AN ASSESSMENT OF HEURISTICS FOR FAST SCHEDULING OF GRID JOBS

Florian Möser, Wolfgang Süß, Wilfried Jakob, Alexander Quinte and Karl-Uwe Stucky
Institute for Applied Computer Science, Karlsruhe Institute of Technology, P.O. Box 3640, D-76021, Karlsruhe, Germany

Keywords: Scheduling, Benchmarks, Heuristics, Computational grid, Restricted resources.

Abstract: Due to the dynamic nature of the grid and the frequent arrival of new jobs, rescheduling of already planned and new jobs is a permanent process that is in need of good and fast planning algorithms. This paper extends previous work and deals with newly implemented heuristics for our **G**lobal **O**ptimizing **R**esource **B**roker and **A**llocator GORBA. Of a range of possibly usable heuristics, the most promising ones have been chosen for implementation and evaluation. They serve for the following two purposes: Firstly, the heuristics are used to quickly generate feasible schedules. Secondly, these schedules go into the start population of a subsequent run of our Evolutionary Algorithm incorporated in GORBA for improvement. The effect of the selected heuristics is compared to our best simple one used in the first version of GORBA. The investigation is based on two synthetically generated benchmarks representing a load of 300 grid jobs each.
A formal definition of the scheduling problem is given together with an assessment of its complexity. The results of the evaluation underline the described intricacy of the problem, because none of the heuristics performs better than our simple one, although they work well on other presumably easier problems.

1 INTRODUCTION

Planning of jobs and resources of a computational grid is a complex optimization task, if the interests of both resource providers and users are taken into account and the resources are modeled with the necessary detail. This will be described in the second section, where a formal definition of the problem will be given as well. In this paper we report about an investigation of four heuristics which have been selected out of seven well-known ones. They are compared to the best simple method used in our Global Optimizing Resource Broker and Allocator GORBA (Süß et al., 2007; Jakob et al., 2008; Jakob et al., 2009). GORBA pursues a two-step planning and optimization strategy: after a fast heuristic construction of feasible, but suboptimal schedules, the latter are improved by our Evolutionary Algorithm (EA) GLEAM (Global Learning Evolutionary Algorithm and Method) (Blume, 1991; Blume and Jakob, 2009). The motivation was to find better heuristics so that either the subsequent EA run gets a better start population or becomes superfluous. GLEAM is an EA of its own, which includes elements from traditional Evolution Strategies and real-coded Genetic Algorithms. Here it is used to determine the processing sequence of the grid jobs and to choose a resource selection strategy out of three possible ones. The latter are described in section 3.2.2. Usage of a heuristic resource allocation has proven to yield good results faster than a gene model, where both resources and grid job sequence are determined by evolution (Jakob et al., 2008).

In section 3 the choice of the heuristics is explained and the heuristics are described in more detail. Section 4 reports about the results and the benchmarks used. The work is summarized and an outlook is presented in section 5.

As we concentrate on the investigation of four different scheduling heuristics here, the benchmarks, GORBA, and GLEAM are described only very briefly due to the lack of space. The interested reader is referred to (Süß et al., 2007) for a detailed description and explanation of the benchmark construction and to (Jakob et al., 2008; Jakob et al., 2009) for GORBA, where a survey of related work is also given, especially of EA-based scheduling in the grid domain. An up-to-date description of GLEAM can be found in (Blume and Jakob, 2002) and in (Blume and Jakob, 2009), where the algorithm is described in detail and various real-world applications are introduced.

Möeser F., Süß W., Jakob W., Quinte A. and Stucky K. (2010).
AN ASSESSMENT OF HEURISTICS FOR FAST SCHEDULING OF GRID JOBS.
In *Proceedings of the 5th International Conference on Software and Data Technologies*, pages 184-191
DOI: 10.5220/0002919201840191
Copyright © SciTePress

2 PROBLEM DEFINITION

The users of a grid 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 fulfill their grid jobs. Resources may depend on 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 specify due dates and cost budgets and may express a preference for cheap or fast execution. The execution times of the grid jobs, which are necessary for planning, are either estimations or values based on experience. Resources are offered at variable costs depending on e.g. day time or day of the week and their usage may be restricted to certain times according to the policy of their owners. In addition, heterogeneous resources usually differ in performance as well as in cost-performance ratios.

To meet the different needs of resource users and providers, the following four objectives are considered: *completion time* and *costs* of each application job, measured and averaged as fulfillment of user-given limits, and to meet the demands of resource providers, the *total makespan* of all application jobs and the ratio of *resource utilization*. 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. Changing an existing plan is the usual situation, because e.g. new application jobs or resources enter the system more or less frequently. All grid jobs that will be started within this time slot according to the old schedule are regarded *fixed jobs* and will not be subject of (re)scheduling (Jakob et al., 2009).

A notation common to the scheduling literature (Brucker, 2004; Brucker, 2006) is used to facilitate comparisons with other scheduling problems. Given are a set M of resources, a set J of application jobs, and a set O of grid jobs. The n grid jobs of application job J_i are denoted $O_{i1}, ..., O_{in}$. The following functions are given:

- a precedence function $p : O \times O \rightarrow \{TRUE, FALSE\}$ for the grid jobs
- an assignment function $\mu : O \rightarrow \mathcal{P}(\mathcal{P}(M))$ 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}
- a function $t : O \times \mathcal{P}(M) \rightarrow \mathbb{R}$, which describes the time needed to process each grid job O_{ij} on a resource set $R_{ij} \in \mu_{ij}$
- a cost function, $c : \mathbb{R} \times \mathcal{P}(M) \rightarrow \mathbb{R}$, which describes the cost per time unit of the given resource set for each time $z \in \mathbb{R}$

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

1. All grid jobs are planned and resources are allocated exclusively:
$$\forall\, O_{ij} : \exists\, s(O_{ij}) \in \mathbb{R}, R_{ij} \in \mu_{ij} : \forall M_j \in R_{ij} :$$
$$M_j \textit{ is in} [s(O_{ij}); s(O_{ij}) + t(O_{ij}, R_{ij})] \textit{ exclusively allocated by } O_{ij}.$$
2. Precedence relations are adhered to:
$$\forall\, i,\ j \neq k : p(O_{ij}, O_{ik}) \Rightarrow$$
$$s(O_{ik}) \geq s(O_{ij}) + t(O_{ij}, R_{ij})$$

A violation of the two following soft constraints is treated by penalty functions such 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:
$$\forall\, i : c_i \geq \sum_{j=1}^{n_i} \int_{s(O_{ij})}^{s(O_{ij}) + t(O_{ij}, R_{ij})} c(z, R_{ij})\, dz$$
2. All application jobs J_i have due dates d_i, which must be adhered to:
$$\forall\, i : d_i \geq s(O_{in}) + t(O_{in}, R_{in})$$
where O_{in} is the last grid job of J_i.

The fitness calculation is based on the above-mentioned four objectives and an auxiliary objective described in (Jakob et al., 2008). Lower and upper estimations of costs and processing times are calculated in the first planning stage of GORBA described in section 3. Except for the utilization rate the relative value rv_i of each criterion i is calculated based on its actual value $v_{i,act}$ relative to these limits:

$$rv_i = \frac{v_{i,act} - v_{i,min}}{v_{i,max} - v_{i,min}}$$

This makes the single values rv_i 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**, again in a percentage-like range. Due to the nature of the problem, values of 100% or close to 100% are unlikely for nontrivial cases. Values greater than 50% can usually be considered good results. To avoid unwanted compensation effects, the criteria are sorted separately or in groups according to priorities. The criteria of the highest priority always contribute to the sum, whereas the others are added, if all criteria of the next higher priority fulfill a given threshold value. Weights and priorities are based on experience and aimed at reaching a fair compromise between users and resource providers. 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 and raw fitness are identical.

Generalizing, this task contains a special case of the *job shop scheduling problem*. 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 multiple objectives. As our task includes the job shop problem, it is NP-complete. For this reason and because of the three minutes runtime limit, only approximated solutions can be expected.

A comparable problem could not be found in literature, see e.g. (Brucker, 2004; Brucker, 2006) whith a comprehensive presentation of scheduling problems. This corresponds to the results of the literature review found in (Setämaa-Kärkkäinen et al., 2006). There, it is concluded that only few publications deal with multiple objectives in scheduling and, if so, they mostly deal with single machine problems.

3 HEURISTICS

In order to find better heuristics than the currently used ones, a literature study was performed (Möser, 2009). In this study a number of heuristics were considered and some of them were selected for further investigation.

3.1 Considered Heuristics

There are currently four heuristics available in GORBA. These are three simple heuristics (Süß et al., 2007) and an adaptation of the Giffler Thompson Algorithm (**GTA**) (Giffler and Thompson, 1960; Neumann and Morlock, 2002). To our surprise, the adapted version of the GTA neither yielded better results than our best simple heuristic method, nor were the results appropriate seeds for the subsequent EA run (Jakob et al., 2008; Sonnleithner, 2008). The best simple heuristic is the *Shortest Due Date Heuristic* (**SDD**) described in section 3.2.2. As the two others yield only poor results as well, other heuristics were searched for in a subsequent investigation (Möser, 2009). They were evaluated in order to select some of them for implementation and further investigation. The methods were assessed according to statements from literature. The criteria were problem coverage, reliability, adaptability with respect to the problem on hand, speed, quality of results, and possible value for GLEAM. The considered heuristics are appropriate variants of the following ones, of which the first four met the criteria for further investigation:

- List Scheduling
- Shifting Bottleneck
- Taboo Search
- GRASP
- Branch & Bound
- Lagrange Relaxation and Neural Networks
- Simulated Annealing

3.2 Investigated Heuristics

The implemented versions of the heuristics *List Scheduling*, *Shifting Bottleneck*, *Taboo Search* and *GRASP* (Greedy Randomized Adaptive Search Procedure) will be introduced briefly hereinafter. They all assign resources to grid jobs and use selection strategies for scheduling, which are presented first. Also, a short description is given for the SDD which presently is the best performing heuristic in GORBA and hence serves as a reference for assessing the four new scheduling heuristics.

3.2.1 Selection Strategies

Several selection strategies which can be divided into *grid job selection strategies* (**JSS**) and *resource selection strategies* (**RSS**) are used. If it is not clear which grid job should be scheduled next, one of these JSS is applied, if not stated otherwise. The implemented strategies select a grid job according to the following rules:

- *ShortestJob*: grid job with the shortest duration
- *LongestJob*: grid job with the longest duration
- *MostSuccessors*: grid job with the greatest numbers of successors
- *EarliestStart*: grid job with the earliest possible start time

- *LeastRemainingTime*: grid job where the difference between the maximal end time of the corresponding application job and the earliest possible start time of that grid job is minimal

Selecting a resource for a grid job works similarly. The implemented strategies select a resource according to the following rules:

- *Cheapest*: fastest one of all cheapest resources
- *Fastest*: cheapest one of all fastest resources
- *Pref*: "*Cheapest*" or "*Fastest*", depending on the user's preference stored in the application job
- *EarliestStart*: earliest available resource
- *MostRemaining*: minimally used resource

3.2.2 Shortest Due Date Heuristic

This heuristic first creates a grid job sequence by ordering the grid jobs of all application jobs by their due dates, beginning with the application job to be finished first. The resulting job sequence is processed by one of three resource allocation strategies (**RAS**). The different RAS assign resources to grid jobs according to the following rules:

- RAS-1: fastest one of the earliest available resources
- RAS-2: cheapest one of the earliest available resources
- RAS-3: RAS-1 or RAS-2, depending on time/cost preference of the application job

SDD yields three solutions each of which goes into the initial population of the subsequent GLEAM run. The best, of course, is regarded as the preliminary result which should be improved by GLEAM or with which the new heuristics are compared.

3.2.3 List Scheduling

List Scheduling (**LS**) is a simple heuristic that creates a schedule by first putting all grid jobs into a list as described below and afterwards assigning resources to the grid jobs. It is an adaptation of the ideas outlined in (Schuster, 2003). The RSS used is a strategy parameter of this procedure. List Scheduling is deterministic and consists of the following steps:

1. Creation of the list
   First, all grid jobs are put into a list by using a path-based approach. From all application jobs, all possible paths are built from the first to the last grid jobs. As a result, grid jobs are contained in various paths, if the application job has parallel branches. Then, all paths are processed iteratively and for each path the first grid job is removed (it is also removed from all other paths containing that grid job). The removed grid job is added to the list. This is done until all paths are empty. The list now contains the resulting grid job sequence. This method maintains the precedence rules of grid jobs.
2. Scheduling of the list
   Finally, the resulting schedule is built by choosing a resource and a time slot for every grid job using the given RSS.

3.2.4 Shifting Bottleneck

Variants of Shifting Bottleneck (**SB**) for job shop scheduling are described for example in (Adams et al., 1988), in (Dauzère-Pérès and Lasserre, 1993), and for parallel resources in (Chen et al., 2006). Unfortunately, none of them deals with inhomogeneous resources, which is why another approach must be used. The implemented version builds resource groups with associated grid jobs and schedules the grid jobs using the resources of the respective group only. The resulting schedules are ordered by a bottleneck criterion as described below. Using the results of these schedules, the final schedule is created and returned. An RSS is used as a strategy parameter. SB is deterministic and comprises the following steps:

1. Creation of resource groups
   Each grid job has one ore more usable resources. For each grid job a resource group consisting of these resources is built. If a group that is going to be created contains exactly the resources of an already existing one, a reference to that group will be established instead of creating a new one. If a resource group is built, which contains only some resources overlapping with resources from other resource groups, the overlapping resource will be removed from the group with more resources or, if two groups have the same number of resources, from the newer one. In case a resource group is emptied, it is removed and the corresponding grid jobs are associated with the remaining group. After this step, each resource group contains at least one resource and at least one associated grid job.
2. Scheduling of resource groups
   In the next step the grid jobs of each resource group are scheduled using the resources of the respective resource group only. The scheduling order of grid jobs is given by the order in which they were added to the group and the resource for a grid job is chosen by the given RSS. When all grid jobs of a resource group have been processed, they have a temporary start time which will be

used later as an order criterion. The total lateness (or earliness) of each resource group is calculated and the groups are ordered from biggest lateness to biggest earliness.

3. Creating the final schedule
   Finally, all grid jobs are scheduled again, this time using all available resources. The order of scheduling the grid jobs is given firstly by the order of resource groups and secondly by the temporary start times of their grid jobs, beginning from the earliest start time. This means that all grid jobs belonging to the first resource group are scheduled before those of the second resource group and so on. Thus, bigger bottlenecks have more "scheduling freedom", whereas smaller bottlenecks have less freedom due to already allocated resources and time slots. Precedence rules are adhered to. The given RSS is used to select one resource for each grid job.

3.2.5 Taboo Search

Taboo Search (**TS**) is a heuristic with random elements. It creates a schedule in x steps, where x is the number of grid jobs to be scheduled. There is a JSS and an RSS for each step. These chains of strategies are used as the representation of a solution. This approach yields a feasible schedule for each solution and was originally presented by (Baykasoğlu et al., 2002). Another approach is to use the grid job sequence as the representation, as done in GLEAM. To change a solution, the JSS and/or RSS are exchanged as described below. Various solutions are declared "taboo" during the process, as common with TS. They may be used further in special cases only. They are stored in a taboo list which is a First-In-First-Out list of a specific size. Taboo search performs a number of iterations, creating solutions in each iteration. The currently best solution is always saved. The following steps are conducted in the heuristic:

1. Setup
   In this step, the taboo list size tl_{size}, the number of solutions per iteration tl_i, the minimal number of changes of JSS jc_{min} and RSS rc_{min} after each iteration and the likelihood l of accepting a change are set. An initial solution is created by choosing a random JSS and RSS from the pool for each step. The schedule for this solution is computed and evaluated.

2. Updating solutions
   Starting from the initial solution, a number of iterations is performed. In each iteration tl_i solutions are created by exchanging strategy rules from the initial solution with randomly chosen ones from a pool. The number of changes depends on jc_{min} and rc_{min} respectively. The parameters jc_{min}, rc_{min}, and l are adapted during the process such that the chance of a change is increased with the number of unsuccessful iterations (i.e. an iteration that did not improve the currently best solution). If there is a successful iteration, the three parameters will be reset to their original values. In each iteration the schedules for the solutions are created by choosing one of the grid jobs that have no unscheduled predecessor by the current JSS and by selecting a resource for the chosen grid job by the current RSS. The resulting schedule is evaluated using the criteria of cost and makespan only. A solution is better than another one, if its schedule is better than that of the other. The solutions that are worse than the initial one are put into the taboo list. The Pareto optimal solutions from that list are aspirants. The best non-taboo solution is chosen as the new initial one. If no non-taboo solution exists, the best aspirant is chosen instead. If the new initial solution is better than the currently best one, the latter is updated. If a given number of iterations or a given amount of time has passed, the schedule of the currently best solution is returned as the final result.

3.2.6 GRASP

The main idea of GRASP presented in (Pitsoulis and Resende, 2001) has been adapted to the task on hand. The adapted implementation makes use of the same solution representation as taboo search. To create a schedule for a solution, x steps are needed, where x is the total number of grid jobs. Although the representation is the same, the resulting schedule is different, because the methods of solution creation and scheduling used in GRASP are different. An RCL (restricted candidates list) is used to select grid jobs to be scheduled. GRASP is a multi-start heuristic, which means that there are multiple independent iterations. In each iteration one solution is produced. The currently best solution is always saved. GRASP is stochastic and consists of the following steps:

1. Setup
   In this step the maximal size *max* and the tuning parameter α for the RCL are set. A high value of α means more random behavior, whereas a low value means more greediness.

2. GRASP iterations
   A number of GRASP iterations are performed. After a given number of iterations or a given amount of time has passed, the schedule of the currently best solution is returned. Each iteration

works as follows: A solution is created by choosing x JSS and RSS at random from the pool of selection strategies. Afterwards, a schedule is created from that solution. At all times, a list of grid jobs without predecessors exists. Let that list be L_{1st}. In each step, the RCL consists of the first *max* grid jobs from L_{1st}. The RCL is ordered by the criterion reflected by the current JSS. One of the first α% of the grid jobs in the RCL is selected randomly and removed from L_{1st}. All successors of that grid job are added to L_{1st}, if they do not have any other predecessors that are still in that list. The removed grid job is assigned a resource using the current RSS. When L_{1st} is empty, the schedule was created and is evaluated. The currently best solution will be updated, if it is undefined or if the newly created one is better.

4 EXPERIMENTAL RESULTS

In order to compare the new heuristics with our SDD, different benchmark tests have been performed. The setup and the results of these benchmarks are presented below.

4.1 Benchmarks

The benchmarks used are created synthetically which allows for controllable diversity, as opposed to most benchmarks originating from real-world applications, see also (Süß et al., 2007).

The characteristics of *total number of application jobs N*, *total number of grid jobs n*, *variation of grid job number n_{var}*, *average number of resources per grid job R_{avg}*, *variation of resource number R_{var}*, and *average degree of dependencies $d \in [0.0, 1.0]$* are used to categorize a benchmark scenario. There are N application jobs to be scheduled in the scenario. Each application job consists of an average of N/n grid jobs. This number may vary by an amount of n_{var} percent, but the total number of grid jobs is fixed. Each grid job has an average of R_{avg} usable resources. This number may vary by an amount of R_{var}. The degree of dependencies d indicates the parallelism of the workflow or how much the grid jobs depend on each other. For $d = 0.0$ all grid jobs are parallel and for $d = 1.0$ there is no parallelism at all. Table 1 shows the parameters used to create the benchmarks sl300 and ll300. The first letter denotes the freedom of resource selection, the second one the average degree of dependencies. Both can either be s (small) or l (large). A large degree of dependencies was chosen, because such benchmarks were the hardest ones in previous investigations (Jakob et al., 2008; Jakob et al., 2009).

Table 1: Parameters used for creating the benchmark scenarios.

PARAMETER	sl300	ll300
Total no. of application jobs N	60	60
Total no. of grid jobs n	300	300
Variation of grid job no. n_{var}	0.3	0.3
Avg. res. no. per grid job R_{avg}	3	8
Variation of resource number R_{var}	1	2
Avg. degree of dependencies d	0.9	0.9

As in earlier experiments, time and cost budgets were set such that our simple SDD heuristic could not solve them. The following criteria were applied for assessing the heuristics:

- Does the heuristic yield a schedule that adheres to the budgets?
- How do the fitness values compare to the each other?
- Do the results improve the subsequent GLEAM run?

4.2 Results of Heuristic Planning

The results of the heuristic planning step can be seen in Figure 1 (raw fitness) and Figure 2 (end fitness). Neither old nor the new heuristics yielded schedules that adhered to all budgets for the given scenarios. This is also evident from the fitness values. Only SB achieved a better raw fitness than SDD, and only in one of the two scenarios. However, the results of LS and GRASP were also acceptable. TS did not perform well, which had not been expected, because it had performed much better for smaller scenarios. After applying the penalty function, all of the new heuristics dropped to a fitness of below 1%, whereas SDD maintained a fitness of roughly 10%. The reason why TS yields such bad results may be the representation and the method for creating new children from a solution. A great part of it is changing selection rules randomly, which favors smaller scenarios. In scenarios with many grid jobs it is more likely that the exchange of a certain percentage of rules will result in a child that is inferior to the parent. Moreover, TS internally works with a simpler evaluation as described before and the comparison of the results is based on the more comprehensive GLEAM evaluation. GRASP experiences similar problems. However, it does not depend on a single initial solution, but rather creates randomized ones in each iteration. This is beneficial, because new solutions do not inherit bad parts from a parent.

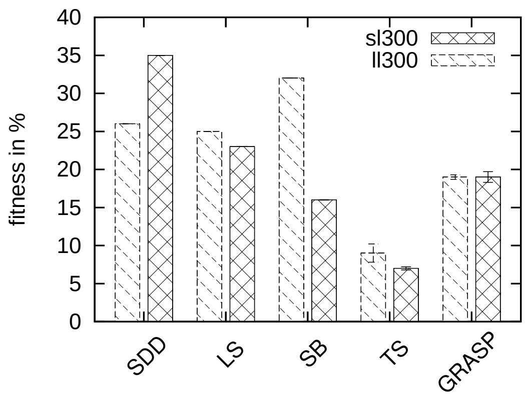

Figure 1: Raw fitness after the heuristic planning step. The 95% confidence intervals are shown for the stochastic heuristics.

4.3 Results of GLEAM Runs

As GLEAM works stochastically, 100 runs were done for each setting. The start populations are either plain randomly seeded or a mixture of random seeds and results from each heuristic. Start populations using heuristic outcomes must be completed by randomly generated (poor) solutions, because there are not enough results and, more important, there must be a great variety to start with for the evolutionary search. More or less homogeneous (start) populations are the death of evolution. The comparisons are based on the success rate and the averaged fitness in conjunction with 95% confidence intervals. The success rate of GLEAM optimization is measured by the percentage of penalty-free runs. A penalty-free run is a run, where all application jobs have been scheduled to meet their budget constraints.

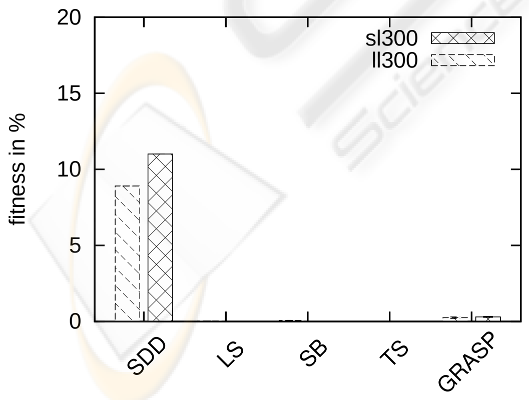

Figure 2: End fitness after the heuristic planning step.

Population sizes from 120 to 300 have been tried out, because the population size yielding the best quality may differ for each scenario and heuristic used for seeding.

The results are shown in Figure 3. If the initial population was created with SDD, GLEAM achieved a success rate of about 50% in either scenario within the given time frame of three minutes. With both the new heuristics and random initialization, GLEAM only achieved a success rate between 0% and 2%. Consequently, there was no improvement. A successful run can be spoiled by exceeding cost or time limits. For scenario ll300, the reason of unsuccessful runs mainly was a violation of the time limit for all tested heuristics. Only up to 10% of the runs were spoiled by exceeding cost limits. For scenario sl300, the number of cost and time limit violations was about equal. Even though violations could not always be removed completely, the schedules produced by the heuristic planning step were improved by GLEAM in all cases by reducing the amount of violations and improving raw fitness.

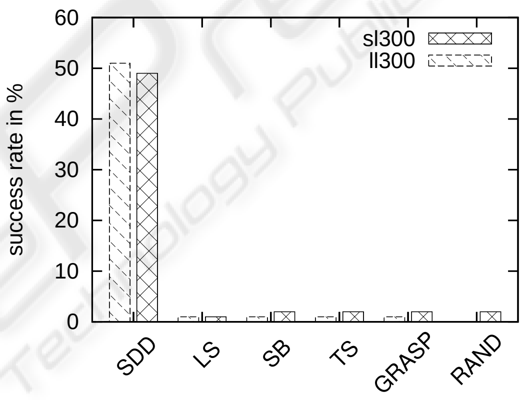

Figure 3: Success rate of the GLEAM optimization.

5 CONCLUSIONS

Although SDD is a very simple heuristic, it yields the best results of all heuristics tested for the given benchmarks. This confirms previous investigations, where similar results were found. The fact that SDD beats all heuristics tested so far underlines its ability to solve the given problem well. Although it is not perfect, the results are by far the best ones obtained during all our investigations.

It must also be noted that the current version of GLEAM performs better with a good heuristic seeding of the initial population than with pure random initialization. Obviously, the raw fitness from heuristic planning is not at all related to the success of GLEAM, the end fitness turns out to be the decisive factor. For scenarios of the given size (amount of jobs and resources) it is essential for the initial population to be as penalty-free as possible, which currently is achieved best by SDD. GLEAM cannot compensate flaws in the initial population within the runtime of three minutes only.

The unexpected poor results of the new heuristics are a motivation for us to change our focus to the identification of suited local searchers that can assist GLEAM. They are intended to improve the offspring of a mating of the Evolutionary Algorithm. This concept of hybridization was applied successfully to optimization tasks from the continuous parameter domain, as reported by (Jakob, 2006).

REFERENCES

Adams, J., Balas, E., and Zawack, D. (1988). The shifting bottleneck procedure for job shop scheduling. *MANAGEMENT SCIENCE*, 34(3).

Baykasoğlu, A., Özbakır, L., and Dereli, T. (2002). 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)*, pages 396–402, Milwaukee, Wisconsin, USA.

Blume, C. (1991). GLEAM - a system for simulated 'intuitive learning'. In Schwefel, H.-P. and Männer, R., editors, *PPSN I (1990)*, LNCS 496, pages 48–54. Springer.

Blume, C. and Jakob, W. (2002). GLEAM - an evolutionary algorithm for planning and control based on evolution strategy. In Cantú-Paz, E., editor, *GECCO 2002*, volume LBP, pages 31–38.

Blume, C. and Jakob, W. (2009). *GLEAM - General Learning Evolutionary Algorithm and Method : ein Evolutionrer Algorithmus und seine Anwendungen*, volume 32 of *Schriftenreihe des AIA*. (in German), KIT Scientific Publishing, Karlsruhe.

Brucker, P. (2004). *Scheduling Algorithms*. Springer, Berlin Heidelberg.

Brucker, P. (2006). *Complex Scheduling*. Springer, Berlin Heidelberg.

Chen, K.-P., Lee, M. S., Pulat, P. S., and Moses, S. A. (2006). The shifting bottleneck procedure for job-shops with parallel machines. *Int. Journal of Industrial and Systems Engineering 2006*, 1(1/2):244–262.

Dauzère-Pérès, S. and Lasserre, J. B. (1993). A modified shifting bottleneck procedure for job-shop scheduling. *Int. Journal of Prod. Research*, 31(4):923–932.

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

Jakob, W. (2006). Towards an adaptive multimeme algorithm for parameter optimisation suiting the engineers' need. In Runarsson, T. P., Beyer, H.-G., and Merelo-Guervos, J. J., editors, *PPSN IX*, LNCS 4193, pages 132–141, Berlin. Springer.

Jakob, W., Quinte, A., Stucky, K.-U., and Süß, W. (2008). Fast multi-objective scheduling of jobs to constrained resources using a hybrid evolutionary algorithm. In Rudolph, G., Jansen, T., Lucas, S. M., Poloni, C., and Beume, N., editors, *PPSN X*, LNCS 5199, pages 1031–1040. Springer.

Jakob, W., Quinte, A., Süß, W., and Stucky, K.-U. (2009). Fast multi-objective rescheduling of grid jobs by heuristics and evolution. In *Conf. Proc. PPAM 2009 (to be published in LNCS)*, Berlin. Springer.

Möser, F. (2009). Integration von Optimierungsalgorithmen in den Grid Resource Broker GORBA. (in German) Bachelor Thesis, DHBW Karlsruhe, Karlsruhe Institute of Technology.

Neumann, K. and Morlock, M. (2002). *Operations Research*. Carl Hanser, München.

Pitsoulis, L. S. and Resende, M. G. C. (2001). Greedy randomized adaptive search procedures. In Pardalos, P. M. and Resende, M. G. C., editors, *Handbook of Applied Optimization*, pages 168–181. Oxford University Press.

Schuster, C. J. (2003). *No-wait Job-Shop Scheduling: Komplexität und Local Search*. PhD thesis, Universität Duisburg-Essen, Duisburg.

Setämaa-Kärkkäinen, A., Miettinen, K., and Vuori, J. (2006). Best compromise solution for a new multiobjective scheduling problem. *Computers & Computers and Operations Research archive*, 33(8):2353–2368.

Sonnleithner, D. (2008). Integration eines Giffler-Thompson-Schedulers in GORBA. (in German), research paper, Faculty of Mechanical Engineering, University of Karlsruhe.

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