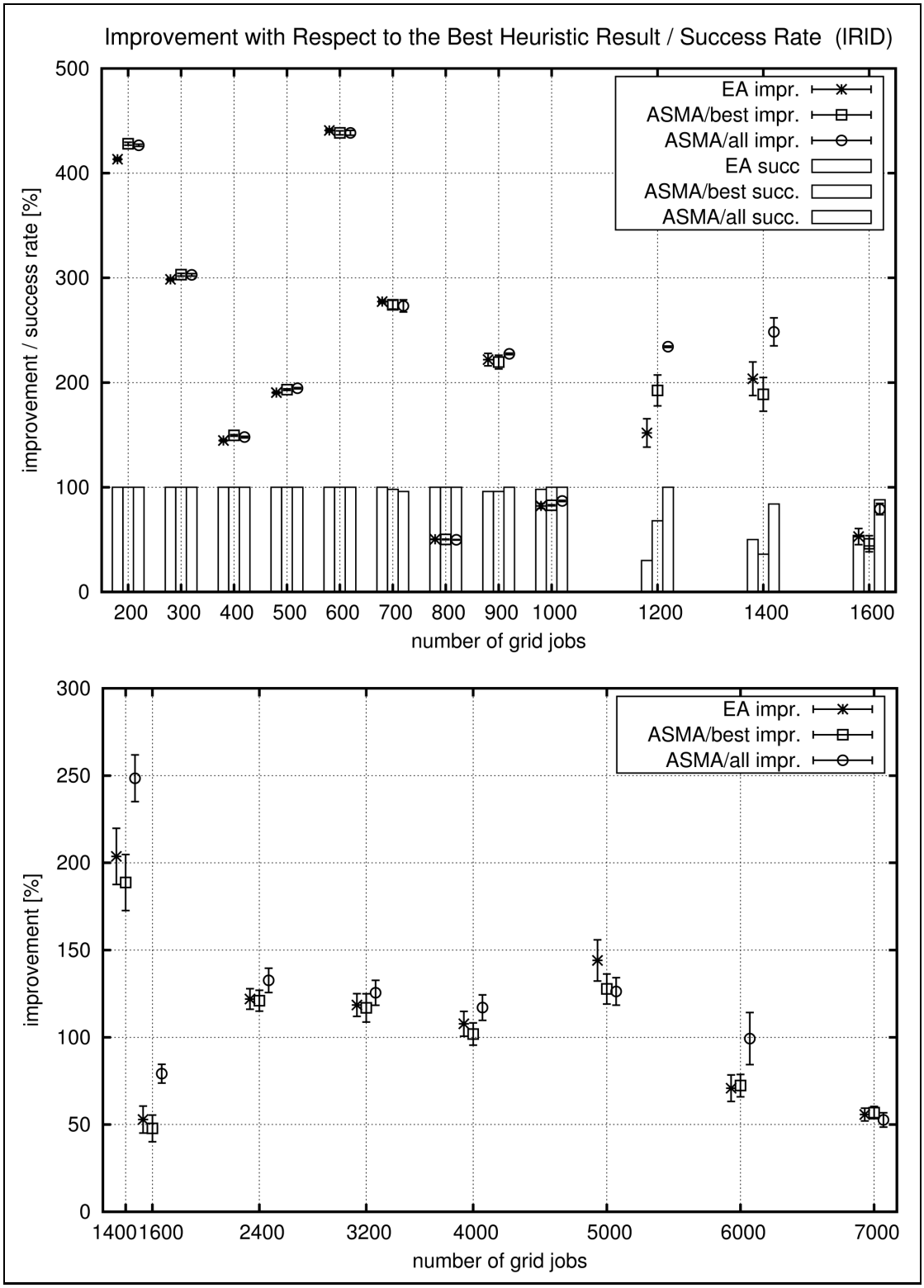

FIG. 4.4. *Success rate and EA improvement as described in Figure 4.2 for the lRlD benchmark series comparing GLEAM with the two ASMA variants described in §3.4.*

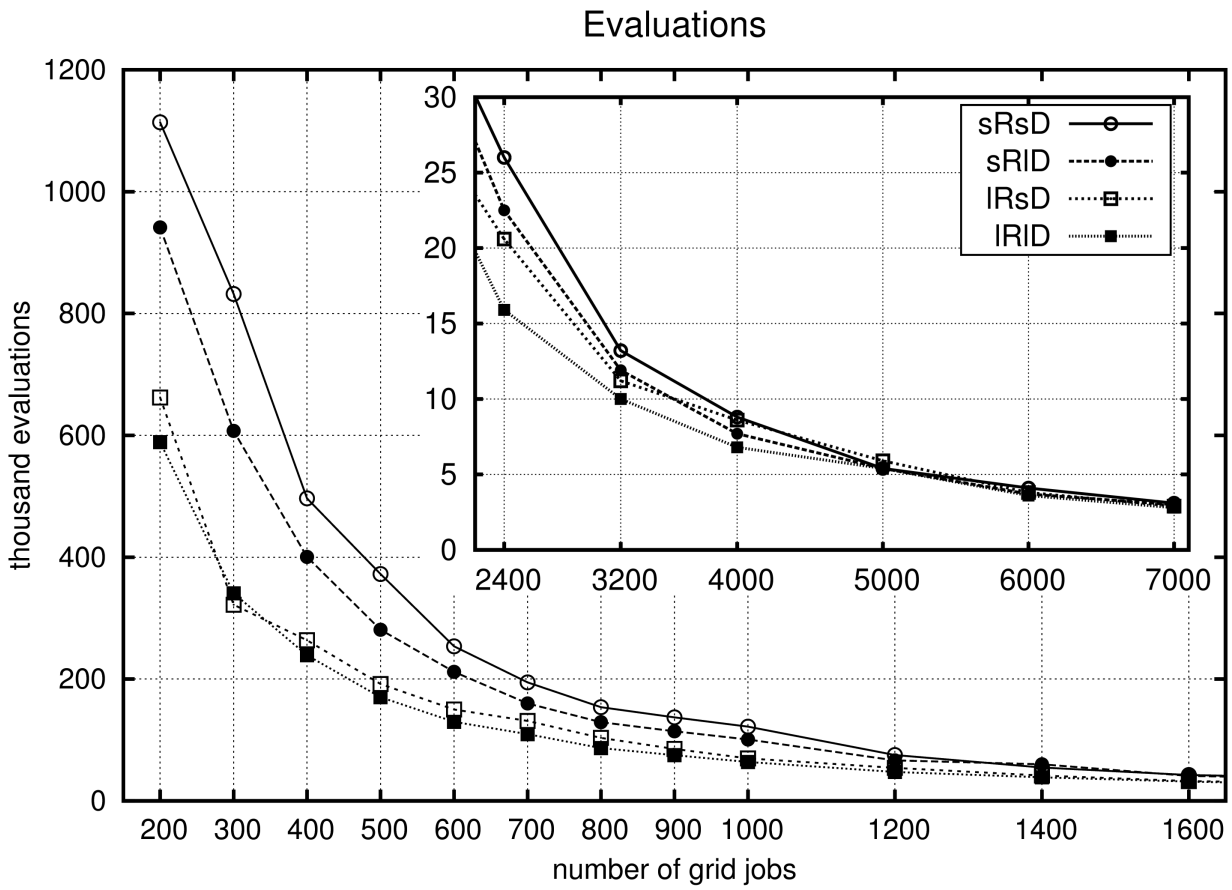

FIG. 4.5. *Evaluations possible within the three minutes time frame at increasing load. The number of resources is in each case 10% of the number of grid jobs. For a better visibility, the diagram for 2400 and more grid jobs is shown separately.*

5. Conclusion and Future Work. It was shown that the problem of scheduling grid jobs to resources based on realistic assumptions and taking the demands of resource users and providers into account is a much more complex task than just *job shop scheduling*, which, in fact, is complex enough, as it is NP-complete. The task on hand enlarges the classical problem by alternative and heterogeneous resources, co-allocation, and last, but not least by multi-objective optimisation. A local searcher was presented, which checks reasonable alternatives of the used resource allocation strategy (RAS) and it was shown how it was integrated in the Evolutionary Algorithm GLEAM. The resulting adaptive simple Memetic Algorithm (ASMA) was assessed using two of the four benchmark classes introduced. The promising results encourage us to implement and examine three more local searchers, which shift genes based on information obtained from the corresponding schedule.

The investigated problems are rescheduling problems, which are the common case in grid resource management. Rescheduling is necessary, if new jobs arrive, planned ones are cancelled, resources break down or new ones are introduced, to mention only the more likely events. For this purpose, new heuristics that exploit the information contained in the "old plan" were introduced. It was shown that the solution for the common case of smaller changes, i. e. in the range of up to 20% finished and new grid jobs, could be improved significantly.

The processible work load was also investigated for 10% finished and new grid jobs at an increasing number of jobs and resources. It was found that for loads of up to 7000 grid jobs and 700 resources, it was still possible to gain an improvement by the EA run, even if it is comparably small. As with this load only 27 generations and population sizes in the range of 20 or 30 are possible within the three minutes time frame for rescheduling, not many greater loads can be expected to be processible. A faster implementation of the evaluation software or better hardware could help. On the other hand, single resources only have been considered up to now. In

the future we want to extend the system to resources with capacities like clusters of homogeneous hardware or data storage resources.

REFERENCES

[1] J. Adams, E. Balas, and D. Zawack, *The shifting bottleneck procedure for job shop scheduling*, MANAGEMENT SCIENCE, 34 (1988).

[2] M. B. Bader-El-Den, R. Poli, and S. Fatima, *Evolving timetabling heuristics using a grammar-based genetic programming hyper-heuristic framework*, Memetic Computing, 1 (2009), pp. 205–219.

[3] A. Baykasoğlu, L. Özbakir, and T. Dereli, *Multiple dispatching rule based heuristic for multi-objective scheduling of job shops using tabu search*, in Proceedings of MIM 2002: 5th Int. Conf. on Managing Innovations in Manufacturing (MIM), Milwaukee, Wisconsin, USA, 2002, pp. 396–402.

[4] C. Blume and W. Jakob, *Gleam - an evolutionary algorithm for planning and control based on evolution strategy*, in GECCO 2002, E. Cantú-Paz, ed., vol. LBP, 2002, pp. 31–38.

[5] ———, *GLEAM - General Learning Evolutionary Algorithm and Method : ein Evolutionärer Algorithmus und seine Anwendungen*, vol. 32 of Schriftenreihe des Instituts für Angewandte Informatik - Automatisierungstechnik, (in German), KIT Scientific Publishing, Karlsruhe, 2009.

[6] P. Brucker, *Scheduling Algorithms*, Springer, Berlin Heidelberg, 2004.

[7] ———, *Complex Scheduling*, Springer, Berlin Heidelberg, 2006.

[8] K.-P. Chen, M. S. Lee, P. S. Pulat, and S. A. Moses, *The shifting bottleneck procedure for job-shops with parallel machines*, Int. Journal of Industrial and Systems Engineering 2006, 1 (2006), pp. 244–262.

[9] L. D. Davis and M. Mitchell, *Handbook of genetic algorithms*, Van Nostrand Reinhold, (1991).

[10] P.-F. Dutot, L. Eyraud, G. Mounié, and D. Trystram, *Bi-criteria algorithm for scheduling jobs on cluster platforms*, in SPAA '04: Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures, New York, NY, USA, 2004, ACM, pp. 125–132.

[11] I. Foster, C. Kesselman, and S. Tuecke, *The anatomy of the grid-enabling scalable virtual organizations*, International Journal of Supercomputer Applications, 15 (2001), pp. 200–222.

[12] B. Giffler and G. L. Thompson, *Algorithms for solving production scheduling problems*, Operations Research, 8 (1960), pp. 487–503.

[13] M. Gorges-Schleuter, *Genetic Algorithms and Population Structures—A Massively Parallel Algorithm*, PhD thesis, University of Dortmund, FRG, 1990.

[14] ———, *Explicit parallelism of genetic algorithms through population structures*, in PPSN I (1990), H.-P. Schwefel and R. Männer, eds., LNCS 496, Springer, 1991, pp. 150–159.

[15] W. E. Hart, N. Krasnogor, and J. Smith, eds., *Recent Advances in Memetic Algorithms. Studies in Fuzziness and Soft Computing*, vol. 166, Springer, 2004.

[16] W. Jakob, *Towards an adaptive multimeme algorithm for parameter optimisation suiting the engineers' need*, in PPSN IX, T. P. Runarsson, H.-G. Beyer, and J. J. Merelo-Guervos, eds., LNCS 4193, Berlin, 2006, Springer, pp. 132–141.

[17] ———, *A cost-benefit-based adaptation scheme for multimeme algorithms*, in Conf. Proc. PPAM 2007, LNCS 4967, R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Waśniewski, eds., Springer, Berlin, 2008, pp. 509–519.

[18] ———, *A general cost-benefit based adaptation framework for multimeme algorithms*, Memetic Computing, (accepted, to be published).

[19] W. Jakob, M. Gorges-Schleuter, and C. Blume, *Application of genetic algorithms to task planning and learning*, in PPSN II, R. Männer and B. Manderick, eds., North-Holland, Amsterdam, 1992, pp. 291–300.

[20] W. Jakob, A. Quinte, K.-U. Stucky, and W. Süss, *Fast multi-objective scheduling of jobs to constrained resources using a hybrid evolutionary algorithm*, in PPSN X, G. Rudolph, T. Jansen, S. M. Lucas, C. Poloni, and N. Beume, eds., LNCS 5199, Springer, 2008, pp. 1031–1040.

[21] W. Jakob, A. Quinte, W. Süss, and K.-U. Stucky, *Optimised scheduling of grid resources using hybrid evolutionary algorithms*, in Conf. Proc. PPAM 2005, LNCS 3911, R. Wyrzykowski, J. Dongarra, N. Meyer, and J. Waśniewski, eds., Springer, Berlin, 2006, pp. 406–413.

[22] ———, *Fast multi-objective rescheduling of grid jobs by heuristics and evolution*, in Conf. Proc. PPAM 2009, LNCS 6067 or 6068 (to be published in July, 2010), R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Waśniewski, eds., Springer, Berlin, 2010.

[23] N. Krasnogor, B. P. Blackburne, E. K. Burke, and J. D. Hirst, *Multimeme algorithms for protein structure prediction*, in Conf. Proc. PPSN VII, LNCS 2439, 2002, pp. 769–778.

[24] K. Kurowski, J. Nabrzuski, A. Oleksiak, and J. Weglarz, *Scheduling jobs on the grid - multicriteria approach*, Computational Methods in Science and Technology, 12 (2006), pp. 123–138.

[25] M. W. S. Land, *Evolutionary Algorithms with Local Search for Combinatorial Optimization*, PhD thesis, University of California, USA, 1998.

[26] P. Moscato, *On evolution, search, optimization, genetic algorithms and martial arts - towards memetic algorithms*, 1989. Tech. Rep. Caltech Concurrent Computation Program, Rep. 826, California Institute of Technology, Pasadena, CA.

[27] F. Möser, W. Jakob, A. Quinte, K.-U. Stucky, and W. Süss, *An assessment of heuristics for fast scheduling of grid jobs*. Submitted to the ICSOFT 2010 conference.

[28] K. Neumann and M. Morlock, *Operations Research*, Carl Hanser, München, 2002.

[29] Y. S. Ong, M. H. Lim, N. Zhu, and K. W. Wong, *Classification of adaptive memetic algorithms: A comparative study*, IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics, 36 (2006), pp. 141–152.

[30] L. S. Pitsoulis and M. G. C. Resende, *Greedy randomized adaptive search procedures*, in Handbook of Applied Optimization, P. M. Pardalos and M. G. C. Resende, eds., Oxford University Press, 2001, pp. 168–181.

[31] R. Sakellariou, H. Zhao, E. Tsiakkouri, and M. D. Dikaiakos, *Scheduling workflows with budget constraints*, in in Integrated Research in Grid Computing, S. Gorlatch and M. Danelutto, Eds.: CoreGrid series, Springer-Verlag, 2007.

[32] A. Setämaa-Kärkkäinen, K. Miettinen, and J. Vuori, *Best compromise solution for a new multiobjective scheduling problem*, Computers & Computers and Operations Research archive, 33 (2006), pp. 2353–2368.

[33] K.-U. Stucky, W. Jakob, A. Quinte, and W. Süss, *Tackling the grid job planning and resource allocation problem using a hybrid evolutionary algorithm*, in Conf. Proc. PPAM 2007, LNCS 4967, R. Wyrzykowski, J. Dongarra, K. Karczewski, and J. Waśniewski, eds., Springer, Berlin, 2008, pp. 589–599.

[34] W. Süss, A. Quinte, W. Jakob, and K.-U. Stucky, *Construction of benchmarks for comparison of grid resource planning algorithms*, in ICSOFT 2007, Proc. of the Second ICSOFT, Volume PL/DPS/KE/WsMUSE, Barcelona, Spain, July 22-25, 2007, J. Filipe, B. Shishkov, and M. Helfert, eds., Inst. f. Systems and Techn. of Inf., Control and Com., INSTICC Press, 2007, pp. 80–87.

[35] M. Wieczorek, A. Hoheisel, and R. Prodan, *Taxonomy of the multi-criteria grid workflow scheduling problem*, in Grid Middleware and Services - Challenges and Solutions, D. Talia, R. Yahyapour, and W. Ziegler, eds., Springer US, New York, 2008, pp. 237–264.

[36] F. Xhafa, E. Alba, B. Dorronsoro, B. Duran, and A. Abraham, *Efficient batch job scheduling in grids using cellular memetic algorithms*, in Metaheuristics for Scheduling in Distributed Computing Environments, F. Xhafa and A. Abraham, eds., Springer, Berlin, 2008, pp. 273–299.

[37] J. Yu and R. Buyya, *A budget constrained scheduling of workflow applications on utility grids using genetic algorithms*, in Workshop on Workflows in Support of Large-Scale Science, Proceedings of the 15th IEEE International Symposium on High Performance Distributed Computing (HPDC, IEEE, IEEE CS Press, 2006.

Edited by: Marcin Paprzycki
Received: March 30, 2010
Accepted: June 21, 2010