UNIVERSITY OF DORTMUND

REIHE COMPUTATIONAL INTELLIGENCE

COLLABORATIVE RESEARCH CENTER 531

Design and Management of Complex Technical Processes and Systems by means of Computational Intelligence Methods

Scheduling Algorithm Development based on Complex Owner Defined Objectives

Carsten Ernemann, Uwe Schwiegelshohn,
Michael Emmerich, Lutz Schönemann
and Nicola Beume

No. CI-190/05

Technical Report ISSN 1433-3325 January 2005

Secretary of the SFB 531 · University of Dortmund · Dept. of Computer Science/XI
44221 Dortmund · Germany

This work is a product of the Collaborative Research Center 531, "Computational Intelligence," at the University of Dortmund and was printed with financial support of the Deutsche Forschungsgemeinschaft.

Scheduling Algorithm Development based on Complex Owner Defined Objectives

Carsten Ernemann[1], Uwe Schwiegelshohn[1], Michael Emmerich[3],
Lutz Schönemann[2], Nicola Beume[2] *

[1] Computer Engineering Institute, University of Dortmund, 44221 Dortmund, Germany
(email: {carsten.ernemann, uwe.schwiegelshohn}@udo.edu)
[2] Dept. of Computer Science XI, University of Dortmund, 44221 Dortmund, Germany
(email: {lutz.schoenemann, nicola.beume}@cs.uni-dortmund.de)
[3] Leiden Institute for Advanced Computer Science, Niels Bohr Weg 1, 2333 CA Leiden, NL
(email: emmerich@liacs.nl)

Abstract. This paper presents a methodology to automatically generate an online scheduling algorithm for a complex objective defined by an owner of parallel identical machines. The objective is based on a combination of simple conventional objectives. Specifically, we address a non-preemptive online scheduling problem on parallel identical machines for a small user community with independent jobs. As the scheduling problem is evaluated using simple conventional objectives for different users that are often contradicting a multiobjective approach is needed. First, Evolutionary Algorithms are applied to create an approximation of the Pareto front in a 7-dimensional space of possible schedules. This enables the machine owner to define a complex objective according to his preferences. For the actual scheduling, a simple Greedy algorithm is proposed whose parameters are determined by an Evolutionary Algorithm such that the selected objective is observed.

1 Introduction

We address the development of online algorithms for scheduling parallel and independent jobs on m identical parallel machines using a complex objective that can be defined by the machine owner. Within the online scenario, different users are submitting jobs $j \in \tau$ over time. Both, the submission time r_j and the execution time p_j of job j are unknown. The users only provide a rough estimate $\bar{p}_j$ for p_j. In practice, this estimate is used to cancel job j if $p_j > \bar{p}_j$, i.e. if the actual execution length of job j exceeds the estimate. In order to make sure that a job without errors is not cancelled just before its completion, users tend to overestimate the execution time [16]. Therefore, $\bar{p}_j$ is of limited help for the purpose of schedule prediction unless users gain a benefit from a correct estimate. Further, job j requires the concurrent and exclusive availability of $m_j \leq m$ machines during the whole execution time. Each machine is used by at most one job at

* All authors, except M. Emmerich, are members of the Collaborative Research Center 531, "Computational Intelligence", at the University of Dortmund with financial support of the Deutsche Forschungsgemeinschaft (DFG).

any time. As the network does not favor any subsets of machines, the execution time p_j of job j is invariant of the subset of m_j machines allocated to j. Also note that m_j is known once job j has been submitted. The completion time of job j in schedule S is denoted by $C_j(S)$. Finally, all jobs are run to completion, that is, we do not allow any preemption [19, 18]. Therefore, job j is started in schedule S at time $C_j(S) - p_j \geq r_j$ and requires a resource consumption (RC) of $p_j \cdot m_j$. We denote by $C_j(S) - r_j$ the flow time or response time of job j in schedule S. Contrary to other online scheduling models, see, e.g. Albers [1], we do not require that machines must be allocated to a job at time r_j. Instead, the scheduler decides just before the start time $C_j(S) - p_j$ which machines are used to execute job j. In practice, this scheduling model is used in many parallel computers [15, 20], like, for instance, the IBM SP2.

Similar scheduling problems were already addressed in the past (e.g. Schwiegelshohn et al. [25]). However, in all those publications, only simple scheduling objectives, like makespan, (weighted) completion time, (weighted) flow time, or utilization were used. With respect to these objectives, either the competitive factor of an online scheduling algorithm is determined [26] or the performance of an algorithm is evaluated with the help of simulations using trace data from real installations [8].

Most real installations of parallel computers have many users (about 200) who are frequently partitioned into user groups. Each job j belongs to one of these user groups denoted by g_j. The machine owner often does not treat all those groups in the same way, but favors some user groups over others. To this end, he defines a so-called owner policy. The preferences of certain user groups is usually implemented by using priorities. Nevertheless, such priority schemes are often not flexible enough, especially in combination with several different objectives. As the above mentioned simple objectives do not consider user groups at all, other methods are applied in order to implement the owner policy. Those methods are often based on quotas. Unfortunately, even the policies of a machine owner are rather vague as the impact of a more complex policy on schedule quality in terms of several objectives has not been studied before. Moreover, quotas themselves are not well suited to fine tune a schedule with respect to an objective.

In the work at hand we allow complex scheduling objectives. This requires both the selection of such a complex objective and the development of a scheduling algorithm that generates good schedules with respect to the selected objective. Here, we assume that the complex objective can be generated by using the simple objectives. However, the resulting complex objective is not a mathematical combination of the underlying simple objectives. These simple objectives may concern the whole schedule, like the total utilization, as well as parts of the schedule, like the average flow time for all jobs belonging to a user group. We define a simple objective $f_i : \mathbb{S} \rightarrow \mathbb{R}^{\geq 0}$ to be a function from the space $\mathbb{S}$ of all legal schedules for a given job system τ to the nonnegative real numbers. It is the goal of the scheduling algorithm to find an optimal schedule $\sigma = \arg\min_{S \in \mathbb{S}}\{f_i(S)\}$. In our case, there are k different objectives $f_1, \ldots, f_k$. Therefore, we speak of a multiobjective or multicriteria scheduling problem [22]. It is clear that usually, those objectives cannot all be met at the same time as the jobs of different user groups compete for the available resources, that is, in general no legal schedule σ exists

with

$$\sigma = \arg\min_{S\in\mathbb{S}}\{f_1(S)\} = \cdots = \arg\min_{S\in\mathbb{S}}\{f_k(S)\}.$$

However, in some cases it is easy to decide that some schedule S_1 is better than some other schedule S_2. This is clearly the case, if schedule S_1 is better in all objectives than schedule S_2. This observation leads to the concept of Pareto optimality. In order to introduce this concept, we first establish the so-called Pareto preference relation $\prec_p$ among schedules:

$$S_1 \prec_p S_2, \text{ iff} \tag{1}$$

$$\forall i \in \{1,\ldots k\}:\ f_i(S_1) \leq f_i(S_2) \quad \wedge \quad \exists i \in \{1,\ldots k\}:\ f_i(S_1) < f_i(S_2).$$

If $S_1 \prec_p S_2$ holds then we say that schedule S_1 dominates schedule S_2. From a decision theoretic point of view, schedule S_1 is always preferable to schedule S_2, iff we have $S_1 \prec_p S_2$. If neither $S_1 \prec_p S_2$ nor $S_2 \prec_p S_1$ hold then a more detailed specification of preferences would be required in order to decide which schedule is the better one.

Given a set of schedules $G \subset \mathbb{S}$, the non-dominated subset of G is defined as

$$nd(G) = \{S \in G | \text{there is no } S' \in G \text{ with } S' \prec_p S\}. \tag{2}$$

In *Pareto optimization*, we are searching for the Pareto set $nd(\mathbb{S})$, that is the non-dominated set for the whole space of legal schedules $\mathbb{S}$. A schedule that belongs to $nd(\mathbb{S})$ is said to be *Pareto-optimal*. The set of objective function values of the Pareto-optimal set is called the *Pareto front*. In our case, the machine owner can choose one Pareto-optimal schedule of this set by taking into account further preferences. Note that it is not trivial to decide in favor of one Pareto optimum if more than 3 objectives are used. To this end, additional support in data representation is required. As this representation problem is not subject of this paper, we simply assume that the machine owner is able to pick the required Pareto optimum or a set of Pareto optima.

However, the Pareto set cannot be computed due to the online character of our scheduling problem. Therefore, we consider a workload trace of the same computer and the same user community and assume that future workloads will exhibit a similar distribution. Unfortunately, the problem of finding an optimal schedule is NP-hard even if all release dates are 0 and we use the makespan objective $f = C_{max} = \max_{j\in\tau}\{C_j\}$ [22]. Therefore, it is not easy to compute an optimal schedule even for a single workload and most simple objectives, particularly as typical workloads comprise more than 200,000 jobs. Trying to find the Pareto set for a workload with conventional methods is not meaningful under these circumstances [22]. Therefore, we use heuristics that find approximations of the Pareto front. In Section 3, these heuristics are described in detail.

Once we know the Pareto front or a good approximation of it, and the choices of the machine owner, we need to generate an online scheduling algorithm that achieves a good schedule for a new set of input data with respect to the selected objective. Our activities are restricted to reordering the set of jobs that have already been submitted but not yet started. Remember that it is not allowed to preempt or cancel a running job.

Here we suggest a Greedy scheduling approach, that is, we will start the first job in the list of the waiting jobs if enough machines are available for allocation. However,

if this is not the case, we may decide to immediately start another job of the list provided enough machines are available for it. This approach is called Backfilling [17] and can frequently be found in schedulers for parallel computers. Many classical scheduling algorithms use priority lists to determine the scheduling order of the jobs. In case of parallel jobs, for instance, sorting in decreasing order of the generalized Smith ratio $\frac{w_j}{m_j \cdot p_j}$ may be the method of choice (Schwiegelshohn [23]). There, each job j is also assigned a weight w_j. In many scheduling papers, this weight is simply an input variable. Some papers require that it is the same as the resource product $m_j \cdot p_j$ thus giving all jobs the same priority [25]. In this paper, the definition of the weight becomes part of an algorithm that tries to generate a schedule following the objective of the user. Note, that in general each possible schedule can be described by one sequence of jobs in combination with a Greedy scheduling approach in the offline scenario. This is not the case within the online scheduling scenario as jobs need to wait for later, not yet visible jobs in order to generate the corresponding offline schedule. This cannot be fulfilled by the chosen Greedy scheduling approach. Therefore, the quality of the generated online scheduling algorithm will be lower in comparison to the offline case. During the determination of appropriate job weights, we apply heuristics to determine the relationship between the input data of a job and the weight. Further, note that the extended Smith ratio also requires the processing time of a job. To this end, we use the user provided estimate of this value even as we are aware of a potentially large overestimation. However, as a smaller estimated execution time favors a faster start of this job, we hope that this will be an incentive for the job owners to provide accurate estimates.

The remaining parts of the paper are organized as follows. After having outlined the general concept in this section, we address several specific problems next. In Section 3, we describe the used multiobjective Evolutionary Algorithms in detail. The evaluation of schedules is subject of Section 4. The paper ends with a conclusion of the main results.

2 Specific Constraints of the Problem

Online scheduling algorithms are frequently evaluated with the help of competitive factors, e.g. [21, 13]. This competitive factor is the upper bound for the ratio between the quality of the schedule generated by the algorithm and the quality of an optimal schedule for the same job set. Please note that the schedule quality is a real positive number with lower values being better. Therefore, competitive factors are based on a worst case approach. Unfortunately, this concept is of limited help for many real applications as the worst case occurs very rarely. In contrast, strategies with a very bad worst case behavior often work well in practice [14, 24].

Similarly, competitive factors cannot be applied in our case as there is currently no applicable description of the structure of a typical workload. In addition, the determination of an optimal schedule for a given workload is computationally expensive. Therefore, we approximate this optimal schedule by using a heuristic. Then we consider the ratio between the quality of the schedule generated by our algorithm and the quality of the approximation of the optimal schedule for each available workload. The final ratio is calculated as the average of all those workload specific ratios.

Figure 1 presents our problem. Here, different users or customers with varying goals are submitting jobs to a system. The sequence of those jobs forms the overall user demand. Then the scheduling system is responsible to allocate the requested machines.

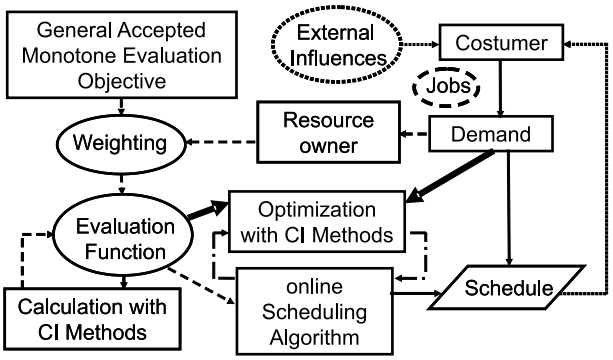

Fig. 1. Principle problem description.

The scheduling algorithm for the assignment of machines to jobs can be specified by the machine owner. The quality of the assignment is determined by a complex objective function. Note, the quality of a single assignment of machines to a job cannot be measured, because an optimal assignment at the moment might result in a poor solution in the future. Therefore, only the assignment of all jobs within one trace can be evaluated at the end of the scheduling process.

A major goal of our work is the development of a robust scheduling algorithm. Robust means that the same algorithm can be applied to various workloads with a similar scheduling behavior. The development process takes this into account since the evaluation of a specific scheduling algorithm incorporates the achieved scheduling results for several workload traces.

The development of the mentioned multiobjective scheduling algorithms is therefore divided into several steps that are shown in Figure 2.

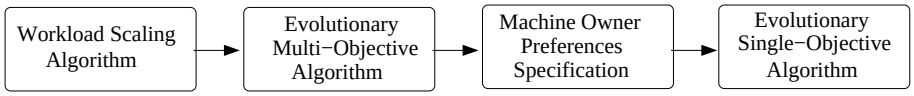

Fig. 2. The development of a multiobjective scheduling system.

As already mentioned, the development of scheduling algorithms in the past was focused on the optimization of simple objective functions. In order to define appropriate scheduling goals, the resource owner needs to know which solutions are achievable.

So the complete space of possible solutions has to be explored by creating the Pareto front of all possible schedules for given workload traces. Multiobjective Evolutionary Algorithms are appropriate for this task. However, these algorithms do not guarantee to produce the true Pareto front. Nevertheless, they have proven to approximate the Pareto front with a high quality.

Using the achieved Pareto front, the owner can define a preferred region within this multidimensional solution space. The definition expresses the machine owner's preferences but does not result in a mathematical combination of the underlying monotone objectives. However, a function that can be used to find the preferred regions with Evolutionary Algorithms can be defined based on the given selection. The selection of the preferred regions by the machine owner is a complicated procedure as we are using seven criteria in this study (see Section 4.1). This procedure is excluded from this paper as the main focus is on the development of the scheduling algorithms. Instead, we use a randomly generated criterion built up from those seven monotone objectives.

Subsequently, an optimized online scheduling algorithm can be developed by parameterizing an existing Greedy scheduling strategy. Such a strategy is represented by a fixed algorithm with a few static parameters. Those static parameters are used to sort new incoming jobs into the queue of waiting jobs. During this process, the static parameters are used to calculate a job weight by combining the static and the job parameters. These job weights are used to sort the jobs. Whenever machines become available, the next job from the waiting queue is scheduled using the well-known First-Come-First-Serve (FCFS) strategy [17].

The used weight function to calculate the job position can be defined by the machine owner. Singleobjective Evolutionary Algorithms are applied in order to optimize the corresponding static parameters regarding the objective function.

Figure 1 further includes external influences and a feedback of the various costumers resulting from the last scheduling decisions. The feedback behavior as well as external influences, e.g. power failures of the parallel computer or certain holidays, are neglected in our work. These influences highly depend on the local environment and therefore are difficult to model. The user feedback is not used as nowadays user models are not able to extract such information.

2.1 Developing an Adapted Scheduling Algorithm

The development of an adapted scheduling algorithm is divided into several steps. First the space of possible scheduling solutions has to be explored, second, the machine owner needs to define the optimization function based on the generated space of possible solutions. This process is followed by adapting an existing scheduling algorithm through the adjustment of a few static parameters.

Exploring the Scheduling Solution Space A normal scheduling procedure itself is not able to generate the Pareto set as the behavior of the scheduler is deterministic and the scheduling result only depends on the given workload trace. Figure 3 shows the simplified scheduling system.

A workload trace from a real installation is used to represent all incoming jobs. Here, job j is moved to the waiting queue when the corresponding submission time r_j

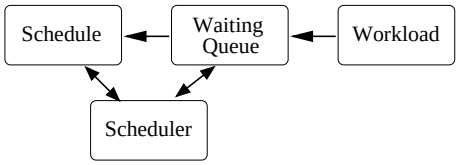

Fig. 3. The figure shows the simplified scheduling system. A workload is used as the simulation input. All jobs that are known to the system but have not been started yet are stored in a waiting queue. A scheduler decides which job of the waiting queue is included in the schedule next.

is reached. Then the First-Come-First-Serve strategy, which will be described in detail later (see page 8), is applied. In order to explore the space of possible scheduling solutions, two different approaches have been examined. First, a reordering of the given workload trace by using Evolutionary Algorithms with a representation to change sequences is applied. Second, a different Evolutionary Algorithm modifies coefficients of a predefined weight function that is used to calculate a weight for each job in the queue of waiting jobs. This queue is reordered according to the weights afterwards. Both approaches will be explained in more detail.

In the first approach Evolutionary Algorithms are used to modify the order of all jobs in the original workload. During this procedure three main constraints must be fulfilled:

- For all jobs $j \in \tau$: j cannot start earlier than its submission time r_j.
- For the new ordering of jobs: a job l can not be started earlier than job j if $j < l$.
- Following the first two constraints, every job is started as soon as possible.

This process will sometimes create big delays in the resulting schedule. However, this might be optimal for the response time of a certain user group and therefore a Pareto optimum within the solution space.

The second approach of changing the order of all jobs within the waiting queue is limited in comparison to the first attempt. Theoretically, every possible sequence of jobs can be created by using the first method. The second method is restricted to local modifications of the job sequence. However, it can be implemented easier and also the corresponding Evolutionary Algorithm is simpler.

In this second approach all situations are divided into three classes. This is the outcome of other studies [7] where significant patterns of workloads are extracted. The three mentioned classes are weekends, week days between 8am and 6pm and week days between 6pm and 8am. Within each of these classes, one of four different mathematical functions is used to calculate job weights that are used during the reordering process. Those four possible functions are:

$$f_1(j) = \varrho_i(j) \cdot w_i \cdot \left(K_i + a \cdot \frac{t_j}{\bar{p}_j} + b \cdot \frac{\bar{p}_j}{m_j} \right) \tag{3}$$

$$f_2(j) = \varrho_i(j) \cdot w_i \cdot (K_i + a \cdot t_j + b \cdot \bar{p}_j \cdot m_j) \tag{4}$$

$$f_3(j) = \varrho_i(j) \cdot w_i \cdot \left(K_i + a \cdot \frac{t_j}{\bar{p}_j \cdot m_j} \right) \tag{5}$$

$$f_4(j) = \varrho_i(j) \cdot w_i \cdot \left(K_i + a \cdot t_j + b \cdot \frac{\bar{p}_j}{m_j} \right) \tag{6}$$

$$\varrho_i(j) = \begin{cases} 1, \text{ if } g_j = i \\ 0, \text{ otherwise.} \end{cases} \tag{7}$$

Here, t_j denotes the actual waiting time of job j. The Evolutionary Algorithms described in Section 3 modify the static parameters w_i, K_i, a, and b. Furthermore, the assignment of mathematical functions to the three different classes is modified. The definitions of the Functions (3) to (6) are based on the adjustment of parameters w_i and K_i depending on the user group i that job j belongs to. This is motivated as the Pareto front represents the different goals of the participating user groups.

Having applied one or both of the described approaches, the overall Pareto set is extracted. Now the administrator of the resources can flexibly define the preferences by combining several parameters within a complex function. This evaluation function can then be used to develop an appropriate scheduling algorithm in a future step.

As the applied scheduling algorithms represent heuristics, good starting solutions are often needed. To this end, we generated initial solutions for the sequencing of the whole workload as well as for the reordering of the waiting queue. The classification of jobs to user groups was used to derive those starting solutions. This is reasonable as some user groups may be more important to the resource owner and, therefore, the corresponding jobs should be processed first.

In detail, we developed a deterministic permutation rule which was statically applied to the original workload trace before the simulation starts or dynamically to the waiting queue during the simulation. In both scenarios, the jobs are ordered according to their corresponding user group. However, we used a more complex permutation as also combined job subsets were used. This will be explained using a small example of three user groups. One permutation is described as a tuple of ordered jobs. The notation (x, y) describes that all jobs from both user groups x and y are taken intersected from the original sequence. So, for the three user groups 13 different permutations can be generated: {1,2,3}, {(1,2),3}, {1,(2,3)}, {(1,2,3)}, {1,3,2}, {(1,3),2}, {2,1,3}, {2,(1,3)}, {(2,3),1} {2,3,1}, {3,1,2}, {3,(1,2)} and {3,2,1}. For five different user groups 540 orderings are possible.

A comprehensive description of the underlying scheduling system that was used during our work will be given in the next paragraph. Note that for each workload trace the same procedure has to be applied to achieve the corresponding results.

Scheduling Algorithm The underlying scheduling strategy is applied in many real-world environments like installations of the IBM SP/2. This FCFS strategy (see [24])

is often combined with Backfilling [17]. For the FCFS strategy all incoming jobs are added at the end of a job queue of all waiting jobs. Jobs within this waiting queue are executed in their submission order. A job can not be started if an earlier submitted job has not been started yet. Each job is executed as soon as possible. So using FCFS no job can overpass any previous job. Backfilling modifies this principle as jobs within the waiting queue can be started earlier than previous jobs if the estimated start times of those previous jobs are not further delayed. The estimation of the start time is calculated by the scheduler using the runtime estimations given by the users. FCFS has a bad worst-case behavior but has proven to work well at real installations. Nevertheless, FCFS and all its variants are only trying to reduce the waiting time as the only scheduling objective. Furthermore, FCFS does not provide any mechanisms for prioritization of jobs or certain user groups. This can only be done by reserving resources for such tasks, which typically reduces the utilization.

Other systems that are able to provide more flexibility than FCFS are mostly rule-based. Only very few of such systems have been developed yet [6, 8]. Based on such rules, market-oriented approaches have been implemented. Those systems provide a very flexible mechanism to find the equilibrium between the supply of resources and the user demand. However, the working principle of such a market environment is that many resources and many users are interacting. This pre-condition is not fulfilled in our scenario as only one parallel computer and a limited number of users exist.

Developing an Adapted Greedy-Scheduling After having generated an approximated Pareto front, the machine owner is able to specify local preferences. Here, a graphical tool is used to select preferred regions within the multidimensional solution space. These preferences are used to construct a fitness function for the following optimization process.

In the next step, this fitness function is used to adapt an existing Greedy scheduling algorithm that uses job weights. Within this process, submitted jobs are inserted into a queue of waiting jobs. The position of the new job within this queue depends on the weight of the job. The following scheduling is based on this ordered queue and tries to start each job as early as possible. The Backfilling strategy introduced above is also applied.

In theoretical literature Greedy scheduling is often called weighted flow time scheduling. However, in such works the job weights are input data. In opposite, our goal is to develop a system which maximize the machine owner's objective by optimizing the job weights. Here, these job weights are static since they are calculated using a deterministic function including all job parameters. Therefore, the weights are independent from the actual schedule. However, as the actual waiting time of jobs can also be used, the dynamic structure of the waiting queue can be included into the scheduling process.

The deterministic Functions (3) to (6) calculate the precise job weights using the static job attributes. The whole set of parameters that optimizes the objective of the machine owner is unknown. Evolutionary Algorithms are used to find good sets of such parameters. In combination with the Greedy scheduling those sets can be used to specify the insertion function. Here, the comparison of parameter sets between different workloads is interesting as we hope to find sets which produce good results for different

workloads, assuming that the machine owners provide similar objectives. The details of the Evolutionary Algorithms are presented in Section 3.

3 Evolutionary Multiobjective Optimization Algorithms

In this section we introduce the *evolutionary multiobjective optimization algorithms (EMOA)* that were evaluated in this paper for generating approximately Pareto-optimal schedules. After some general remarks on the structure and purpose of EMOA, we introduce the selection and variation operators that have been used for the problem at hand. This will be done for two different representations for schedules: First we introduce *evolutionary algorithms (EA)* that work on a direct coding as permutations describing the job sequence. Secondly, we propose an EA working with an indirect, real valued coding, where the real valued optimization variables are the parameters of the job weighting function in a Greedy scheduler, that is then used to generate a job sequence.

3.1 Introduction to EMOA

For reasons of limited time and space it is often not possible to determine the entire Pareto front (see Expressions (1) and (2) on page 3) for a problem and it is justified to search for an approximation of it. This is also the case for the job sequencing problem at hand, because a huge combinatorial search space ($n_{jobs}!$ elements) has to be explored with a very limited number (about 5,000) of black-box evaluations.

Several multiobjective optimization algorithms aim at finding such approximations to the Pareto front. Usually, it is demanded to achieve a good *convergence* as well as a good *coverage* of the Pareto front. A good convergence is achieved if all points in the obtained set are similar or very close – with regard to some distance measure in the solution space – to members of the Pareto set. A good coverage is achieved if the set is well-distributed over the whole Pareto front.

Among the approximation algorithms, EMOA have become very popular. Their flexibility, robustness and their population concept makes them well suited for approximating Pareto sets. EMOA are variants of evolutionary algorithms for multiobjective optimization. Evolutionary algorithms mimic basic principles of biological evolution in order to solve complex problems. Typically, they are applied as randomized search heuristics for solving black-box optimization problems. In particular, they repeatedly apply random variation and selection operators on a set of solution candidates (population). By means of the variation operators (recombination and mutation) the algorithm repetitively generates modified offspring solutions from the current parent solutions. Recombination operators generate an offspring as a mixture of several parents and mutation operators create an offspring as a slight modification of one parent. Single solutions are called individuals for which objective function values are computed. Within the selection operator this function value is interpreted as the individual's fitness. By preferably selecting individuals with a higher fitness, the population gradually moves to regions of the search space in the proximity of optimal solutions, or Pareto-optimal solutions in the presence of multiple objectives.

To put things into more concrete terms, a sketch of a so-called $(\mu \stackrel{+}{,} \lambda)$-EA, a common variant of an EA, is given in Algorithm 1.

Algorithm 1 $(\mu \stackrel{+}{,} \lambda)$-EA

$P_0 \leftarrow \text{evaluate}(\text{init}()), P_0 \in \mathbb{I}^\mu$
$t \leftarrow 0$
while not terminate() **do**
  $Q_t \leftarrow \text{evaluate}(\text{generate}(P_t)), Q_t \in \mathbb{I}^\lambda$ {Generate λ solutions using recombination and/or mutation}
  $P_{t+1} \leftarrow \begin{cases} \text{select}(Q_t) & \text{for } (\mu, \lambda) \text{ selection} \\ \text{select}(P_t \cup Q_t) & \text{for } (\mu + \lambda) \text{ selection} \end{cases}$
  $t \leftarrow t + 1$
end while

The algorithm starts with the initialization and evaluation of a parent population of μ individuals in the individual space $\mathbb{I}$. Then the evolutionary loop is repeated until a termination criterion is fulfilled: First, λ offspring individuals are generated by means of the random variation operators from the current parent population. This is done by employing recombination and/or mutation operators, which can be specific to the problem at hand. Then, the objective function values of all new individuals are evaluated. Based upon the evaluation results, a new population of individuals is selected. In case of a comma strategy the new parents are chosen from the offspring (Q_t) only. The plus strategy takes also the parents into account ($P_t \cup Q_t$). Each loop incorporating the procedures above is called a generation.

In EMOA the selection operator favors non-dominated solutions to dominated solutions and thereby gradually moves the population to the Pareto front of a given problem. When comparing solutions that are mutually non-dominated, the selection operator typically prefers solutions that contribute more to a high diversity of the population in the solution space.

Common variants of EMOA have been discussed and compared on benchmark functions in the book by Deb [4]. An easy to use, but powerful EMOA is the *non-dominated sorting genetic algorithm* (NSGA). Presently, there are two versions of this algorithm, namely NSGA and NSGA-II. These algorithms had been the starting point for the development of an algorithm for finding optimal scheduled job sequences. Hence, they are described in more detail now.

Basically, the idea of the selection scheme of NSGA is to compute a scalar fitness value for each solution based on a dominance rank and its contribution to the population's diversity. Given a set G_t of candidate solutions, in NSGA these solutions are ranked by means of non-dominated sorting and *fitness sharing* in order to decide which of these solutions shall enter the subsequent population P_{t+1}. Non-dominated sorting partitions G_t into several subsets of solutions that are mutually non-dominated. Ranks are assigned to these partitions, meaning that each solution from a partition with a

higher rank is dominated by a solution of every lower ranked partition. Thus, the solutions of partition with rank 1 are non-dominated and the partition with the highest ranking number is the worst. To obtain these partitions, first the non-dominated set N_1 of a set of candidate solutions G_t is determined. To the elements of N_1 the rank 1 is assigned. From the remaining set $G_t \setminus N_1$, again the non-dominated set N_2 is detected and to its members the rank 2 is assigned. Then the non-dominated set from $G_t \setminus (N_1 \cup N_2)$ is determined. This process continues until $G_t \setminus (N_1 \cup N_2 \dots)$ is empty. Now, to each element of the set a rank was assigned, though several individuals might share the same rank. After non-dominated sorting G_t, to each ranking partition a dummy fitness value is assigned. Then for each particular individual its fitness value is slightly altered according to a sharing criterion. The substitute fitness value is shared between similar individuals and therefore rewards solutions that well contribute to the diversity of a partition. The parameter of this sharing term is chosen heuristically [4]. We will not go into detail here, because later on a new version of the NSGA will be introduced that skips the sharing parameter. It is made sure in the NSGA sorting procedure that the substitute value for a member of a partition is by all means worse than the substitute fitness of a partition of lower rank.

Having computed the dummy fitness value for all solutions standard selection methods from EA can be used. In NSGA only the offspring are considered to become parents of the next generation. Because the number offspring was limited, additionally we used a fitness proportional selection scheme for reproduction [2, C2.2]. This means that individuals with higher fitness potentially generate more offspring than individuals with worse fitness. Whereas in NSGA the selection mechanism is a comma strategy, an elitist strategy had often been recommended for combinatorial problems [3, p. 8]. The $(\mu + \lambda)$ selection scheme is elitist, meaning that an all dominating solution would be selected. In addition, it is known from literature that this elitist method allows for a comparable fast approximation of Pareto-optimal points [4].

A new variant of evolutionary algorithms that employs non-dominated sorting, too, is the NSGA-II algorithm. This algorithm applies a plus selection and makes use of a different diversity measure, in order to rank solutions that share the same dominance level (rank assigned by the non-dominated-sorting algorithm). In order to establish a total order among the elements of a partition, *crowding distance* sorting gets employed. By means of this procedure, elements that contribute more to the diversity of a Pareto front are ranked better. In particular, the distance to the nearest solution is determined for each positive and negative coordinate direction in the solution space. Here, the coordinates are always sought as the coordinates of the solution vectors. In order to calculate a scalar value, all these relative distances are summed up and the resulting value is the crowding distance. Extremal solutions that have no neighboring solutions in at least one of the coordinate directions, are always preferred to non-extremal solutions. In contrast to the sharing distance employed in NSGA, the crowding distance does not require the specification of an extra parameter.

In Figure 4 and Figure 5 the sorting procedure of NSGA-II is illustrated for a two-objective minimization problem. Figure 4 displays the three partitions detected by the non-dominated sorting procedure, while Figure 5 displays the ordering of the first par-

tition due to crowding distance sorting. The notches in the figures indicate a gap in the co-domain of the vector valued objective function.

3.2 Problem Representation

EAs can deal with nearly arbitrary representations, as the coding of a problem is called within this context. For a given optimization problem there often exist several approaches to an appropriate representation. In most cases these could be divided in general and more specialized ones. In the general approach a representation is chosen which reflects the given problem in a natural manner. In our situation a schedule is an individual and is coded as a permutation representing the processing sequence of the jobs. For solving the schedule problem with a permutation representation we could fall back on the wide range of standard variation operators for combinatorial problems [2, C3]. This approach works on the whole workload.

The scheduling problem consists of a high problem dimension. To reduce this, an alternative EA with a relatively low dimensional real coded representation was tried. In this approach the EA only determines the parameters of the job weighting function of a Greedy scheduler. The Greedy scheduler then generates a job sequence on basis of these parameters.

Coding as Permutation The design of variation operators was oriented at some general design principles, e.g., that small changes are more probable than great ones and that by a concatenated application of the operator any element in the search space can be reached from any other element. Moreover, variations should be reversible, meaning that a single operation should have the same probability as its inverse operation.

Following these guidelines, for permutation based optimization problems two elementary operations were identified. The first one is the swap of two elements and the other one the movement of one job to another position [2, C3.2.3]. A mutation operator then consists of repetitive applications of these elementary variations. The number of operations that are carried out within one mutation step can be controlled by certain parameters that are part of the individual. A general parameter is the number of elementary operations, called mutation frequency. Moreover, there is a parameter for the distance of jobs that shall be swapped, called mutation jump length. This parameter is used as well for the number of jobs a moved job shall pass. All parameters are initialized externally by user defined values. Within the EA, those parameters are subjected to an evolutionary process as well, which is called self-adaptation. Practically speaking, the mutation operator, with the same probability, multiplies or divides the scaling parameters of an individual with a constant value of $\gamma = 1.3$ in each generation. The modified parameters are then used in the computation of the modified sequence.

The first mutation operator swaps only adjacent elements, which is the smallest possible change of a permutation. This only leads to small changes in the objective function values. To reach noticeable improvements for the given scheduling problem with more than 100,000 elements, a large number of adjacent swaps is necessary.

A second mutation operator swaps two distant randomly chosen elements. The mutation jump length serves as the mean distance between the two elements. This distance

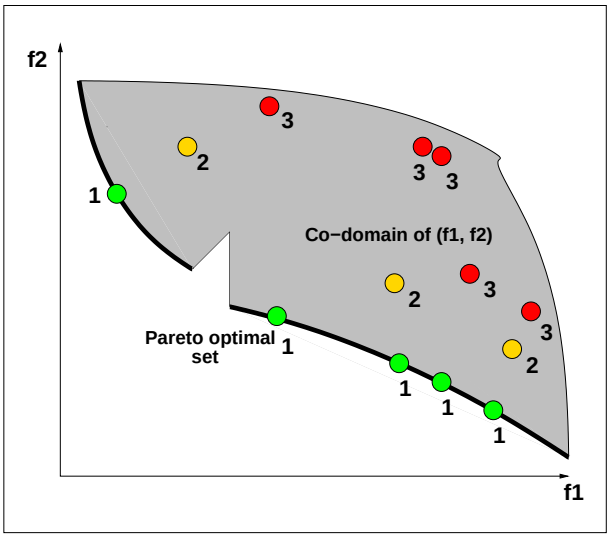

Fig. 4. Illustration of the non-dominated sorting procedure. The numbers attached to the solutions indicate the rank that was assigned to them by the non-dominated sorting.

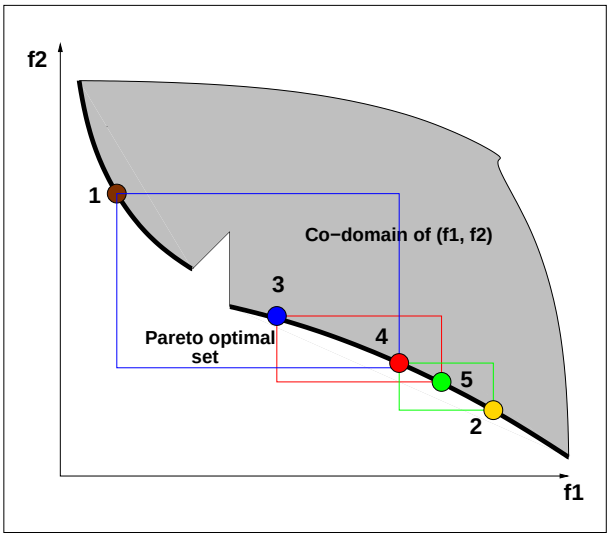

Fig. 5. Illustration of the crowding distance sorting of members on the Pareto front. The circumference of the boxes touching neighboring solutions are the ranking criteria. The numbers assigned to the solutions indicate their rank. Note, that extremal solutions are always preferred to non-extremal solutions.

is chosen as a geometrically distributed random number. The swap operator allows for greater changes with fewer mutations.

In a problem-specific implementation of the mutation operator a job could only move towards the end. Such a job must follow some rules, concerning some job properties. For example, the jobs need to have a minimal number of processors as well as a minimal approximated run time. This means that only "big" jobs are moved, which is assumed to have greater impacts on the results.

A new design of the mutation operator was chosen as an alternative, that will be described next. In an outer loop every time a user group is chosen randomly. In an inner loop the jobs of the other user groups are selected randomly and moved some positions towards the end. At the same time, only big jobs, as specified before, are moved. As before, the number of the moved jobs as well as the mean moving distance are controlled by endogenous strategy parameters, which could change during the run by self-adaptation. By means of this new mutation operator it is likely that the mutation operator generates an improvement in at least one of the objectives. Since another user group is chosen each time, improvements can be obtained in each direction of the solution space leading to a well-spread Pareto set.

Many recombination operators were proposed for symmetric problems such as TSP. For the asymmetric scheduling optimization problem such operators are inappropriate. Therefore, the recombination of schedules was done with several strategies as described in [2, C3.3], but none of them were successful. Hence, we reject to present the tried recombination operators in detail.

In EA there are two possibilities to initialize the starting population. On the one hand the population could be initialized with random elements. On the other hand we could start with some heuristically generated solutions, e. g. a mix of solutions such that each of solutions favors one of the users. These start solutions are only used for generating the first population and can not survive the first generation. Both alternatives were tested for the permutation representation.

Real-valued Coding To reduce the problem dimension, an approach using an alternative representation was developed. Instead of directly determining the positions of the jobs in the schedule, the parameters of an online scheduler are optimized. The online scheduler then generates a job sequence, which depends on these parameters. Again, for this sequence the different objectives are evaluated. By means of this indirect representation of job sequences, it is possible to reduce the number of variables to be optimized to 36 real valued parameters. Of course, the suggested representation has the disadvantage that not all possible job sequences can be coded. But even in the former strategy, due to the huge search space it is impossible to visit all elements of the search space within the given time. However, it can be assumed that the represented solutions comprise an interesting part of the search space with regards to the problem at hand.

A real-valued instantiation of an NSGA-II was used to optimize the parameters of the online-scheduler as described in Section 2. The evolutionary algorithm optimizes the parameters of the functions which calculate the precise weight of a job. We employed an NSGA-II with a $(\mu + \lambda)$-selection scheme. This time, the genome of an individual consists of the 36 real parameters of the job weighting function used by the

Greedy scheduler to generate a job sequence. These values are initialized with values chosen uniformly at random between problem specific lower and upper bounds. After the μ individuals of the initial population have been generated randomly, they are evaluated. The Greedy scheduler reorders the jobs in the waiting queue regarding its weighting functions which are parameterized by NSGA-II. Then the seven objective functions are evaluated and these values are returned to the evolutionary algorithm. From the perspective of the evolutionary algorithm, the combination of Greedy scheduler and evaluation of a job sequence serves as a black-box, and therefore the evaluation of each individual is just a function call of the simulated scheduler.

After having evaluated the initial population, the evolution process starts. Now, λ offsprings are generated from the current population by means of variation operators. First, with probability of $p_c = 0.9$ a recombination operator called SBX (see [4] for details) is applied. This operator randomly chooses two individuals from the current population as parents. From these parents one offspring is generated by simulating a single-point crossover of a binary string, meaning that the beginning of the string is taken from the first parent and behind a randomly chosen point the values from the second parent are taken. By using the SBX operator, the radius in the search space in which random variations of individuals are generated is proportional to the radius of the parent population. Therefore, the step size of variations gets smaller as the population concentrates around a local optimum. This offspring individual is then changed by a mutation operator called polynomial mutation [4]. Here with probability equalling the reciprocal number of parameters, each gene is slightly modified by an additive term that is distributed according to a polynomial function. The parameterization of the variation operators is exactly chosen as given in literature [5].

4 Scheduling Evaluation

The described methodology to develop an appropriate scheduling algorithm has been applied to various workload traces. Within this section the results achieved during the development process are presented.

In the following the evaluation objectives used in this work will be introduced followed by a detailed discussion about the achieved results.

4.1 Evaluation Objectives

In order to evaluate different schedules many scheduling objectives can be used. Objectives in the presence of Evolutionary Algorithms are often called criteria. In the following we use both terms interchangeable. Our research is restricted to seven objectives. On the one hand, those seven criteria enable the provider of the resources to choose an appropriate region within this 7-dimensional space where the preferred schedule should be generated. To our knowledge no other research on scheduling algorithms has used more than four objectives. On the other hand, such a decision space is a very good environment to use and adapt existing multiobjective Evolutionary Algorithms as seven dimensions are only very rarely examined.

All of the seven objectives to describe the achieved scheduling results can only be calculated at the end of a whole simulation. A simulation consists of one specific workload, a resource description and a selected scheduling method. The outcome is evaluated using the following seven objectives:

- the utilization of the whole parallel computer.
- the average weighted response time for all users together (AWRT).
- the average weighted response time for user group 1 (AWRT 1).
- the average weighted response time for user group 2 (AWRT 2).
- the average weighted response time for user group 3 (AWRT 3).
- the average weighted response time for user group 4 (AWRT 4).
- the average weighted response time for user group 5 (AWRT 5).

The calculation of the average weighted response time (AWRT) is defined in Equation (8). It uses the resource consumption (RC) for job j, that is defined on Page 2. By weighting the response time of each job with its resource consumption, all jobs with the same resource consumption have the same influence on the schedule quality. Otherwise a job with only few processors would have the same impact on this metric as a wider job with the same response time [25]. Note that in this context wider job means that many processors are needed to process the job.

$$\text{AWRT} = \frac{\sum\limits_{j \in \tau} (\text{RC}_j \cdot (C_j(S) - r_j))}{\sum\limits_{j \in \tau} \text{RC}_j} \tag{8}$$

The classification of the response time for different user groups is introduced as in real installations of parallel computers different user groups have different rights and priorities. This has been modeled in our work, as we assign jobs to certain user groups by calculating the resource consumption for all jobs. Now for each user the sum of all corresponding job resource consumptions is calculated and the user is assigned to a group. In our work we assume that the groups are built by users with a certain level of resource demand. Group 1 represents the ”power users” and group 5 the very infrequent users.

As the evaluation objectives are computed for each user group it is necessary to explain the generation of those user groups in more detail. Table 1 consists of the overall resource consumption which is needed for a user to become a member of the specified user group. Again, group 1 represents the ”power user” and group 5 the very infrequent user.

4.2 Used Workload Traces

The workload traces which were used for the evaluation process of the whole system are taken from the Standard Workload Archive [11]. Here we chose the seven well-known workloads from the NASA [12], the Cornell Theory Center (CTC) [15], the Swedish Royal Institute of Technology (KTH) [20], the Los Alamos National Lab (LANL) [9]

Table 1. The assignment of user to user groups depending on the corresponding resource consumption (RC).

User Group	1	2	3	4	5
user RC/overall RC	$> 8\%$	$> 2\%$	$> 1\%$	$> 0.1\%$	$\leq 0.1\%$

and the San Diego Supercomputer Center (SDSC SP2/SDSC 95/SDSC 96) [10, 27]. So each workload is recorded at a real parallel computer and includes detailed information about the stream of job requests for the computational resources. Each job description consists of 18 parameters like the job arrival time, the required number of nodes, the requested period of execution and the user identification [11]. For each job the user group, as explained above, was added.

The available workload traces differ much in their characteristics. One of the main reasons is the different number of available resources as shown in Table 2. This also prevents the comparison of the scheduling behavior between these traces. To this end, we scaled the traces as described in [7] up to 1024 processors. The data of those modified traces are presented in Table 3. These modified workload traces were used to create the multiobjective scheduling solution space.

Table 2. Original workload traces from the Standard Workload Archive [11].

Identifier	NASA	CTC	KTH	LANL	SDSC SP2	SDSC 95	SDSC 96
Machine	iPSC/860	SP2	SP2	CM-5	SP2	SP2	SP2
Period	10/01/93 - 12/31/93	06/26/96 - 05/31/97	09/23/96 - 08/29/97	04/10/94 - 09/24/96	04/28/98 - 04/30/00	12/29/94 - 12/30/95	12/27/95 - 12/31/96
Processors	128	430	100	1024	128	416	416
Jobs	42264	79302	28490	201378	67667	76872	38719

Table 3. Modified workload traces (only the modified values) regarding to [7].

Identifier	NASA	CTC	KTH	LANL	SDSC SP2	SDSC 95	SDSC96
Processors	1024	1024	1024	1024	1024	1024	1024
Jobs	188986	136471	167375	201378	310745	131762	66185

4.3 Generating an Approximate Pareto Front

As described in Section 2.1 the set of Pareto-optimal solutions can be generated in two ways. The approach of modifying the sequence of jobs for the whole workload will be described first.

During this step some technical restrictions exists. Each workload trace, described in Section 4.2 has a size of 10-15 MB. Therefore, the offspring within the Evolutionary Algorithm is restricted to about 60 elements. Otherwise the physical memory requirements within the simulations exceed the existing physical memory of the computer which performs the evolutionary operations. Each individual scheduling simulation has been performed on a separate computer within a cluster. Depending on the original workload file each simulation took about 40-120 minutes.

In the following, only results for the CTC workload will be presented. Except of the NASA workload all other traces show a similar behavior. The NASA workload is in some sense artificial as nearly no job ever waits for the execution, that is the jobs are started directly after submission [7]. Therefore, a reordering of the jobs is not meaningful in this case.

Furthermore we present several diagrams e.g. one for the utilization in combination with the overall AWRT, and one for the overall AWRT and the AWRT 1. For all other user groups similar diagrams could be drawn. Also the AWRT 1 and AWRT 2 will be compared. The diagrams for all other user group pairs look similar.

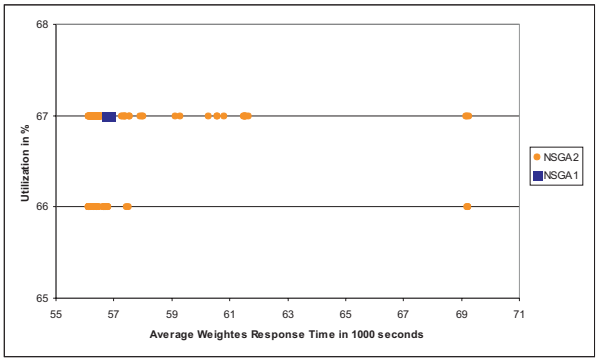

Fig. 6. NSGA and NSGA-II: Utilization versus AWRT using the CTC workload.

In Figures 6 to 8 all achieved non-dominated solutions for the NSGA and NSGA-II variants are presented. In both scenarios the sequence of the whole workload before the simulation was modified. The results clearly show, that the number of found solutions for both approaches is very small. In the sum, we only found three non-dominated solutions using NSGA. During the exploration of the solution space we tested several strategies. First, NSGA was used with a randomly generated initial population. The achieved results were very poor, as the best solution had a utilization of about 40 %.

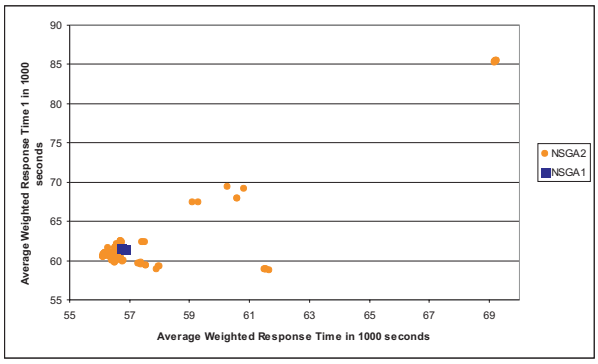

Fig. 7. NSGA and NSGA-II: AWRT versus AWRT 1 using the CTC workload.

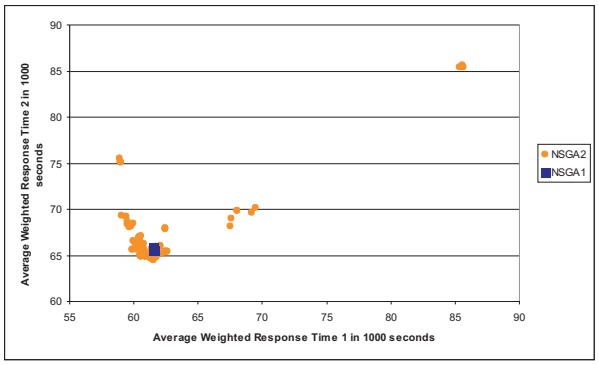

Fig. 8. NSGA and NSGA-II: Comparison of the AWRT 1 and AWRT 2 using the CTC workload.

In the following, the deterministically generated initial solutions were used (Figures 9 to 11). Note, the axis for the average weighted response times have in two cases (Figures 10 and 11) a logarithmic scaling. The use of starting solutions increased the quality, but still the number of achieved solutions was very small. The deterministically generated solutions provide already a higher diversity than the solutions found by NSGA. Here the utilization varies between about 38 % and 68 %. Also the average weighted response times for all users and all user groups vary of several magnitudes.

The quality of the achieved solutions was increased by changing the mutation operator as not only neighboring elements were exchanged but jobs could move for a number of jobs. This reflects the characteristics of the underlying scheduling problem as some jobs should be moved so that several later jobs can overcome this job in order to increase the scheduling quality for certain objectives.

In the following, NSGA-II was used with a subset of the deterministically generated start generation (121 solutions) and the changed mutation operator. The subset of

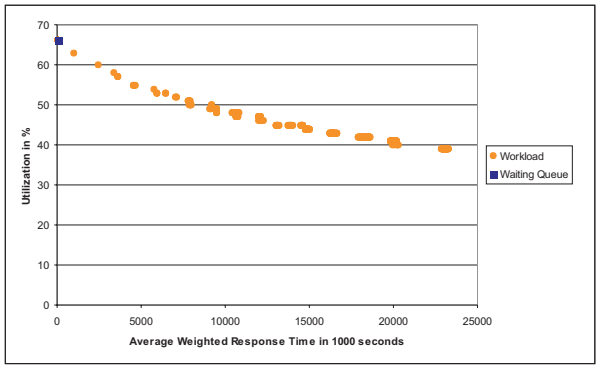

Fig. 9. Utilization versus AWRT of the start solutions generated by the permutations of the whole workload (Workload) and the local waiting queue. The CTC workload was used.

the initial solutions is the set of schedules that is generated by 5! permutations of the corresponding jobs of the five user groups and the original job sequence. The number of solutions and the diversity increased. Obviously the introduced elitism is necessary to generate better multiobjective solutions. Furthermore, the results demonstrate the advantage of using the crowding distance sorting within NSGA-II in comparison to the sharing method used in NSGA. The crowding distance sorting method results in a better diversity of the whole population. This is especially true for solutions that are not in the non-dominated set. A higher diversity in such regions results in a better diversity of the non-dominated solutions in later generations. Nevertheless, the overall number of found non-dominated solutions was small at around 100. The diversity of the non-dominated set was also small. Therefore, we applied three modifications to the system. First, we restricted the direction of the mutation in the way that jobs can only be moved into the direction of the end of the sequence. Another change that was motivated from the previous simulation results is the deselection of the recombination through the EMOA as all results which were created using recombination were of low quality. Furthermore, the usage of good starting solutions was helpful to improve the quality of the solutions. Nevertheless, a widely spread front of non-dominated solutions could not be processed. Therefore, multiple copies of the original job sequence were used as the only start solutions instead of the deterministically generated job sequences. All three modifications were applied to the NSGA-II algorithm. This did not affect the number of found solutions, but the diversity increased. The last additional modification that was applied to the NSGA-II algorithm was the selection of jobs to move. Here two restrictions were introduced as only jobs with a requested number of processors of 32 and/or jobs with a requested run time of more than 1 hour were moved. We applied both, the "and" and the "or" condition. This increased the diversity of the achieved results again. This reflects the influence of those jobs with a higher resource demand on the resulting scheduling objectives.

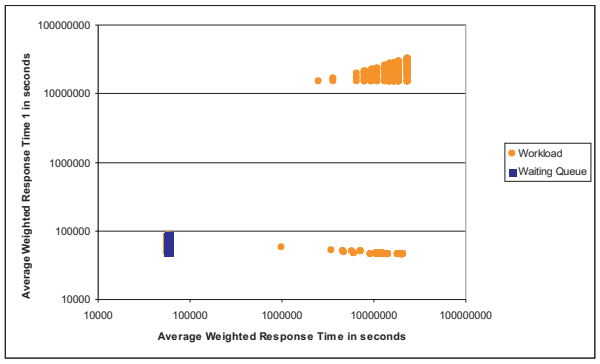

Fig. 10. AWRT versus AWRT 1 of the start solutions generated by the permutations of the whole workload (Workload) and the local waiting queue. The CTC workload was used.

Nevertheless, the number of found non-dominated solutions and the achieved diversity were not good. The final set only consists of solutions with 66 or 67 % utilization and the AWRT do not vary much in comparison to the given start solutions.

The results for the second approach by dynamically modifying the waiting queue with a real value coded Evolutionary Algorithm are presented in Figures 12 and 13. As stated already, the influence of this attempt is limited as only a limited number of jobs are reordered. As a result, the utilization does not vary much and is constant at about 66 %. However, the diversity of the overall AWRT and the average weighted response times of all user groups is of a high quality. Figure 12 represents the relative outcome of a special user group (here user group 1) to all users for the AWRT. Here, the AWRT 1 has a lower bound of about 50,000 seconds. The overall AWRT does not have such a clear bound. The comparison of the average weighted response time between two user groups presents a well generated Pareto front as it can be seen in Figure 13. For both representative user groups a clear lower bound can be observed.

The diversity of the starting solutions is higher than the diversity of the solutions generated by the parameter value coded approach. Nevertheless, the second approach has a much better diversity than the first attempt. Additionally, all solutions are in the "most" interesting area with a high machine utilization but a higher variation of waiting times between the different user groups. Furthermore, the second approach provides about 13,000 non-dominated solutions and therefore highly covers the solution space in this restricted area.

The final approximated Pareto set is generated by using all solutions of the NSGA, NSGA-II and the real-value coded approach as well as all starting solutions. The approximate Pareto set is calculated using all those solutions. Figures 14 to 16 present this set which consists of about 14,000 solutions. Again, the axis for the average weighted response times are scaled logarithmic. For all objectives the front has an acceptable diversity as the parameters for the waiting times are varying between several magnitudes and the utilization is covered between about 38 and 67 %. This set is used for the final development of an adapted Greedy scheduling.

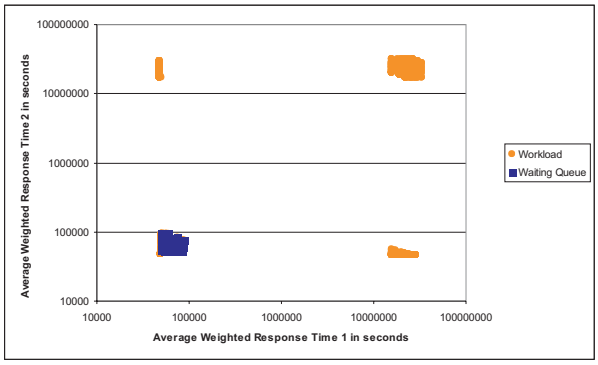

Fig. 11. Comparison of the AWRT 1 and AWRT 2 of the start solutions generated by the permutations of the whole workload (Workload) and the local waiting queue. The CTC workload was used.

Table 4 summarizes all results. The deterministic permutation of the whole workload and the local waiting queue produce 540 non-dominated solutions each. The overall performance of the permutation coded Evolutionary Algorithms is very low in comparison to the real value coded approach. However, the NSGA-II algorithm performs better than NSGA and the applied modifications increase those effects. The real value coding solves the problem of generating the approximation to the Pareto front best. Here the number of found solutions is very high. Therefore, this strategy should be applied in such high dimensions and with workloads of a similar size.

Results for the Development of an Adapted Greedy-Scheduling This report does not present the results for the development of an adapted Greedy scheduling as this step is still in progress. Corresponding results will be published soon.

5 Conclusion

This paper presents a methodology to automatically generate an adapted scheduling algorithm based on the machine owner preferences. To this end, the space of possible scheduling solutions was explored by using multiobjective Evolutionary Algorithms. Exemplarily a solution space of seven dimensions was created. This is the first attempt to generate the Pareto set of possible solutions with seven dimensions using real workload traces.

The paper presents two ways for the generation of the approximate Pareto front and compares the achieved results. During the usage of well-known multiobjective Evolutionary Algorithms several observations were made. In the presence of scheduling with workload traces of about 300,000 jobs and seven objectives and a restricted number of 5,000 evaluations it is not possible to use the Evolutionary Algorithms as a black-box optimization tool. Instead, problem specific knowledge has to be incorporated into the

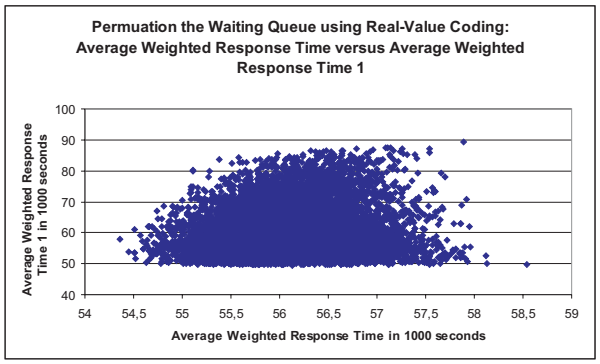

Fig. 12. Comparison of the AWRT and the AWRT 1 using the CTC workload and the real-value coded approach.

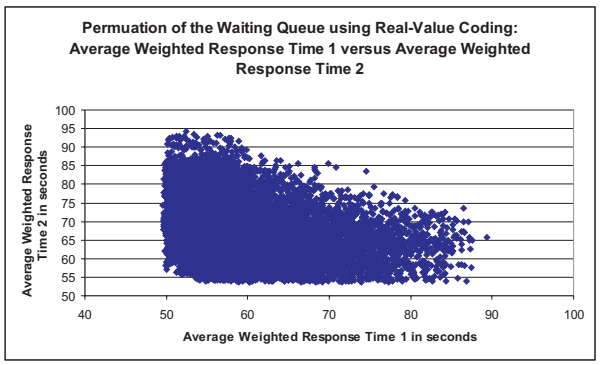

Fig. 13. Comparison of the AWRT 1 and AWRT 2 using the CTC workload and the real-value coded approach.

variation operators. For example it is necessary to restrict the type of jobs that are moved during the mutation as well as to determine the direction of the mutation. Furthermore, the usage of recombination can not be recommended.

Furthermore, this paper demonstrates that the applied Evolutionary Algorithms need acceptable start solutions. Otherwise they are not able to find appropriate solutions within the restricted number of evaluations.

The paper demonstrates the known advantage of combining the elitist selection and the crowding distance sorting implemented in NSGA-II compared to the used NSGA.

From the scheduling point of view it seems to be more promising to vary the weighting parameters for jobs in the waiting queue instead of directly modifying the sequence even noticed that by this procedure not all possible job sequences can be generated. The variation of whole workload traces does not work well as the number of jobs in

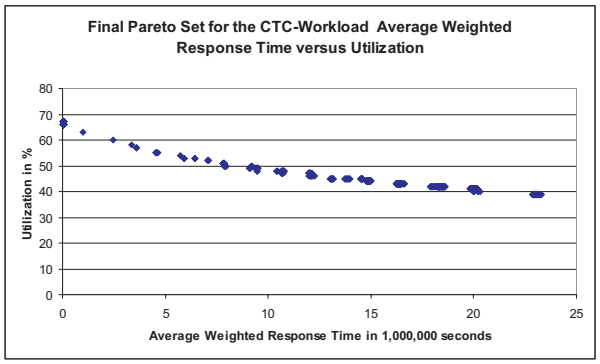

Fig. 14. Utilization and AWRT of the final results for the CTC workload trace.

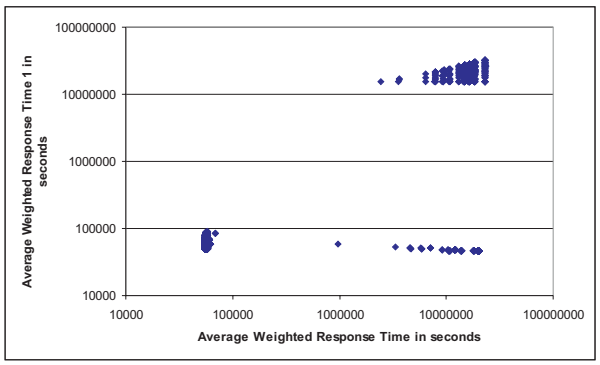

Fig. 15. AWRT and AWRT 1 for the final results for the CTC workload trace.

combination with the higher dimensions of the objective space is still too complex to be solved with Evolutionary Algorithms within the restricted number of evaluations.

The created space of possible solutions represents the foundation to generate an adapted Greedy scheduling algorithm. The corresponding results will be presented soon.

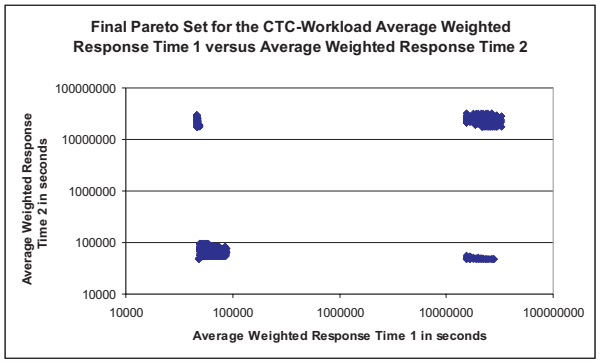

Fig. 16. AWRT 1 and AWRT 2 for the final results for the CTC workload trace.

Table 4. Summarized evaluation results. MJL ≙ Mutation Jump Length. MJN ≙ Mutation Jump Number. StS ≙ is the number of start (initial) schedules. NoS ≙ number of non-dominated solutions. FC ≙ function calls, which is equal to the number of generated schedules.

Representation	Evolutionary Multiobjective Algorithm	Mutation	Adaptation	Recombination	StS	NoS	FC
Workload Permutation	-	-	-	-	-	540	540
Queue Permutation	-	-	-	-	-	540	540
Permutation Coded	NSGA	MJL=1, MJN=5,000	-	0.3	121	3	24,000
Permutation Coded	NSGA	MJL=4,000, MJN=5,000	-	0.3	121	1	24,000
Permutation Coded, All NSGA Results	NSGA	-	-	-	-	3	48,000
Permutation Coded	NSGA-II	MJL=4,000, MJN=5,000	-	0.3	121	106	24,000
Permutation Coded	NSGA-II	MJL=1,500, MJN=5,000	Movement of jobs towards the end.	0.0	1	108	30,000
Permutation Coded	NSGA-II	MJL=1,500, MJN=5,000	Randomly chosen user group. Movement of jobs (with 32 processors and/or 1 hour run time) of other groups towards the end.	0.0	1	125	30,000
Permutation Coded, All NSGA-II Results	NSGA-II	-	-	-	-	131	84,000
Real Value Coded	NSGA-II	Polynomial	Real Value Coded	SBX	-	12,832	20,000
Final Solution	-	-	-	-	-	13,954	153,080

References

1. S. Albers. Online algorithms: A survey. *Mathematical Programming*, 97:3–26, 2003.
2. T. Bäck, D. B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation*. Oxford University Press, New York, and Institute of Physics Publishing, Bristol, 1997. Release 97/1.
3. H.-G. Beyer. *The Theory of Evolution Strategies*. Springer, Berlin, Heidelberg, 2001.
4. K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. John Wiley & Sons, Chichester, UK, 2001.
5. K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A Fast and Elitist Multiobjective Genetic Algorithm: NSGA–II. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, April 2002.
6. C. Ernemann, V. Hamscher, and R. Yahyapour. Economic Scheduling in Grid Computing. In D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing*, volume 2537 of *Lecture Notes in Computer Science*, pages 128–152. Springer-Verlag, 2002.
7. C. Ernemann, B. Song, and R. Yahyapour. Scaling of Workload Traces. In D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, *Job Scheduling Strategies for Parallel Processing: 9th International Workshop, JSSPP 2003 Seattle, WA, USA, June 24, 2003*, volume 2862 of *Lecture Notes in Computer Science*, pages 166–183. Springer-Verlag Heidelberg, October 2003.
8. C. Ernemann and R. Yahyapour. *Grid Resource Management - State of the Art and Future Trends*, chapter Applying Economic Scheduling Methods to Grid Environments, pages 491–506. Kluwer Academic Publishers, 2003.
9. D. G. Feitelson. Memory usage in the LANL CM-5 workload. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1291 of *Lecture Notes in Computer Science*, pages 78–94. Springer Verlag, 1997.
10. D. G. Feitelson. Metric and Workload Effects on Computer Systems Evaluation. *Computer*, 36(9):18–25, September 2003.
11. D. G. Feitelson. Parallel Workloads Archive. http://www.cs.huji.ac.il/labs/parallel/workload/, December 2004.
12. D. G. Feitelson and B. Nitzberg. Job characteristics of a production parallel scientific workload on the NASA Ames iPSC/860. In D. G. Feitelson and L. Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 949 of *Lecture Notes in Computer Science*, pages 337–360. Springer-Verlag, 1995.
13. D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, and K. C. Sevcik. Theory and Practice in Parallel Job Scheduling. *Lecture Notes in Computer Science*, 1291:1–34, 1997.
14. D. G. Feitelson and A. M. Weil. Utilization and Predictability in Scheduling the IBM SP2 with Backfilling. In *Proceedings of 12th International Parallel Processing Symposium and the 9th Symposium on Parallel and Distributed Processing*, pages 542–547, Los Alamitos CA, 1998. IEEE Computer Society Press.
15. S. Hotovy. Workload Evolution on the Cornell Theory Center IBM SP2. In Dror G. Feitelson and Larry Rudolph, editors, *Job Scheduling Strategies for Parallel Processing*, volume 1162 of *Lecture Notes in Computer Science*, pages 27–40. Springer-Verlag, 1996.
16. C. B. Lee, Y. Schwartzman, J. Hardy, and A. Snavely. Are user runtime estimates inherently inaccurate? In *Proceedings of the 10th Workshop on Job Scheduling Strategies for Parallel Processing*, pages 153–161, June 2004.
17. D. A. Lifka. The ANL/IBM SP scheduling system. In *Proceedings of IPPS'95 Workshop: Job Scheduling Strategies for Parallel Processing*, volume 949 of *Lecture Notes in Computer Science*, pages 295–303. Springer, 1995.

18. D. McLaughlin, S. Sardesai, and P. Dusgupta. Preemptive Scheduling for Distributed Systems. In *11th International Conference on Parallel and Distributed Computing Systems*, September 1998.
19. R. McNaughton. Scheduling with deadlines and loss functions. *Management Science*, 6(1):1–12, October 1959.
20. A. W. Mu'alem and D. G. Feitelson. Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling. *IEEE Trans. Parallel & Distributed Syst.*, 12(6):529–543, June 2001.
21. E. Naroska and U. Schwiegelshohn. On an on-line scheduling problem with parallel jobs. *Information Processing Letters*, 81:297–304, 2002.
22. M. Pinedo. *Scheduling: Theory, Algorithms, and Systems*. Prentice-Hall, New Jersey, 2 edition, 2002.
23. U. Schwiegelshohn. Preemptive weighted completion time scheduling of parallel jobs. *SIAM Journal of Computing*, 33:1280–1308, 2004.
24. U. Schwiegelshohn and R. Yahyapour. Improving first-come-first-serve job scheduling by gang scheduling. In *IPPS'98 Workshop: Job Scheduling Strategies for Parallel Processing*, volume 1459 of *Lecture Notes in Computer Science*, pages 180–198. Springer-Verlag, March 1998.
25. U. Schwiegelshohn and R. Yahyapour. Fairness in parallel job scheduling. *Journal of Scheduling*, 3(5):297–320, 2000.
26. D. Sleator and R. E. Tarjan. Self-adjusting binary search trees. *Journal of the ACM*, 32(2):652–686, 1985.
27. K. Windisch, V. Lo, R. Moore, D. G Feitelson, and B. Nitzberg. A comparison of workload traces from two production parallel machines. In *6th Symposium Frontiers Massively Parallel Computing*, pages 319–326, October 1996.