Development of Scheduling Strategies with Genetic Fuzzy Systems

Carsten Franke [a,1], Frank Hoffmann [b], Joachim Lepping [a], Uwe Schwiegelshohn [a]

[a] *Robotics Research Institute - Section Information Technology, University Dortmund, D-44221 Dortmund, Germany*

[b] *Chair for Control System Engineering, University Dortmund, D-44221 Dortmund, Germany*

Abstract

This paper presents a methodology for automatically generating online scheduling strategies for a complex objective defined by a machine provider. To this end, we assume independent parallel jobs and multiple identical machines. The scheduling algorithm is based on a rule system. This rule system classifies all possible scheduling states and assigns a corresponding scheduling strategy. Each state is described by several parameters. The rule system is established in two different ways. In the first approach, an iterative method is applied, that assigns a standard scheduling strategy to all situation classes. Here, the situation classes are fixed and cannot be modified. Afterwards, for each situation class, the best strategy is extracted individually. In the second approach, a Symbiotic Evolution varies the parameter of Gaussian membership functions to establish the different situation classes and also assigns the appropriate scheduling strategies. Finally, both rule systems will be compared by using real workload traces and different possible complex objective functions.

Key words: Scheduling Algorithm Development; Online Scheduling; Genetic Fuzzy System; Symbiotic Evolution

1. Introduction

In this paper, we address an online scheduling problem with independent jobs submitted by different users. Unfortunately, for those problems, most common simple scheduling objectives, like the makespan [19], the total weighted completion [30] or response time [44] are not sufficient to represent the intentions of the machine provider. As example take the scheduling of computational jobs on a parallel computer system. There the users typically are partitioned into a small set of user groups which are assigned different priorities. Today, parallel computer systems are increasingly part of Computational Grids, that is, users from other sites with often low priority use those systems as well. As the common scheduling objectives are not suited to efficiently handle those priorities, the machine provider often sets quotas to prevent lower priority groups to occupy too many resources. However, those quotas tend to reduce machine utilization significantly.

Similar problems occur in many other application areas of practical importance, like telecommunications and logistics. Nevertheless, we focus in this paper on parallel computer systems as for those systems, real trace data are available that can be used to evaluate new approaches, see, for instance, the Standard Workload Archive [15], described by Chapin et al. [9]. It is vital to use real data to develop good scheduling algorithms

Email addresses: carsten.franke@udo.edu (Carsten Franke), frank.hoffmann@udo.edu (Frank Hoffmann), joachim.lepping@udo.edu (Joachim Lepping), uwe.schwiegelshohn@udo.edu (Uwe Schwiegelshohn).

[1] Born Carsten Ernemann.

as there is no method that guarantees an optimal or almost optimal solution for all input data.

The scheduling of parallel computer systems is an online problem as jobs are submitted over time and the precise processing times of those jobs are frequently not known in advance. Furthermore, information about future jobs are usually not available [24]. Formally, each job j is part of a job system τ and belongs to a user group. It is characterized by its degree of parallelism m_j, its processing time p_j, and additional criteria, see Feitelson et al. [16]. During the execution phase, job j requires the concurrent and exclusive access to $m_j \leq m$ processing nodes with m being the total number of nodes on the parallel computer system. The number of required processing nodes m_j is available at the release date r_j of job j and does not change during the execution. As the network does not favor any subset of the nodes and all nodes of a parallel computer system are either identical or very similar, we assume that a job j can be processed on any subset of m_j nodes of the system. Note that in the rest of this paper, we replace the expression node by machine as this is the common terminology in scheduling problems.

As already mentioned, the processing time p_j is not known before the job has been completed. However, some systems require the users to provide an estimate $\bar{p}_j$ of this processing time. If the actual processing time of a job exceeds this estimate, the job is canceled to protect the system and the user from the costs of additional resource consumption by a possibly faulty job. In addition, this estimate is also used for some scheduling algorithms, like Backfilling [32]. Unfortunately, those estimates are of limited use for scheduling purposes as users tend to overestimate the processing time of their jobs in order to avoid the termination of correct jobs, see, for example, Lee et al. [31]. Most current real installations of parallel computers do not use preemption but let all jobs run to completion unless they are terminated as discussed above. The completion time of job j within the schedule S is denoted by $C_j(S)$. Furthermore, precedence constraints between jobs are rare and it is almost impossible for the machine owner to detect them if they exist. Hence, we assume that all jobs are independent.

Contrary to many theoretical online scheduling problems addressed previously, see, for instance, Albers [1], the scheduling of job j does not need to take place directly after its release date r_j. Instead, the allocation is established when sufficient machines become available, that is just before the start of job j at time $(C_j - p_j)$. Due to the frequency of job submissions and the size of m for many parallel computers, scheduling decisions must still be made within a short period of time after the release of a job. As information about future job submissions is not available compute intensive quasi offline scheduling algorithms cannot be used in this scenario.

In general, it is very difficult to optimally solve most scheduling problems. This is even true for offline problems with simple scalar objective functions as many of them are NP-hard [20]. Therefore, existing online scheduling systems at real installations mainly use heuristics, like Greedy Scheduling [23] in combination with a First Come First Serve (FCFS) approach and the above mentioned Backfilling [32,21]. Although those algorithms produce a very low utilization in the worst case [39] they yield reasonably good results in practice [17]. But they do not allow different priorities for jobs or user groups apart from exclusively assigning computer partitions to user groups [27], limiting the total processing time of a user group, or statically weighing the waiting time of jobs. All those algorithms are rather cumbersome and often lead to a low utilization as already mentioned. Recently, there have been suggestions to consider market oriented scheduling approaches [7] which accept more flexible objectives and are able to adapt to variations in the job submission pattern. However, existing research projects in this area only use two simple linear objective functions: time and cost minimization [6]. So far no really flexible scheduling algorithm has been developed for parallel computer systems.

Within this work, we present a methodology to automatically generate a rule based scheduling system that is able to produce good quality schedules with respect to a given complex objective. This objective defines the prioritization of different user groups. Even if different providers use the same basic objectives for the various groups the transformation of a generic multiobjective scenario into a specific scheduling problem with a single objective depends on the actual priorities assigned to the user groups and is likely to be individual. Hence, we focus on the development of a suitable methodology. To this end, we present a rule based scheduling that is able to adapt to various scenarios.

So far, the use of rule based systems in scheduling environments is rare. Nevertheless, first attempts [12,8] have shown the feasibility of such an approach. However, those scheduling systems are all based on single objective evaluation functions that are not optimized.

The proposed scheduling process is divided into two steps. In the first step, the queue of waiting jobs is reordered according to a sorting criterion. Then an algorithm uses this order to schedule the jobs onto the available machines in the second step. Based on the present scheduling state, the rules determine the sorting criterion and the scheduling algorithm. In order to guarantee general applicability, the system classifies all possible scheduling states. This classification considers the actual schedule, the current waiting queue, and additional external factors, like the time of day.

As already mentioned, the development of scheduling strategies for parallel computers is typically based on real workload traces. Those traces implicitly include all relevant characteristics and hidden properties, like temporal patterns, dependencies between jobs, or certain user behaviors. As local scheduling decisions influence the allocation of future jobs, the effect of a single decision cannot be determined individually. Therefore, the whole rule base is only evaluated after the complete scheduling of all jobs belonging to a workload trace. This has a significant influence on the learning method to generate this rule base as this type of evaluation prevents the application of a supervised learning algorithm. Instead, the reward of a decision is delayed and determined by a critic. Clearly, all potential rules compete against each other during the training phase. But as the final scheduling system will probably consist of many different rules, they also need to cooperate in order to produce rule bases with high quality. For the training, we use a genetic reinforcement algorithm in combination with a *Symbiotic Evolution* as introduced by Juang et al. [29].

To exemplarily show the results of our approach, we use a linear priority functions which favors user group 1 oder over user group 2 over all other user groups. The choice of another priority function may lead to slightly different results but does not affect the feasibility of our methodology. Due to the lack of a scheduling algorithm supporting priority functions, no priority functions are available in practice. Therefore, we had to define one.

For the evaluation of the derived scheduling algorithms, we demonstrate the distance of this schedule from the Pareto front of all feasible schedules for this workload. However, the generation of an approximate Pareto front is not subject of this paper. But two restrictions must be noted:

(i) For real workloads, we are only able to generate approximate Pareto fronts. Therefore, schedules of this front are not guaranteed to be lower bounds.

(ii) The schedules are generated off-line. On-line methods may not be able to achieve those results due to the on-line constraints.

Moreover, we also show the results of the best conventional strategy that does not support priorities.

The remainder of this paper is organized as follows. In Section 2, we introduce the underlying scheduling system, Evolutionary Algorithms, and Symbiotic Evolution in more detail. The next section contains scheduling objectives and features. Then the model of our approach is described in Section 4. This is followed by a detailed analysis of the system behavior and an evaluation of the results. The paper ends with a brief conclusion.

2. State of the Art

This section introduces classical algorithms for job scheduling and their application within our rule based scheduling system. Furthermore, we present the concept of Evolution Strategies. Those strategies are used to optimize the scheduling rule base during Symbiotic Evolution. The concept of this evolution is shown at the end of this section.

2.1. *Scheduling Concepts*

As already mentioned, scheduling strategies for high performance parallel computers must pay more attention to certain privileged users or user groups in order to achieve a higher degree of satisfaction for them and to support Grid computing. Unfortunately, workload data neither contain a classification of users nor any target objective function for the scheduling algorithms. It is likely that most users of the same user group show a similar job submission pattern. This is especially true in commercial settings where high vol-

ume customers may obtain preferential treatment. But even in science, jobs from nuclear physicists often resemble each other while they are typically different from weather simulations or jobs with frequent access to databases. Therefore, an arbitrary assignment of users to user groups is not likely to represent reality. Instead we base our user groups assignment on job submission patterns.

Song et al. [42] have already shown that users at real installations can typically be divided into 5 groups based on their resource demands. Due to the lack of given user membership information, we use Song's grouping approach to generate 5 user groups such that user group 1 represents all users with the highest resource consumption whereas all users in group 5 have a very low resource demand. Details of the user group definitions are provided by Ernemann et al. [3].

Here, we consider only on-line systems where jobs must at least temporarily compete for computer resources. This will automatically lead to waiting jobs that are typically ordered in form of a queue. In some installations, a static ordering like sorting by submission time or sorting by estimated runtime is applied while other systems dynamically reorder the waiting queue depending on the system state by using a complex sorting function. Remember that the state of a scheduling system mainly consists of the current schedule that describes the actual allocation of processor nodes to certain jobs, the scheduling results achieved so far, and the queue of waiting jobs.

The various scheduling algorithms mainly differ in the way they select a job from the sorted waiting queue to insert it into the existing schedule, that is, they obey different restrictions when choosing the next job. This results in different algorithmic complexities and correspondingly different execution times for the scheduling algorithms.

In the following, we present four simple scheduling algorithms that together cover most of the algorithms used in existing installations. Note that the first three algorithms use a statically sorted waiting queue while the last algorithm dynamically reorders this queue.

– *First Come First Serve (FCFS)* starts the first job of the waiting queue whenever enough idle resources are available. Hence, this algorithm has a constant computational complexity as the scheduler only tests whether the first job can be started if a job in the schedule has completed its execution or if a new job is on top of the waiting queue.
– *List Scheduling* as introduced by Graham [22] is not applied in this work. However, it is the base for the two backfilling variants. When applying List Scheduling, the scheduler scans the queue of waiting jobs in the given order until it finds the first job for which a feasible allocation on the currently idle resources exists. The computational complexity is higher than in the case of FCFS as the whole queue may be tested each time the scheduling procedure is initiated.
  · *EASY Backfilling (EASY)* requires that a runtime estimation is provided for each job by its user. If the first job of the waiting queue cannot be started immediately, the algorithm allows an allocation of another job if it does not delay the completion of the *first job*. EASY has a higher computational complexity than List Scheduling, as the scheduler may need to consider the allocation of the first job.
  · *Conservative Backfilling (CONS)* also needs runtime estimations and extends the concept of EASY. Here, an allocation of a job is feasible if the completion time of *no preceding job* in the waiting queue is delayed. This results in a much higher computational complexity as in the worst case, the expected completion time of almost all jobs within the waiting queue must be determined each time the scheduling process is initiated.
– *Greedy Scheduling (Greedy)* dynamically reorders the waiting queue whenever it is invoked. To this end, it uses a potentially complex sorting criterion. After the resorting, a simple FCFS is applied. The complexity of this algorithm is based on sorting which may be executed frequently. Greedy allows the specification of user or user group dependent preferences within the sorting criterion.

2.2. *Evolution Strategies*

For the generation of a rule based scheduling system, we apply Symbiotic Evolution, see Section 2.3. This concept is generally based on Evolutionary Algorithms which mimic natural evolutionary processes to execute search and optimization procedures. Although many different algorithms and principles have been proposed, they usually fit into one of the following three major classes.

(i) The class of Evolutionary Programming defined by Fogel [18],
(ii) the class of Genetic Algorithms invented by Holland [26], and
(iii) the class of Evolution Strategies explored by Schwefel [37] and Rechenberg [35].

Here, we will discuss Evolution Strategies in more detail as we apply them during the Symbiotic Evolution to generate a rule based scheduling system in form of a Genetic Fuzzy system. In contrast to the majority of publications, we do not apply Genetic Algorithms for the evolutionary generation of our Fuzzy systems as our underlying scheduling system is real value encoded and Jin et al. [28] have shown that Evolution Strategies outperform Genetic Algorithms when applied to Genetic Fuzzy systems with a real value encoded representation. This is further supported by Bäck and Schwefel's comparative studies [2] on Evolution Strategies and Genetic Algorithms that demonstrated the general superiority of Evolution Strategies for real valued parameter optimization problems. In addition, Evolution Strategies support self adaptation which is often helpful when exploring the structure of the fitness landscape.

Algorithm 1 depicts the conceptual structure of a $(\mu \dot{+} \lambda, \rho)$-Evolution Strategy [4].

The algorithms initializes the first population P_0 with μ individuals $(\boldsymbol{o}_i, \boldsymbol{s}_i)$, $1 \leq i \leq \mu$ where $\boldsymbol{o}_i$ and $\boldsymbol{s}_i$ denote the sets of object and strategy parameters of the i-th individual, respectively. The evolutionary loop is executed until a given termination criterion, like a fixed number of generations or a quality level within the objective space, is satisfied. The actual generation P_t is used to produce the new offspring generation Q_t of λ individuals by applying mutation to the parent individuals. If $(\rho > 1)$ a new individual is breed by the recombination of ρ randomly selected parent individuals and subsequent mutation. An evaluation procedure with a predefined objective function that represents the real problem determines the parameters of this new offspring generation. In the *comma* strategy, the next parent generation P_{t+1} is selected just from the offspring individuals Q_t while the *plus* strategy selects the best individuals of both the parent and offspring generations. All other individuals are removed from the system. Afterwards the next loop iteration starts.

Algorithm 1 $(\mu \dot{+} \lambda, \rho)$-Evolution Strategy

$t \leftarrow 0$
$P_{0,\mu} \leftarrow$ initialization with μ individuals
evaluation($P_{0,\mu}$)
repeat
  for $k = 1$ to λ **do**
    $M_{k,\rho} \leftarrow$ random selection of ρ individuals $(P_{t,\mu})$
    $\boldsymbol{s}_k \leftarrow$ strategy recombination $(M_{k,\rho})$
    $\boldsymbol{o}_k \leftarrow$ object recombination $(M_{k,\rho})$
    $\tilde{\boldsymbol{s}}_k \leftarrow$ strategy mutation $(\boldsymbol{s}_k)$
    $\tilde{\boldsymbol{o}}_k \leftarrow$ object mutation $(\tilde{\boldsymbol{s}}_k, \boldsymbol{o}_k)$
  end for
  $Q_{t,\lambda} \leftarrow \{(\tilde{\boldsymbol{o}}_k, \tilde{\boldsymbol{s}}_k), k = 1, \ldots, \lambda\}$
  evaluation($Q_{t,\lambda}$)
  if (μ, λ)-selection **then**
    $P_{t+1,\mu} \leftarrow$ select μ individuals $(Q_{t,\lambda})$
  else if $(\mu + \lambda)$-selection **then**
    $P_{t+1,\mu} \leftarrow$ select μ individuals $(P_{t,\mu} \cup Q_{t,\lambda})$
  end if
  $t \leftarrow t + 1$
until termination condition

2.3. *Symbiotic Evolution of Genetic Fuzzy Systems*

There are several methods to generate rule based scheduling systems. Here, we distinguish between assigning a combination of sorting criteria and scheduling algorithms to feature subspaces with fixed boundaries and Genetic Fuzzy systems.

In our scheduling problem, neither precise knowledge about the applicability of scheduling strategies for a given scheduling situation nor training data are available. Furthermore, individual scheduling decisions can only be evaluated after all jobs have been assigned to resources, see Section 1. Hence, a critic can only issue the award for the assignment of a scheduling strategy to a situation once the schedule of the whole workload trace is complete. Finally, an appropriate situation classification is also not known in advance but must be determined implicitly during the generation of the rule based scheduling system.

Many design methods for Fuzzy logic controllers use Evolutionary Algorithms to adjust the membership function and to define the output behavior of individual rules, see, for example, Hoffmann [25]. Among those methods, Genetic Fuzzy systems are especially

well suited as they do not need any training data nor expert knowledge. All those Genetic Fuzzy systems either encode single rules (*Michigan approach* [5]) or complete rule bases (*Pittsburgh approach* [41]).

As already mentioned, we use a Symbiotic Evolutionary Algorithm as introduced by Juang et al. [29] to generate a Genetic Fuzzy system. The Symbiotic Evolutionary Algorithm is based on the Michigan approach and evaluates different rules together as several rules are needed to compose a complete rule base. This approach does not require the definition of several complete rule bases at the beginning of the evolution. Hence, we do not need to establish a set of well suited initial rule bases which is generally quite difficult, see Juang et al. [29].

It has already been pointed out in Section 1, that single rules do not only cooperate but also compete with each other during the evolution. In order to alleviate this contrast, Hoffmann [25] restricts the competition of rules to subsets that trigger for similar inputs. This restriction influences the selection of rules and the recombination between them. The resulting rules typically possess a higher diversity and therefore further improve the coverage of the possible input values. Due to the better coverage this diversity often reduces the number of generations needed to establish the final rule base system. Therefore, we combine the Symbiotic Evolutionary Algorithm with Hoffmann's approach as the computational effort is of great importance for our problem due to our time consuming simulations of the schedule execution.

3. Scheduling Objectives and Features

As described in Section 1, we are are looking for scheduling algorithms that optimize a given complex objective function of a machine provider. Without loss of generality, we assume that this complex objective function is a combination of simple classic objective functions that are partially restricted to the jobs of a user group.

Further, we need features that enables us to classify possible scheduling situations within our rule based scheduling system. As there must be a relationship between the scheduling objective and the feature set those features are also defined with respect to the user groups.

Before presenting the objectives and the features in detail, we introduce some useful definitions and notations.

- $\mathrm{RC}_j = p_j \cdot m_j$ as the Resource Consumption of a single job j,
- τ the set of all n jobs within our scheduling system,
- $\xi(t)$ the set of already finished jobs at time t,
- $\pi(t)$ the set of actual running jobs at time t, and
- $\nu(t)$ the set of waiting jobs at time t.

To represent the relation between a job and a user group, we use function

$$\varrho_i(j) = \begin{cases} 1 & \text{if job } j \text{ belongs to user group } i \\ 0 & \text{otherwise.} \end{cases}$$

3.1. *Scheduling Objectives*

Within this work, we use seven classical scheduling objectives as basic blocks for the complex objective. The first two objectives do not consider the user group of a job.

Overall Utilization (U):

$$\mathrm{U} = \frac{\sum\limits_{j \in \tau} p_j \cdot m_j}{m \cdot \left(\max\limits_{j \in \tau} \{C_j(S)\} - \min\limits_{j \in \tau} \{C_j(S) - p_j\} \right)}$$

Average Weighted Response Time (AWRT) over all jobs of all users:

$$\mathrm{AWRT} = \frac{\sum\limits_{j \in \tau} p_j \cdot m_j \cdot (C_j(S) - r_j)}{\sum\limits_{j \in \tau} p_j \cdot m_j}$$

Note that the weight of a job is identical to its resource consumption independent of their processing time and degree of parallelism, see Schwiegelshohn et al. [40]. Therefore, the objectives U and AWRT are closely related to each other. Nevertheless, we decided to use both of them as utilization is common criterion used in large installations and AWRT can easily be restricted to a specific user group. This yields the remaining user group specific objectives:

$$\mathrm{AWRT}_i = \frac{\sum\limits_{j \in \tau} p_j \cdot m_j \cdot (C_j(S) - r_j) \cdot \varrho_i(j)}{\sum\limits_{j \in \tau} p_j \cdot m_j \cdot \varrho_i(j)}$$

As we have 5 user groups the complex objective function in our system is based on the combination of 7 simple objectives.

3.2. *Feature Definitions*

While scheduling objectives are used to evaluate a complete schedule, features describe the present state of the schedule in order to enable decisions about the next scheduling action. Note that our simple scheduling objectives require the knowledge of job completion times which are not available for jobs that are presently processed. However, those jobs should be considered when describing the present schedule state. Therefore, our features slightly differ from the above presented objectives.

Before introducing those features, we define the *Slowdown* (SD_j) for a single job j within schedule S:

$$\mathrm{SD}_j = \frac{C_j(S) - r_j}{p_j}$$

SD_j will reach its minimum value of 1 if job j does not wait before it starts execution. Then the release date is identical with the job's start time. Although, the range of this feature is theoretically unbounded, we can delimit it to the interval of [1,100] as values greater than 10 occur very rarely in practice.

The feature *Average Weighted Slowdown* (SD) for all already processed jobs $j \in \xi(t)$ uses the same weight definition as the objective AWRT.

$$\mathrm{SD} = \frac{\sum\limits_{j \in \xi(t)} p_j \cdot m_j \cdot (C_j(S) - r_j)}{\sum\limits_{j \in \xi(t)} p_j^2 \cdot m_j}$$

SD represents the weighted average delay of jobs due to competition and the scheduling algorithm. Further, this feature represents the scheduling decisions in the past as only already finished jobs are used to calculate this feature. Here, we have not limited the window of the SD. In practical cases, a limitation to, for instance, the last month may be appropriate.

The *Momentary Utilization* (U_m) of the whole parallel computer at time t only considers those jobs that are processed at this time:

$$\mathrm{U}_m = \frac{\sum\limits_{j \in \pi(t)} m_j}{m}$$

The *Proportional Resource Consumption of the Waiting Queue for User Group* i (PRCWQ_i) represents the share of this user group among the jobs in the waiting queue:

$$\mathrm{PRCWQ}_i = \frac{\sum\limits_{j \in \nu(t)} \bar{p}_j \cdot m_j \cdot \varrho_i(j)}{\sum\limits_{j \in \nu(t)} \bar{p}_j \cdot m_j}$$

Note that the real processing time p_j is unknown for the jobs in the waiting queue. Therefore, we use the estimated processing time $\bar{p}_j$ instead.

Here, we have preferred simple scheduling objectives and features that are frequently found in real installations of parallel systems. However, our methodologies are not restricted to this selection. Remember, we assume 5 user groups within our scheduling system, Thus, those 5 feature values represent the expected future which might enable the system to react on a changed user demand in a flexible fashion.

4. Rule Based Scheduling

For a rule based scheduling approach, every possible scheduling state must be assigned to a corresponding situation class. A complete rule base consists of a set of rules R_i that cover all possible situations of the scheduling system. Each rule contains a conditional and a consequence part. The conditional part describes the conditions for the activation of the rule and the consequence part represents the corresponding controller behavior. However, as we are using several rules the term recommendation specifies the output of a single rule while the consequence is the superpositioning of the recommendations of all rules in a rule base. In order to specify each scheduling state in an appropriate fashion each rule defines certain partitions of features as a conditional part. The rule base must select at least one rule for every possible system situation.

The scheduling strategy also includes two steps:

(i) A **sorting criterion** for the waiting queue $\nu(t)$.

(ii) A **scheduling algorithm** that uses the order of $\nu(t)$ to schedule one or more jobs.

We use the term *strategy* to describe the whole scheduling process that consists of both steps. First, the chosen sorting criterion determines the sequence of jobs within the waiting queue. Second, the selected

scheduling algorithm finds a processor allocation for at least one job of the sorted waiting queue. We have chosen four different sorting criteria:

– *Increasing Number of Requested Processors:* Preference of jobs with little parallelism and therefore higher utilization.
– *Increasing Estimated Run Time:* Preference of short jobs and therefore higher job throughput.
– *Decreasing Waiting Time:* Preference of long waiting jobs and therefore more fairness.
– *Decreasing User Group Priority:* Preference of jobs from users with a higher resource demand.

The selected scheduling algorithm is one of the four methods presented in Section 2.1. Note that Greedy is already a complete scheduling strategy while the other described scheduling algorithms need a sorting criterion of $\nu(t)$ in addition.

The general concept of the rule based scheduling approach is depicted in Figure 1.

As 4 different sorting criteria with 3 possible scheduling algorithms and the combined Greedy strategy are available, we have to chose one of 13 strategies for each possible state. However, it is not practicable to test all possible assignments in all possible states. For example, lets assume a very coarse division of each feature into only 2 partitions. Then 13 possible strategies and 7 features result in $13^{2^7} \approx 3.84 \cdot 10^{142}$ simulations if all combinations in all possible situations are tested. Additional problems occur during the generation of a rule based scheduling system as the number and reasonable partitions of features that are required to describe the conditional parts of the rules in an appropriate way are generally unknown in advance.

Hence, we introduce two possible approaches to derive a rule based scheduling system using only a limited number of simulations. The first static attempt, the iterative rule base generation, is based on a sequential optimization of the output behavior assignment to a predefined situation class, see Section 4.1. The second dynamic approach, Symbiotic Evolution, automatically extracts an appropriate situation classification with the corresponding output behavior and is presented in Section 4.2.

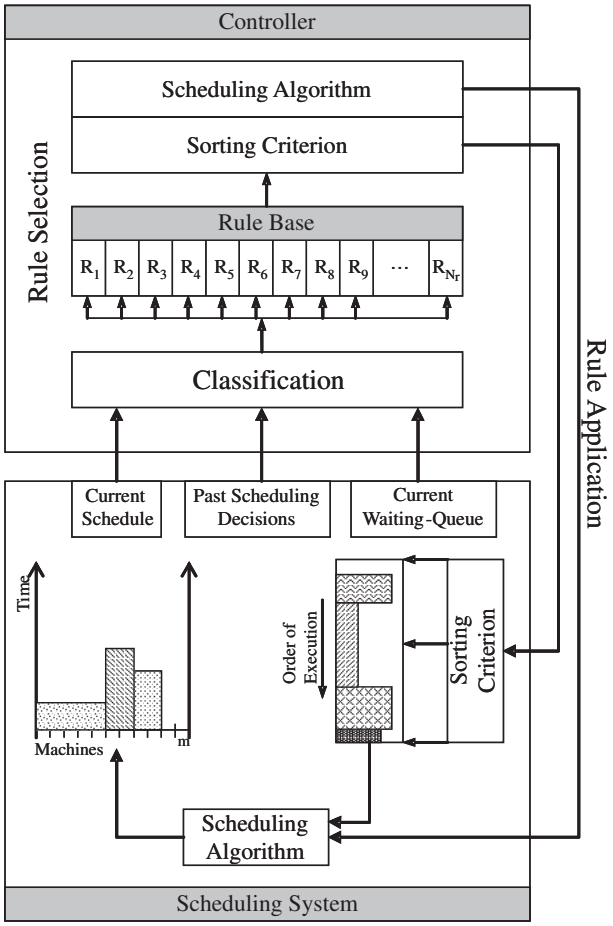

Fig. 1. General Concept of the Rule-Based Scheduling Approach.

4.1. *Iterative Rule Base Generation*

We use $N_F = 7$ features within our rule based system. For the iterative approach, the number of intervals that divide the whole value range of the features is fixed, that is, each feature ω has $N_p^{(\omega)} - 1$ static bounds, that divide the possible value range of ω into $N_p^{(\omega)}$ partitions. Generally, a larger number of partitions $N_p^{(\omega)}$ of a feature ω potentially leads to a more accurate rule set while more situation classes must be optimized. Overall, this results in N_r situation classes that must be provided to cover all possible system states with

$$N_r = \prod_{\omega=1}^{N_F} N_p^{(\omega)}. \quad (1)$$

Then Algorithm 2 assigns sorting criteria and scheduling algorithms to situation classes.

Algorithm 2 Iterative Optimization of Rules.

```
Assign a standard strategy to all N_r rules
R_1, ..., R_{N_r} of rule base R
for i = 1 to N_r do
  f_old(R_i) = ∞
  for s = 1 to 13 do
    Use strategy s as the consequence part of
    rule R_i and run a simulation with the result
    f_new(R_i)
    if (f_new(R_i) < f_old(R_i)) then
      Assign strategy s to rule R_i
      f_old(R_i) = f_new(R_i)
    end if
  end for
end for
```

FCFS together with the sorting by waiting time has been selected as the standard scheduling strategy in Algorithm 2. This strategy is used at most real installations.

The iterative rule base generation has the advantage of easy implementation and generates rule bases that can easily be interpreted by providers. Future scheduling development may benefit from knowledge gained through this kind of interpretation. The selected scheduling algorithms and sorting criteria for a certain scheduling situation can directly be extracted from the corresponding rules without further computation.

However, the fixed division of the whole feature space has a critical influence on the performance of the scheduling system. At the moment, no mechanism is provided that automatically adjusts the defined partitions. In addition, a reduction of the number of features is not yet considered, that means that all situation classes or rules must contain and specify all features. This approach optimizes each situation class separately with limited influence on other allocations. Thus, it can be assumed that the resulting rule base can still be improved by considering the cooperation aspects of different rules. The next section describes our second approach that avoids most of the mentioned problems.

4.2. *Symbiotic Evolutionary Approach*

Our generation of a Genetic Fuzzy system is based on a Symbiotic Evolutionary Algorithm, as introduced by Juang et al. [29]. First, we describe how a single Fuzzy rule is coded as an individual within the Symbiotic Evolutionary Algorithm.

4.2.1. *Coding of Fuzzy Rules*

For a single rule R_i, every feature ω of all N_F features is modeled by a Gaussian membership function (GMF).

$$g_i^{(\omega)}(x) = \frac{1}{\sigma_i^{(\omega)}\sqrt{2\pi}} \exp\left\{ \frac{-(x-\mu_i^{(\omega)})^2}{2{\sigma_i^{(\omega)}}^2} \right\} \quad (2)$$

Rule R_i consists of a set of feature descriptions in the conditional part where each feature ω is described by a pair of real values $\mu_i^{(\omega)}$ and $\sigma_i^{(\omega)}$. $\mu_i^{(\omega)}$ is the center of feature ω. Therefore, this value defines an area of system states in the feature space where the influence of the rule is very high. Note that using a GMF as feature description the condition

$$\int_{-\infty}^{\infty} g_i^{(\omega)}(z)dz = 1 \,\forall\, i \in \{1, \dots N_r\} \wedge \omega \in \{1, \dots N_F\}$$

always holds. In other words, for increasing $\sigma_i(\omega)$ values, the peak value of the GMF decreases because the integral remains constant. Using this property of a GMF we are able to reduce the influence of a rule for a feature ω completely by setting $\sigma_i^{(\omega)}$ to a very high value. For $\sigma_i^{(\omega)} \to \infty$, a rule has no influence for feature ω anymore. With this approach, it is also possible to establish a kind of default value that is used if no other peaks are defined in a feature domain. Based on this feature description, a single rule can be described by

$$R_i = \left\{ g_i^{(1)}(x),\ g_i^{(2)}(x),\ \dots,\ g_i^{(N_F)}(x), \Omega_i \right\}. \quad (3)$$

In Figure 2, we show a coding scheme for an individual object vector where one individual represents one rule. The consequence of rule R_i is vector of numbers $\mathbf{\Omega}(R_i)$ that specify a weight for each possible scheduling strategy. Thereby, it is possible to specify a scheduling strategy which is particularly favorable for a certain system state and another one which is clearly

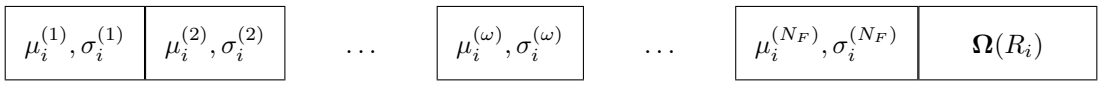

Fig. 2. Coding of a Chromosome for the Symbiotic Evolutionary Algorithm.

unfavorable. The final output decision is then computed by the superposition of the consequence parts of the involved rules.

The main advantage of using several GMFs for describing a single rule is the automatic coverage of the whole feature space. In contrary to the iterative approach, a rule gives a scheduling strategy for all possible situations. Hence, it is the focus of this approach to find a meaningful set of N_r rules that generates a good rule base system

$$\mathcal{R} = \{R_1,\ R_2,\ \ldots,\ R_{N_r}\}. \tag{4}$$

4.2.2. *Controller Output Decision*

For each rule R_i and the actual feature vector $\boldsymbol{x} = (x_1, x_2, \ldots, x_{N_F})$, we compute a degree of membership $\phi_i(\boldsymbol{x})$

$$\begin{aligned}\phi_i(\boldsymbol{x}) &= \bigwedge_{\omega=1}^{N_F} g_i^{(\omega)}(x_\omega) \\ &= \prod_{\omega=1}^{N_F} \frac{1}{\sigma_i^{(\omega)}\sqrt{2\pi}} \exp\left\{\frac{-(x_\omega - \mu_i^{(\omega)})^2}{2\sigma_i^{(\omega)^2}}\right\}\end{aligned} \tag{5}$$

The degree of membership of a single rule is calculated by performing an "AND" operation on the GMFs of the rule's features.

4.2.2.1. *General Output Decision* For all rules we collect the corresponding values $\phi_i(\boldsymbol{x})$ in a membership vector

$$\boldsymbol{\phi}(\boldsymbol{x}) = \Big(\ \phi_1(\boldsymbol{x})\ \ \phi_2(\boldsymbol{x})\ \ \ldots\ \ \phi_{N_r}(\boldsymbol{x})\ \Big). \tag{6}$$

Further, $\boldsymbol{\Omega}(R_i) \in \mathbb{R}^{N_\Omega}$ specifies the recommendation vector of rule R_i. This vector includes the corresponding weights for the N_Ω possible consequence parts. In our example, each of the 13 possible strategies is assigned to a position in $\boldsymbol{\Omega}(R_i)$.

We provide a weight for every element in $\boldsymbol{\Omega}(R_i)$. These weights can assume the values:

$-5 \mathrel{\hat{=}}$ particularly unfavorable
$-1 \mathrel{\hat{=}}$ unfavorable
$0 \mathrel{\hat{=}}$ no recommendation
$1 \mathrel{\hat{=}}$ favorable
$5 \mathrel{\hat{=}}$ particularly favorable.

The consequences of all rules are described by the matrix

$$\boldsymbol{C}^{N_\Omega \times N_r} = \Big[\ \boldsymbol{\Omega}(R_1)\ \boldsymbol{\Omega}(R_2)\ \ldots\ \boldsymbol{\Omega}(R_{N_r})\ \Big]. \tag{7}$$

Now we compute the weight vector $\boldsymbol{\Psi}$ by multiplying the membership vector $\boldsymbol{\phi}(x)$ with the transposed matrix $\boldsymbol{C}^T$

$$\boldsymbol{\Psi} = \boldsymbol{\phi}(\boldsymbol{x}) \cdot \boldsymbol{C}^T \tag{8}$$

such that $\boldsymbol{\Psi} = (\Psi_1 \Psi_2 \ldots \Psi_{N_\Omega})$ contains the superpositioned weight values for all N_Ω possible output decisions. The expression

$$\arg \max_{1 \le k \le N_\Omega} \{\boldsymbol{\Psi}_k\} \tag{9}$$

determines the final output decision of the whole rule system.

4.2.2.2. *Default Output Decision* Unfortunately, Fuzzy control systems may lead to controller scattering. In some cases, the system may initiate a change of output even for a minimal input parameter variation. To avoid frequent switching between two output decisions, we introduce a threshold for the activation of a controller output. Then, the output of the system is only determined by the output corresponding to the superpositioned membership value if the total degree of membership value exceeds a predefined constant threshold level. Otherwise, a standard output is used.

Using a normalized GMF a feature encoding enables the establishment of a default value by selecting a large $\sigma_i^{(\omega)}$ for the corresponding feature ω. This modeling of a default value is now combined with the above mentioned thresholds.

The threshold value and the default output are part of an evolutionary optimization process. They are encoded in a single chromosome. Furthermore, a second

population of default value individuals is optimized in parallel. The evaluation of the two populations is performed in combination, see Section 4.3.4.

4.2.3. *Similarity Measure for Rules*

An acceptable Fuzzy rule base is expected to possess a good coverage, that is, the correlation between GMFs should be small. Otherwise if only rules with a high correlation are combined then the system may not be able to cover most of the possible scheduling states in an appropriate fashion. Therefore, a measure of similarity is needed for the selection of rules during the evolutionary optimization process.

For the Symbiotic approach, a rule system $\mathcal{R}$ is composed by N_r rules. In many cases, several rules exist in a population, that are centered on very similar configurations for the conditional part. In order to combine only rules which represent different states within a system $\mathcal{R}$, a similarity measure is needed. This similarity measure is used to build rule bases by incrementally adding rules with the smallest similarity to the last selected rule. The first rule is picked randomly.

Specifically, we define the similarity of two rules by the cross correlation of the corresponding GMFs. This cross correlation $\varphi_{g_1 g_2}$ of two GMFs $g_1(z)$ and $g_2(z)$ for one feature can be computed by:

$$\varphi_{g_1 g_2} = \int_{-\infty}^{\infty} g_1(z) \cdot g_2(z)\, dz$$
$$= \frac{1}{\sqrt{2\pi}\sqrt{\sigma_1^2 + \sigma_2^2}} \exp\left\{ \frac{-(\mu_1 - \mu_2)^2}{2(\sigma_1^2 + \sigma_2^2)} \right\}. \quad (10)$$

For two rules R_i and R_j with N_F features, we compute the total cross correlation by averaging the cross correlation over all features:

$$\varphi_{R_i,R_j} = \frac{1}{N_F} \sum_{\omega=1}^{N_F} \int_{-\infty}^{\infty} g_i^{(\omega)}(z) \cdot g_j^{(\omega)}(z)\, dx$$
$$= \frac{1}{N_F \cdot \sqrt{2\pi}} \sum_{\omega=1}^{N_F} \frac{\exp\left\{ \frac{-(\mu_i^{(\omega)} - \mu_j^{(\omega)})^2}{2(\sigma_i^{(\omega)^2} + \sigma_j^{(\omega)^2})} \right\}}{\sqrt{\sigma_i^{(\omega)^2} + \sigma_j^{(\omega)^2}}}. \quad (11)$$

Remember that for our Symbiotic Evolutionary Algorithm, the number of features N_F remains constant during the whole optimization process.

Based on the similarity, φ_{R_i,R_j} we compose the rule bases as shown in Figure 3. Intuitively, only those rules are combined to a rule base which have a small similarity. Thus, we try to ensure a high degree of diversity between rules within the rule bases and a high coverage of the feature space.

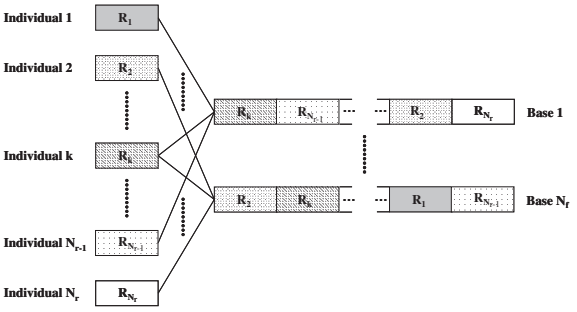

Fig. 3. Composition of Rules to Rule-Systems by Juang et al. [29].

4.2.4. *Fitness Assignment*

The fitness assignment for all individuals within a generation is obtained by Algorithm 3 where N_f and N_r are the number of bases and the number of rules per base, respectively. With $N_s^{(i)}$ we denote the number of Fuzzy systems in which R_i participated.
We subtract the resulting fitness $f(R_i)$ of a single rule R_i from a predefined maximum value $f_{\max}$ to achieve a minimization problem. Then, we are consistent with the scheduling objective functions which also must be minimized.

Formally, the fitness assignment $f(R_i)$ of rule R_i is given by

$$f(R_i) = f_{\max} - \frac{1}{N_s^{(i)}} \sum_{n=1}^{N_s^{(i)}} \frac{f_n(R_i)}{N_r} \quad (12)$$

In Equation 12, $f_n(R_i)$ denotes the temporal fitness assignment determined by the overall performance of the n-th Fuzzy system of all $N_s^{(i)}$ rule bases in which R_i participated during the population evaluation process. Usually, it can be assumed that a fair fitness evaluation of the whole population is only guaranteed if the number of participations within the rule bases is constant for all rules within the population. To ensure that this condition is obeyed, we also allow the composition of rules to systems which may have a high degree of similarity. This is only applied, if a rule is

Algorithm 3 Fitness Assignment for the Symbiotic Evolutionary Algorithm

$f(R_i) = 0 \; \forall i \in \{1, \dots \lambda\}$
$N_s^{(i)} = 0 \; \forall i \in \{1, \dots \lambda\}$
for $n = 1$ to N_f **do**
  $S = Q_{t,\lambda}$
  $\mathcal{R}_n = \emptyset$
  R_i is selected randomly from S
  $S = S \backslash R_i$
  $\mathcal{R}_n = \mathcal{R}_n \cup R_i$
  for $a = 1$ to $(N_r - 1)$ **do**
    Select $R_j \in S$ with $R_j = \arg\min_{R_k \in S} \{\varphi_{R_i,R_k}\}$
    $\mathcal{R}_n = \mathcal{R}_n \cup R_j$
    $S = S \backslash R_j$
    $R_i = R_j$
  end for
  $f_n \leftarrow$ evaluate$(\mathcal{R}_n)$
  $f(R_i) = f(R_i) + f_n / N_r \; \forall R_i \in \mathcal{R}_n$
  for $i = 1$ to N_r **do**
    if $R_i \in \mathcal{R}_n$ **then**
      $N_s^{(i)} = N_s^{(i)} + 1$
    end if
  end for
end for
$f(R_i) = f(R_i)/N_s^{(i)} \; i \in \{1, \dots, \lambda\}$

very similar to another rule, but has not participated sufficiently enough in different bases.

4.2.5. *Selection by Clusters*

Selection is an important operator for Evolutionary Algorithms because it controls the diversity of the whole population. The selection operator is able to conserve good solutions for multiple generations. In our case, a rule base should cover most of the possible system states. Furthermore, the population of an Evolutionary Algorithm should also keep and improve those rules that describe niches within the feature space. Therefore, a special selection mechanism has been developed, that does not only refer to the individuals fitness but also to their degree of coverage within the feature space, see Algorithm 4.

As already mentioned, a rule's position in the N_F-dimensional feature space is determined by its GMF-$\mu_i^{(\omega)}$-value. To ensure a good coverage of most of the possible system states it is sufficient to consider only the clusters of the GMF centers. Therefore, we restrict our analysis to the identification and clustering of similar GMF-$\mu^{(\omega)}$-values of the population. For the determination of clusters we apply the computationally efficient k-Means algorithm with an exchange method as proposed by Späth [43].

Algorithm 4 Selection Strategy.

Determine k clusters $\{c_1, \dots, c_k\}$ of all rules $R_1, \dots, R_{N_r}$ in the population $Q_{t,\lambda}$ based on their GMF-$\mu^{(\omega)}$-values.
for $i = 1$ to k **do**
  Compute the mean fitness $\bar{f}(c_i)$ for cluster c_i by selecting the $b = \min\{|c_i|, n\}$ individuals $\{R_1, \dots, R_b\} \in c_i$ with the best fitness according to

$$\bar{f}(c_i) = \frac{1}{b} \sum_{l=1}^{b} f(R_l)$$

  {Here, n represents the number of individuals that are used to determine the mean of the fitness of individuals in all clusters.}
end for
for $i = 1$ to k **do**
  Add μ_i individuals from c_i to $P_{t+1,\mu}$ with

$$\mu_i = \left\lfloor \frac{\mu_s}{\bar{f}(c_i) \sum_{l=1}^{k} 1/\bar{f}(c_l)} + 0.5 \right\rfloor$$

end for
while $|P_{t+1,\mu}| \leq \mu_s$ **do**
  Select best R_i from the remaining individuals according to the best cluster and individual fitness.
  $P_{t+1,\mu} = P_{t+1,\mu} \cup R_i$
end while

Note that the fitness of a cluster is determined by the mean of the n best individuals and that we address the minimization of the fitness objective. This allows the selection of individuals with a good fitness value even if the majority of the cluster members have a very bad fitness. Algorithm 4 describes by μ_s the number of rules that must be selected over all cluster. μ_i describes the number of rules that are selected from the i-th cluster. Of course this must be an integer value and hence we round to the next integer value. For the evaluation we have chosen $k = 10$ and $n = 5$.

4.3. *Configuration of the Symbiotic Evolutionary Algorithm*

Different modifications of Evolution Strategies have been developed for parameter optimization. Here, we have chosen a $(\mu + \lambda)$-Evolution Strategy. The specification of the μ and λ values is closely connected to the parameter values for the Symbiotic Approach.

4.3.1. *Population Size*

Note that the variables μ and λ correspond to an Evolution Strategy in this section. The number N_r of rules that form a Fuzzy system is static for the Symbiotic Approach and must be chosen at the beginning of the optimization. After applying several experiments with different values of N_r we have selected $N_r = 50$. The resulting systems have shown a higher performance than systems with a different number of participating rules. As no rule exists twice in a Fuzzy system, each population must consist of at least $\mu = 50$ potential rules. As recommended by Schwefel [38], the ratio of $\mu/\lambda = 1/7$ should be used for the Evolution Strategy. This results in $\lambda = 350$ child individuals in the evolutionary process. Hence, 350 individuals must be evaluated within each generation.

As proposed by Juang et al. [29], we must ensure that every rule within the population participates sufficiently enough in Fuzzy systems. In our work, we assume that it is possible to determine the fitness of a rule if it is part of exactly $N_s = 10$ Fuzzy systems. Consequently, we must evaluate $N_f = \frac{350 \cdot 10}{50} = 70$ rule bases in each generation. This results in 70 simulations for each generation.

4.3.2. *Recombination Operator*

The recombination operator and also the mutation operator of the next section modify the individual rules during the evolutionary process. During the discussion of the Evolution Strategy we use a vector $\boldsymbol{a}$ of components a_k, where each a_k corresponds to either a $\mu_i^{(\omega)}$, a $\sigma_i^{(\omega)}$, or an output of the underlying rule.

We haven chosen discrete multirecombination as the recombination operator. For this approach, a single new offspring is recombined from $\rho = 12$ parents. Each component a_k of the offspring is randomly chosen to be one of the 12 possible components of the $\rho = 12$ parents at the same position. As we need to generate λ new individual we select for each new individual ρ parents from the μ potential parents for recombination and repeat this procedure λ-times.

However, those 12 parents are not all randomly selected from the current population. We select $\rho/2 = 6$ parents with a very high degree of similarity and the remaining $\rho/2 = 6$ parents randomly. The first 6 similar individuals are selected by determining the first individual randomly and selecting all other individuals using the total cross correlation. Thereby we ensure that the recombination operator leads to new rules in other areas of the feature space while still covering the already explored areas of the feature space. Hence, this procedure reflects the compromise between exploration and coverage of the feature space.

The actual recombination is described by Equation 13 where Z is randomly chosen from the interval [0,1[.

$$a_k^{\mathrm{rec}} = \begin{cases} a_1 & \text{if } 0 \leq Z < \frac{1}{\rho} \\ a_2 & \text{if } \frac{1}{\rho} \leq Z < \frac{2}{\rho} \\ \vdots & \\ a_\rho & \text{if } \frac{\rho-1}{\rho} \leq Z < 1 \end{cases} \tag{13}$$

Since we do not use any complex self adaptive parameters that must be considered for the recombination procedure, the recombination is similarly performed for all elements of the parameter vector.

The recombination does not affect any parameter of the mutation operator.

4.3.3. *Mutation Operator*

For the mutation operator, we have selected the commonly accepted Mutation Success Rule as proposed by Rechenberg [36]. This rule adapts the standard deviation of the Gaussian mutation by incorporating the success of the last mutations. To this end, we count the number of mutations that lead to better rules regarding their fitness and the number of all mutations. For each generation, except the first one, the quotient ζ is again computed by the number of successful mutations in relation to all performed mutations:

$$\zeta = \frac{\text{Number of successful mutations}}{\text{Number of all mutations}}. \tag{14}$$

Note, in our work, a mutation is called successful if the recombined and mutated offspring is better than a randomly selected parent from the ρ parents. Further, we adapt the mutation step size σ in each generation:

$$\sigma^{(new)} = \begin{cases} \alpha\sigma^{(old)} & \text{if } \zeta > \zeta_0, \\ \frac{1}{\alpha}\sigma^{(old)} & \text{if } \zeta < \zeta_0, \\ \sigma^{(old)} & \text{otherwise} \end{cases} \quad (15)$$

with $\alpha = 1.224$ and $\zeta_0 = 1/8$.

This mutation step size influences the variation of each vector component a_k of the vector that describes a rule. The new vector component $a_k^{(\text{new})}$ after mutation is computed by

$$\boldsymbol{a}_k^{(new)} = \boldsymbol{a}_k^{(old)} + \mathcal{N}(0, \sigma^{(old)}). \quad (16)$$

In this paper, we use a relatively large population for the Evolutionary Algorithm. The original Mutation Success Rule was developed for the (1+1)-Evolution Strategy [36]. It is considered, for instance, by Deb [10] that the usually optimal ratio $\zeta_0 = 1/5$ is too optimistic in case of such a configuration. In order to avoid a continuous reduction of the mutation step size, we have decreased the optimal success rate to $\zeta_0 = 1/8$.

4.3.4. *Population for Default Values*

As already described, we generate a second population of default value individuals that is evolutionary optimized in parallel to the optimization of the Fuzzy rules. Therefore, we introduce a new species of real value coded individuals that describe a certain threshold and a recommended output decision which is used as default value. This recommend scheduling strategy is used to generate the next schedule if the superpositioning of all rules within the Fuzzy system generates a value that is smaller than the defined threshold. Due to this additional population the whole procedure can be considered as a Cooperative Coevolutionary Algorithm [34] approach, as a default value individual will participate in each generated Fuzzy system. Both populations exist genetically isolated within the ecosystem, that is, individuals will only mate with other members of their own species. By evolving both species in different populations these mating restrictions are always obeyed.

The default value population consists of N_f individuals for the offspring generation. This is reasonable, as the evaluation of the Fuzzy rules requires N_f Fuzzy systems where each of those Fuzzy systems contains one default value. Hence, the corresponding population of default values should also consist of $N_f = 70$ individuals which leads to a (10+70)-Evolution Strategy by applying Schwefel's recommendation [38] ($\mu/\lambda = 1/7$).

For the actual fitness evaluation, the two populations must cooperate. Once a Fuzzy system has been generated by composing it from rule individuals, a single default value is added to this system, and the overall performance is evaluated by simulation. The assigned fitness value for the default value individual is equal to the performance of the Fuzzy system in which this individual has participated. Note that the fitness value of the rule population is influenced by the default value individuals. So this cooperation may lead to better rules and default values as well. For the default value population, we apply the "two-point rule" of Rechenberg [4] with $\alpha = 1.3$ and do not use recombination.

5. Evaluation

To evaluate the performances of the iterative and the Symbiotic approach addressed in this paper, various simulations have been carried out with workload traces from real computer installations being used as input data. Each resulting Fuzzy system has been simulated by a discrete event simulation. In order to demonstrate the flexibility of the approaches with respect to different preferred user groups, we tested several objective functions.

5.1. *Used Workload Traces*

The used workloads are all available from the Standard Workload Archive [15]. Six well known workloads have been selected. They origin at the Cornell Theory Center (CTC) [27], the Royal Institute of Technology (KTH) [33] in Sweden, the Los Alamos National Lab (LANL) [13] and the San Diego Supercomputer Center (SDSC00/SDSC95/SDSC96) [14,45]. Each of these workloads has been recorded at a real parallel computer and provides information about the job requests for the computational resources.

In order to make those workloads comparable they are scaled to a standard machine configuration with 1024 processors as described by Ernemann et al. [11]. The data of the used workloads are presented in Table 1. The fact that the available workload data are relatively old does not affect the presented methodology.

Identifier	CTC	KTH	LANL	SDSC00	SDSC95	SDSC96
Machine	SP2	SP2	CM-5	SP2	SP2	SP2
Period	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	1024	1024	1024	1024	1024	1024
Jobs	136471	167375	201378	310745	131762	66185

Table 1
Scaled Workload Traces from Standard Workload Archive [15] using the Scaling Procedure by Ernemann et al. [11].

5.2. *Formulation of User Group Prioritizations*

In general, any mathematical function can be used for the objective. However, we restrict ourselves to linear objective functions and assume that an adequate priority assignment can be established by weighting the sum of the AWRT objective of the different user groups:

$$f_{\mathrm{obj}} = \sum_{i=1}^{5} \theta_i \cdot \mathrm{AWRT}_i \quad \text{with} \quad \theta_i \in \mathbb{R}_0^+. \quad (17)$$

Each weight θ_i specifies the priority of user group i defined by the system provider.

In this paper, we exemplarily limited the prioritization to a maximum number of 2 user groups. A typical function that prioritizes user group 1 and user group 2 over all other users may be

$$f_{\mathrm{obj}} = 10 \cdot \mathrm{AWRT}_1 + 4 \cdot \mathrm{AWRT}_2.$$

As already mentioned in the introduction, we present our achieved results relative to the approximated Pareto front (PF) of all feasible schedules [3]. Although, we do not know the real Pareto front, the high density of our approximation indicated that the quality of the approximation is very good. Note that the approximated Pareto front was generated offline and it cannot be taken for granted that this front can be reached by our proposed online scheduling systems at all. Therefore, we refer to this front as a reference of the best achievable solution.

Feature	Intervals
SD	[1-2],]2-100]
$U_{m[\%]}$	[0-75],]75-85],]85-100]
$PRCWQ_{1[\%]}$	[0-20],]20-100]
$PRCWQ_{2[\%]}$	[0-20],]20-100]
$PRCWQ_{3[\%]}$	[0-25],]25-100]
$PRCWQ_{4[\%]}$	[0-25],]25-100]
$PRCWQ_{5[\%]}$	[0-25],]25-100]

Table 2
Feature Partitions for the Iterative Rule-base Generation.

5.3. *Results of the Iterative Approach*

First, we present the results that have been achieved by using the iterative approach. Here, we have assumed a single division of the intervals of SD and PRCWQ1-5 respectively. This results in each case in two partitions. Further, we use three partitions for the U_m feature. Overall, this produces $2^6 \cdot 3 = 192$ different rules that are needed to build a complete rule base.

Furthermore, different division values for the situation class features have been evaluated. Therefore, the complete iterative optimization process has been performed for slightly modified division values. The partitions which achieved the best results for the objective function are listed in Table 2.

With this feature partitions the iterative optimization procedures has been performed. The results of the corresponding simulations are given in Table 3. Note that we present those results as relative values compared with the standard scheduling strategy EASY. We have selected EASY as this strategy performs in most cases better than CONS or FCFS. For an objective K we computed the relative value as:

$$\mathrm{K}_{[\%]} = \frac{\mathrm{K}_{\mathrm{EASY}} - \mathrm{K}_{\text{rule based}}}{\mathrm{K}_{\mathrm{EASY}}} \cdot 100.$$

A positive value for the $\mathrm{AWRT}_{[\%]}$ objective denotes an improvement regarding to the standard strategy. For the $\mathrm{U}_{[\%]}$ objective a positive percentage value denotes a deterioration of the system's performance, as the utilization has to be maximized from the provider's point of view.

In Table 3, we show that it is possible to optimize most of the workloads with respect to the given objective function by the iterative approach. The improvements of the objectives result in a preference

$f_{obj} = 10 \cdot AWRT_1 + 4 \cdot AWRT_2$								
Workload	$AWRT_{[\%]}$	$AWRT_{1[\%]}$	$AWRT_{2[\%]}$	$AWRT_{3[\%]}$	$AWRT_{4[\%]}$	$AWRT_{5[\%]}$	$U_{[\%]}$	$f_{obj[\%]}$
CTC	-2.28	14.59	8.65	-6.46	-17.77	-40.43	0	12.78
KTH	-18.19	5.32	-16.05	-29.76	-20.42	-60.39	0	-8.83
LANL	-6.41	24.5	20.87	-29.02	-24.09	-213.66	0	23.36
SDSC00	-583.68	5.34	-492.24	-718.14	-1370.1	-1824.85	2.08	-130.39
SDSC95	1.21	3.38	-0.23	0.5	-6.58	7.3	0	2.28
SDSC96	-0.46	1.19	5.77	-18.6	1.11	2.76	0	2.49

Table 3
$AWRT_{[\%]}$ and $U_{[\%]}$ Objectives and $f_{obj[\%]}$ after Iterative Optimization of the Rule Base in Comparison to EASY (in %).

of the specified user groups, that is, user groups 1 and 2 achieved an approximately 20 % shorter AWRT compared to the standard algorithms by using the LANL trace. However, for other user groups, like user group 5, the AWRT increases about 200 %. Note that the utilization is nearly constant for all workloads.

However, for two workloads is is not possible to improve the objective with the iterative approach. It can be assumed, that an approach for the automatic generation of rule base also has to provide a mechanism for the automatic adjustment of the feature partitions. The different workload characteristics cannot be covered with a static set of partitions as it has been applied within this approach.

5.4. *Results of the Symbiotic Evolutionary Approach*

Now, the results achieved by the Symbiotic evolutionary approach are presented. Several simulations with different workloads and different objective functions have been done in order to evaluate the performance and the flexibility of this approach.

In all simulations, we could not observe any beneficial effect of the introduced default value and a threshold level. Obviously, the problem of controller scattering does not appear in our rule based scheduling systems or the impact of the threshold mechanism is in our case too small. Furthermore, the general ability of rules to cover the complete feature space by selecting a high value for $\sigma_i^{(\omega)}$ provides already a kind of default value mechanism. So, we do not consider thresholds for rule activations in the following.

The results for the objective function are presented in Table 4.

Figure 4 depicts the improvements of the objective function compared to EASY. Improvements have generally been achieved for all workloads. For the SDSC00 and the KTH workload good results could also be achieved with this approach contrary to the iterative optimization attempt. This indicates that the evolutionary generated rule bases provide more flexibility and adaptability for the different workload characteristics. It can be observed that for the preferred user groups a shorter AWRT is achieved. Along with this objective the AWRT for the other user groups 4 and 5 increase drastically. For the SDSC00 workload, for instance, the reduction of this value for user group 1 of 60 % corresponds to an increase of the AWRT for user group 5 of about 16000 %.

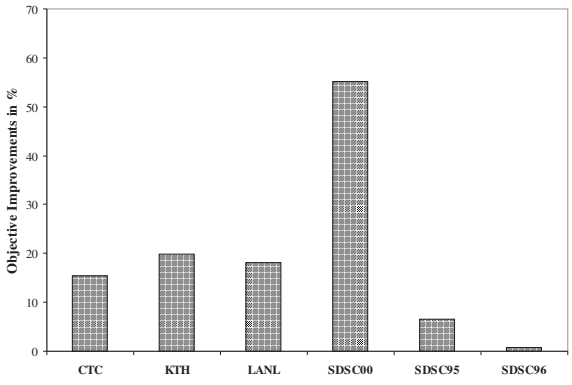

Fig. 4. Improvements of the Objective Function in Comparison to EASY Achieved by the Symbiotic Evolutionary Approach for All Workloads (in %).

Several simulations generated similar results for most of the used workloads. Simulations with different objective functions that expressed different user group prioritizations lead to similar results.

In all cases, the Symbiotic evolutionary approach clearly outperforms the iterative optimization concept.

In Figure 5 the improvements of the evolutionary optimized objective function in comparison to the

$f_{obj} = 10 \cdot AWRT_1 + 4 \cdot AWRT_2$								
Workload	$AWRT_{[\%]}$	$AWRT_{1[\%]}$	$AWRT_{2[\%]}$	$AWRT_{3[\%]}$	$AWRT_{4[\%]}$	$AWRT_{5[\%]}$	$U_{[\%]}$	$f_{obj[\%]}$
CTC	-5.42	16.64	12.73	1.99	-26.36	-153.63	0	15.45
KTH	-44.37	25.67	8.15	-51.32	-189.05	-1095.83	0	19.94
LANL	-13.52	19.71	14.83	-23.32	-47.62	-269.36	0	18.21
SDSC00	-852.06	60.32	41.58	-5.15	-2562.25	-16162.14	4.9	55.21
SDSC95	-1.01	9.07	0.4	-20.74	-43.34	-40.68	0	6.48
SDSC96	-1.28	1.96	-2.07	-16.22	-15.12	-13.34	0	0.81

Table 4
$AWRT_{[\%]}$ and $U_{[\%]}$ Objectives and $f_{obj[\%]}$ after Symbiotic Optimization in Comparison to EASY (in %).

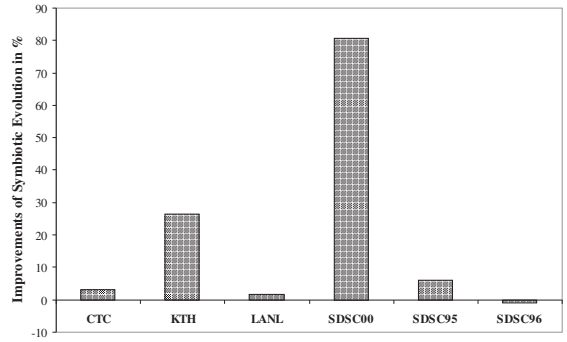

Fig. 5. Improvements of the Symbiotic Evolutionary Approach in Comparison to the Iterative Approach (in %).

iterative optimizations are given. Note that for the SDSC00 workload the Symbiotic evolutionary approach is more than 400 % better than the iterative attempt. Obviously, the Symbiotic evolutionary approach achieves about 20 % better results in mean for the optimization of a given objective function than the iterative approach. Only for the SDSC96 workloads the results are even better applying the iterative approach.

Figure 6 shows the AWRT values of the two preferred user groups. This chart also depicts the mentioned approximated Pareto front of all feasible schedules. Remember that we have 7 simple objectives. Each point within this chart represents a feasible schedule that is not dominated by any other generated feasible solution within the 7-dimensional objective space. As we show only a projection of the actual 7-dimensional Pareto front approximation, the elements are covering an area in this 2-dimensional chart.

As EASY does not favor any user groups, the achieved AWRT values are located in the mid of the projected front area. with our iterative procedure (ITER) it is already possible to move the AWRT values towards the actual front. Obviously, this approach is capable to improve $AWRT_1$ and $AWRT_2$ significantly. However, our Symbiotic approach (SYMB) almost reaches the front in our example. Thereby, is clearly outperforms the iterative approach.

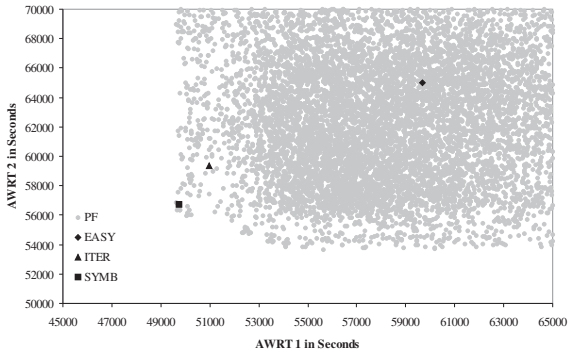

Fig. 6. $AWRT_1$ versus $AWRT_2$ of the Iterative, the Symbiotic Evolution Approach, the EASY Standard Algorithm, and the Approximated Pareto Front of all Feasible Schedules for the CTC Workload.

Finally, a rule base has been generated with the Symbiotic evolutionary approach using a single workload. This rule base has been applied in the following to all other workloads. In Figure 7 the improvements of the objective function in comparison to EASY are presented. The rule base has been optimized using the CTC workload. Although the system was not generated for the other workloads it achieves good results. This behavior provides some evidence that the resulting rule bases can be applied in other scenarios and hence can be called robust.

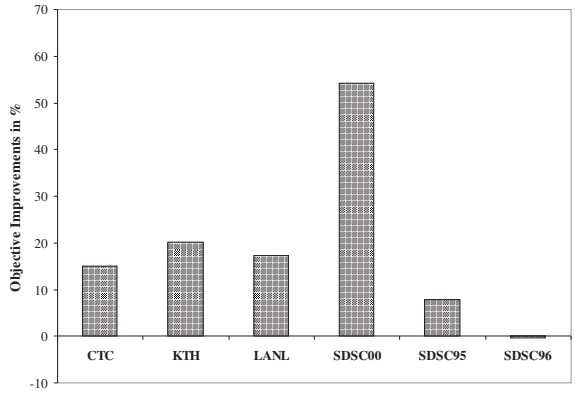

Fig. 7. Improvements of the Objective Function in Comparison to EASY Achieved by the Symbiotic Evolutionary Approach Trained only for the CTC Workload (in %).

6. Conclusion

In this paper, we have presented a methodology for automatically generating online scheduling strategies for a complex objective defined by a machine provider. To this end, we described two methodologies for the development of rule based scheduling systems. A rule system classifies all possible scheduling states and assigns a corresponding scheduling strategy.

For the first approach, we applied an iterative method, that assigns a standard scheduling strategy to all system states while for each rule, the best method is extracted individually. The second approach uses a Symbiotic evolutionary algorithm, that varies the parameter of Gaussian membership functions to establish the different rules.

For both approaches, the performance has been evaluated by simulations with different workloads. The iterative approach is easy to implement but depends on a-priori knowledge about adequate divisions of the situation describing features. All features are divided statically and cannot be modified during the optimization process. So, no automatic adjustment of feature partitions is provided which leads to very bad results for some of the simulated workloads.

The Symbiotic evolutionary approach is able to generate the rule base without any expert knowledge, just by performing evaluations. However, the implementation is more complex. The Symbiotic approach provides more flexibility with respect to different workloads and objective functions. Furthermore, this approach clearly outperforms the iterative version with respect to the resulting objective functions values. We further compared our approaches with the offline generated Pareto front of all feasible schedules which can be considered as a reference for the best achievable results for a certain objective. Here, we are able to almost reach this front with the proposed Genetic Fuzzy systems.

Finally, also the robustness of the evolutionary developed rule bases can be considered from simulation results. The work at hand presented one possible approach of automatically generating appropriate scheduling strategies for various complex provider objectives. The work could not be compared with any other methods, as the authors are not aware of such similar systems.

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

References

[1] S. Albers. Online algorithms: A survey. *Mathematical Programming*, 97:3–26, 2003.

[2] T. Bäck and H.-P. Schwefel. An Overview of Evolutionary Algorithms for Parameter Optimization. *Evolutionary Computation*, 1(1):1–23, 1993.

[3] N. Beume, M. Emmerich, C. Ernemann, L. Schönemann, and U. Schwiegelshohn. Scheduling Algorithm Development based on Complex Owner Defined Objectives. Technical Report CI-190/05, University Dortmund, Germany, 2005.

[4] H.-G. Beyer and H.-P. Schwefel. Evolution Strategies – A Comprehensive Introduction. *Natural Computing*, 1(1):3–52, 2002.

[5] A. Bonarini. Evolutionary Learning of Fuzzy rules: competition and cooperation. In W. Pedrycz, editor, *Fuzzy Modelling: Paradigms and Practice*, pages 265–284. Kluwer Academic Press, 1996.

[6] R. Buyya, D. Abramson, and J. Giddy. An evaluation of economy-based resource trading and scheduling on

computational power grids for parameter sweep applications. In S. Hariri, C.A. Lee, and C.S. Raghavendra, editors, *The 2nd Workshop on Active Middleware Services (AMS 2000), In conjuction with Ninth IEEE International Symposium on High Performance Distributed Computing (HPDC 2000), Pittsburgh*. Kluwer Academic Press, Norwell, 2000.

[7] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger. Economic Models for Resource Management and Scheduling in Grid Computing. *Special Issue on Grid Computing Environments, The Journal of Concurrency and Computation: Practice and Experience (CCPE)*, 14:1507–1542, 2002.

[8] R. Buyya, D. Abramson, and S. Venugopal. The Grid Economy. *Special Issue on Grid Computing, Proceedings of the IEEE*, 93(3):698–714, 2005.

[9] S. J. Chapin, W. Cirne, D. G. Feitelson, J. P. Jones, S. T. Leutenegger, U. Schwiegelshohn, W. Smith, and D. Talby. Benchmarks and standards for the evaluation of parallel job schedulers. In D. G. Feitelson and L. Rudolph, editors, *Proceedings of the 5th Job Scheduling Strategies for Parallel Processing*, volume 1659 of *Lecture Notes in Computer Science (LNCS)*, pages 67–90. Springer, 1999.

[10] K. Deb. *Multi-Objective Optimization using Evolutionary Algorithms*. John Wiley & Sons, Chichester, UK, 2001.

[11] C. Ernemann, B. Song, and R. Yahyapour. Scaling of Workload Traces. In D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, *Proceedings of the 9th Job Scheduling Strategies for Parallel Processing*, volume 2862 of *Lecture Notes in Computer Science (LNCS)*, pages 166–183. Springer, 2003.

[12] 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.

[13] D. G. Feitelson. Memory usage in the LANL CM-5 workload. In D. G. Feitelson and L. Rudolph, editors, *Proceedings of the 3rd Job Scheduling Strategies for Parallel Processing*, volume 1291 of *Lecture Notes in Computer Science (LNCS)*, pages 78–94. Springer, 1997.

[14] D. G. Feitelson. Metric and Workload Effects on Computer Systems Evaluation. *Computer*, 36(9):18–25, 2003.

[15] D. G. Feitelson. Parallel Workload Archive. http://www.cs.huji.ac.il/labs/parallel/workload/, May 2006.

[16] 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, *Proceedings of the 1st Job Scheduling Strategies for Parallel Processing*, volume 949 of *Lecture Notes in Computer Science (LNCS)*, pages 337–360. Springer, 1995.

[17] D. G. Feitelson and A. M. Weil. Utilization and Predictability in Scheduling the IBM SP2 with Backfilling. In *Proceedings of the 12th International Parallel Processing Symposium and the 9th Symposium on Parallel and Distributed Processing*, volume 12, pages 542–547. IEEE Computer Society Press, 1998.

[18] L. J. Fogel. Autonomous automata. *Industrial Research*, 4:14–19, 1962.

[19] M. Garey and R. L. Graham. Bounds for multiprocessor scheduling with resource constraints. *SIAM Journal on Computing*, 4(2):187–200, 1975.

[20] M. Garey and D. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, W. H. and Company, 1979.

[21] R. Gibbons. A historical application profiler for use by parallel schedulers. In D. G. Feitelson and L. Rudolph, editors, *Proceedings of the 3rd Job Scheduling Strategies for Parallel Processing*, volume 1291 of *Lecture Notes in Computer Science (LNCS)*, pages 58–77. Springer, 1997.

[22] R. L. Graham. Bounds for certain multiprocessor anomalies. *Bell System Technical Journal*, 45:1563–1581, 1966.

[23] R. L. Graham, E. L. Lawler, J. K. Lenstra, and A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: A survey. *Annals of Discrete Mathematics*, 15:287–326, 1979.

[24] L. Hall, D. Shmoys, and J. Wein. Scheduling to minimize average completion time: Off-line and on-line algorithms. In *Proceedings of the 7th SIAM Symposium on Discrete Algorithms*, pages 142–151. SIAM, Philadelphia, 1996.

[25] F. Hoffmann. Evolutionary Algorithms for Fuzzy Control System Design. *Proceedings of the IEEE*, 89(9):1318–1333, 2001.

[26] J. H. Holland. Outline for a logical theory of adaptive systems. *Journal of the ACM*, 9:297–314, 1962.

[27] S. Hotovy. Workload Evolution on the Cornell Theory Center IBM SP2. In D. G. Feitelson and L. Rudolph, editors, *Proceedings of the 2nd Job Scheduling Strategies for Parallel Processing*, volume 1162 of *Lecture Notes in Computer Science (LNCS)*, pages 27–40. Springer, 1996.

[28] Y. Jin, W. von Seelen, and B. Sendhoff. On Generating FC^3 Fuzzy Rule Systems from Data Using Evolution Strategies. *IEEE Transactions on System, Man and Cybernetics*, 29(6):829–845, 1999.

[29] C.-F. Juang, J.-Y. Lin, and C.-T. Lin. Genetic Reinforcement Learning through Symbiotic Evolution for Fuzzy Controller Design. *IEEE Transactions on System, Man and Cybernetics*, 30(2):290–302, 2000.

[30] T. Kawaguchi and S. Kyan. Worst case bound of an LRF schedule for the mean weighted flow-time problem. *SIAM Journal on Computing*, 15(4):1119–1129, 1986.

[31] C. B. Lee, Y. Schwartzman, J. Hardy, and A. Snavely. Are user runtime estimates inherently inaccurate? In *Proceedings of the 10th Job Scheduling Strategies for Parallel Processing*, volume 3277 of *Lecture Notes in Computer Science (LNCS)*, pages 253–263. Springer, 2005.

[32] D. Lifka. The ANL/IBM SP Scheduling System. In D. G. Feitelson and L. Rudolph, editors, *Proceedings of the 1st Job Scheduling Strategies for Parallel Processing*, volume 949 of *Lecture Notes in Computer Science (LNCS)*, pages 295–303. Springer, 1995.

[33] A. W. Mu'alem and D. G. Feitelson. Utilization, Predictability, Workloads, and User Runtime Estimates in Scheduling the IBM SP2 with Backfilling. *IEEE Transanctions on Parallel & Distributed Systems*, 12(6):529–543, 2001.

[34] J. Paredis. Coevolutionary Computation. *Artificial Life*, 2(4):355–375, 1995.

[35] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. PhD thesis, TU Berlin, 1971.

[36] I. Rechenberg. *Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog, Stuttgart, 1973.

[37] H.-P. Schwefel. *Evolutionsstrategie und numerische Optimierung*. PhD thesis, TU Berlin, 1975.

[38] H.-P. Schwefel. *Evolution and Optimum Seeking*. John Wiley & Sons, New York, 1995.

[39] U. Schwiegelshohn and R. Yahyapour. Improving first-come-first-serve job scheduling by gang scheduling. In D. G. Feitelson, L. Rudolph, and U. Schwiegelshohn, editors, *Proceedings of the 4th Job Scheduling Strategies for Parallel Processing*, volume 1459 of *Lecture Notes in Computer Science (LNCS)*, pages 180–198. Springer, 1998.

[40] U. Schwiegelshohn and R. Yahyapour. Fairness in parallel job scheduling. *Journal of Scheduling*, 3(1):297–320, 2000.

[41] S. F. Smith. *A Learning System Based on Genetic Adaptive Algorithms*. PhD thesis, Department of Computer Science, University of Pittsburgh, 1980.

[42] B. Song, C. Ernemann, and R. Yahyapour. User Group-based Workload Analysis and Modeling. In *Proceedings of the International Symposium on Cluster Computing and the Grid (CCGRID2005)*. IEEE Computer Society Press, 2005. CD-ROM.

[43] H. Späth. *The Cluster Dissection and Analysis: Theory, FORTRAN Programs, Examples*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1985.

[44] S. Webster. A priority rule for minimizing weighted flow time in a class of parallel machine scheduling problems. *European Journal of Operational Research*, 70:327–334, 1993.

[45] K. Windisch, V. Lo, R. Moore, D. G. Feitelson, and B. Nitzberg. A comparison of workload traces from two production parallel machines. In *6th Symp. Frontiers Massively Parallel Computing*, pages 319–326. IEEE Computer Society Press, 1996.