Proceedings of the 2011 Winter Simulation Conference
S. Jain, R. R. Creasey, J. Himmelspach, K. P. White, and M. Fu, eds.

SIMULATION-BASED OPTIMIZATION FOR GROUPS OF CLUSTER TOOLS IN SEMICONDUCTOR MANUFACTURING USING SIMULATED ANNEALING

Tobias Uhlig
Oliver Rose

Institute of Applied Computer Science
Dresden University of Technology
Dresden, 01062, GERMANY

ABSTRACT

Simulation-based optimization is an established approach to handle complex scheduling problems. The problem examined in this study is scheduling jobs for groups of cluster tools in semiconductor manufacturing including a combination of sequencing, partitioning, and grouping of jobs with additional constraints. We use a specialized fast simulator to evaluate the generated schedules which allows us to run a large number of optimization iterations. For optimization we propose a simulated annealing algorithm to generate the schedules. It is implemented as a special instance of our adaptable evolutionary algorithm framework. As a consequence it is easy to adapt and extend the algorithm. For example, we can make use of various already existing problem representations that are geared to excel at certain aspects of our problem. Furthermore, we are able to parallelize the algorithm by using a population of optimization runs.

1 INTRODUCTION

This paper introduces an approach to generate schedules for tool groups in semiconductor manufacturing. For this study we focus on groups of cluster tools, that show a very complex behavior. To our knowledge there is little previous research regarding efficient scheduling of jobs for cluster tools on a tool group level. A previous approach by Dümmler (1999) showed promising results, but was limited by the employed simulator to small problem instances. Solving larger problem instances, which can be found in semiconductor manufacturing, is a high-ranking goal of our studies. Since semiconductor manufacturing is one of largest industries in the world, there is great interest in efficient production in this field. Improving the operational process can significantly increase the effectiveness of wafer fabrication and thereby reduce the production costs.

The complexity of real world problems is in most cases too high to employ conventional analytical methods and therefore optimization heuristics are used (Weigert et al. 2006). A common approach to scheduling in semiconductor manufacturing is the use of dispatching rules (Varadarajan and Sarin 2006). These rules simply order waiting jobs according to certain criteria. With this simple approach one can generate schedules very fast. This is of great interest in the field of semiconductor manufacturing since the observed problems are very dynamic. Many random events prevent us from determining a valid long term schedule. The most common of these events are frequent machine breakdowns and lot sampling. The main disadvantage of dispatching rules is their static nature. Although they are able to react very fast to dynamic changes, enforcing a simple order of jobs often leads to suboptimal schedules. Dispatching rules are not able to identify and avoid these suboptimal solutions. Furthermore one has to adapt or completely change the employed dispatching rules whenever the scheduling objective changes. Simulation-based optimization is a much more flexible approach and ultimately it may be necessary to improve the effectiveness of wafer fabrication (Pfund et al. 2006).

In this study, we employ simulation-based optimization to generate the schedules for groups of cluster tools, effectively combining the benefits of simulation and an iterative optimization method (see Fig. 1). To use an iterative optimization method it is necessary to evaluate the solutions it generates during the iterations. However, for our problem there is no simple function to evaluate the schedules since the use of cluster tools results in sequence-dependendt process times. Obviously it is not feasible to test each schedule using the actual production system. Therefore Simulation is employed to circumvent this problem. Using simulation we can evaluate a huge amount of possible solutions, limited only by the available computation time. We employ a simple but fast simulator, which is optimized with regard to our problems. It is an event based discrete simulator using an abstract mathematical model for tool groups and schedules. It will be introduced in the next section. Only with fast optimization routines we can effectively compete with dispatching rules. With an adequate simulator we can use one of many different available iterative optimization methods. For our current research we employ simulated annealing.

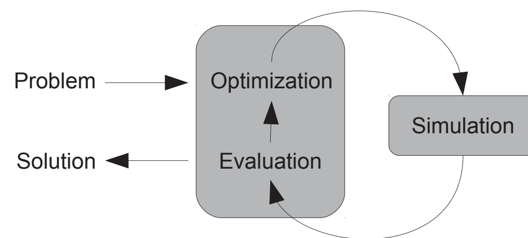

Figure 1: The basic optimization cycle using simulation to evaluate schedules generated by the optimizer.

Simulated annealing (SIMA) is a basic optimization algorithm which is easy to develop. In fact, we already used a simple implementation in our previous studies mainly for reference purposes (Uhlig and Rose 2010, 2011). Simulated annealing performed very well. Hence, we decided to further investigate its capabilities. We use an evolutionary algorithm framework, implemented by us to generate a more sophisticated SIMA algorithm which will be presented in this paper. We will give a short overview of simulated annealing in general and then we will show how we can adapt our generic evolutionary algorithm to obtain a SIMA algorithm (see Section 2). Furthermore, we will show how we can extend the basic idea of SIMA using the capabilities of our framework. Finally, we will present some preliminary results of tests using the new algorithms and will discuss their merits and future developments.

1.1 Model and Simulator

Our model consists of two basic objects: jobs and tools. Regarding semiconductor manufacturing a job is a lot containing multiple wafers. Generally a job (j) is characterized by its recipe and multiple dates. Of those dates the release date, the due date, and the deadline are predefined and part of the problem description. All other dates are updated during simulation (i.e., time the job entered the tool, process start time, and completion time).

$$j := (\underbrace{recipe, weight, t_{release}, t_{due}, t_{deadline}}_{\text{predefined parameters}}, \underbrace{t_{enter}, t_{start}, t_{completion}}_{\text{updated during simulation}}) \quad (1)$$

A discrete event simulator calculates the production dates of a job (j) on a certain tool and updates the job (j') accordingly.

$$j \xrightarrow{\text{simulation}} j' \quad (2)$$

A tool is a resource that can process jobs. The jobs are processed in an order which is defined by a job sequence. A job sequence must not be confused with a queue in front of a machine. A queue contains

the jobs currently waiting for processing while the sequence is a definition in which order jobs should be processed. If for example job B is the only job currently in the queue of an idle machine, but job A (which is not yet released) is in front of B in the sequence, the machine will not process job B before A although it is idle. The order in the job sequence is always enforced. In general, we assume that there is no limitation to the available space in the queues in front of a tool. However, it is possible to monitor the queue length for simulated schedules and reject a solution whenever a certain limit is violated. For simulation a machine (M) is applied to a job sequence - a vector of jobs ($\vec{j}$) - to generate an vector containing processed jobs (see 3).

$$\vec{j}' := M \times \vec{j} \tag{3}$$

There are three basic types of tools in semiconductor manufacturing: simple tools processing one job at a time, batch tools and cluster tools (Kohn and Rose 2011). Our tool model is based on lookup tables which contain the information to calculate the processing times of jobs according to their recipe. These lookup tables are the raw-processing times table (T_{rpt}), the setup times table (T_{stp}), and the slow-down factors table (T_{sdf}).

$$M \begin{cases} simpletool & := (T_{rpt}, T_{stp}) \\ batchtool & := (batchsize, T_{rpt}, T_{stp}) \\ clustertool & := (loadports, T_{rpt}, T_{sdf}) \end{cases} \tag{4}$$

A simple tool is modeled using two tables. The first table (T_{rpt}) contains the raw processing time for all applicable recipes. A second optional table contains the setup times for that machine. For simulation, a simple tool takes one job at a time and calculates its processing time. This is repeated until all jobs are simulated.

$$\begin{aligned} t_{enter} = t_{start} &:= max(t_{now} + T_{stp}(j_{n-1}, j_n), t_{release}) \\ t_{completion} &:= t_{start} + T_{rpt}(j_n) \end{aligned} \tag{5}$$

A batch tool can process multiple jobs at the same time. This is realized by grouping a number of compatible jobs to a batch and process them in parallel. For our purposes jobs are compatible if they have the same recipe. The number of parallel jobs is limited by the batch size. The calculation of the job dates is analogous to a single machine, only the release time is replaced by the batch-release time. The batch-release time equals the maximum release time of all jobs in a batch since processing can only start when all jobs are ready. To group jobs for batching we can use two approaches. The first one is automated grouping. As long as the next job in the sequence is compatible with the jobs currently in the batch and the batch size is not reached we add it to the batch. Explicit batching can be enforced by inserting special dummy jobs (with process time = 0) in the sequence to mark the end of a batch. A second approach is to assign the lots to special batch-jobs which are basically containers for a group of jobs.

Cluster tools are very complex tools which can process parallel jobs. Their processing behavior depends on the recipes of the currently handled jobs. The next section presents a brief introduction to the modeling of cluster tools using slow down factors and the matrix prediction method. In contrast to simple tools cluster tools lead to sequence dependent process times. Therefore it is much more complex to generate efficient schedules for them. Considering for example the minimization of the makespan (minimum completion time of all jobs) without any further constrains. For a simple tool this is trivial since all solutions have the same makespan. However for a cluster tool this problem is NP-hard (Uhlig and Rose 2010).

Having defined jobs and machines we can build a basic schedule by assigning a job sequence to one machine. During simulation the jobs are processed in the order given by the sequence using the dedicated tool. This basic schedule is called an atomic schedule. More sophisticated schedules can be composed from these atomic schedules. If we deal with parallel tools in a tool group the resulting parallel schedule will contain one atomic schedule for each tool. Consequently the problem we face is to assign jobs to the

available machines and to define a certain job sequence for each machine. The algorithms we employ are functions mapping a set of jobs and a set of machines of a tool group to a schedule. Since we use iterative optimization methods we can also use an existing initial schedule as input for the algorithms.

Whenever an algorithm generates a schedule we simulate it with our event based simulator. After simulation we analyze the results using a schedule evaluator which computes certain job performance indicators. According to a given objective function we calculate a value representing the quality of the schedule. For example we may determine the makespan (C_{max}) by comparing all completion times of the jobs in a schedule. Analogously we may determine the maximum Lateness (L_{max}), the total weighted tardiness (TWT), or other objectives. The algorithm uses this feedback to decide whether changes to the schedule should be accepted or rejected.

1.1.1 Cluster Tools

Cluster tools are integrated tools that are used in semiconductor manufacturing. They combine many conventional tools into one machine, which reduces transport times and clean room space. With less transport one can save time and it results in better production quality since wafers stay in vacuum the whole processing time. The basic setup of a cluster tool consists of process chambers, that are arranged around a vacuum mainframe (see Fig. 2). Furthermore, there are load ports which hold lots and one or more robots for transportation between chambers and load ports. Wafers from the lots are processed in the cluster tool visiting different process chambers according to a given recipe. Multiple Wafers can be processed in parallel to utilize all chambers. Essentially a cluster tool is an integrated job shop environment. Pipelining and parallelization improve the performance in comparison to sequential processing on separate tools. However wafers slow each other down since they compete for limited resources. The mutual competition between wafers results in changes of the processing time of lots depending on the current load port recipe mix. As a consequence we can observe sequence-dependent process times since the order of jobs influences the resulting recipe mixes which result in different slow downs for the lots. We can model this using slow down factor to adapt the processing times. Using the raw process time (RPT) for a given recipe and the slow down factor (SDF) depending on the recipe and the current load port recipe mix we can calculate the predicted processing time.

$$\tilde{t}_{process} = RPT(recipe) \cdot SDF(recipe, recipemix) \tag{6}$$

However, we need to recalculate the prediction for all lots whenever the recipe mix changes. Employing the matrix prediction method we can quickly calculate the production dates for all jobs processed in a cluster tool. The method is based on SDFs and successive updated processing times. A detailed description of the matrix prediction method and more details on cluster tools can be found in (Unbehaun and Rose 2007).

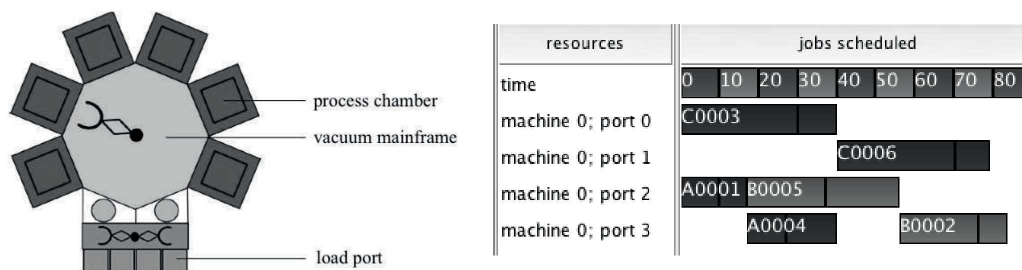

Figure 2: A basic cluster tool with 4 load ports and a gantt chart of jobs processed on it.

1.2 Introduction to Simulated Annealing

Simulated Annealing (SIMA) is a probabilistic heuristic for optimization problems. It is inspired by annealing processes in metallurgy. This process is based on a controlled cooling of molten material. Atoms of the metal establish crystals and reduce the occurrence of defects, effectively minimizing the internal energy. Reaching a certain temperature, the atoms are basically stuck in fixed positions. This state is a local optimum according to the minimization of the internal energy. Abrupt cooling leads to irregular crystallization resulting in configurations which are suboptimal. An appropriate amount of heat enables the atoms to leave their current positions of local minimal energy and wander randomly through states of higher energy. While the metal cools down slowly, the atoms have an increased chance of settling in configurations with lower internal energy.

Kirkpatrick et al. (1983) were the first to map this annealing process to optimization problems. An adaptation of the Metropolis-Hastings-Algorithm (Hastings 1970) became the simulated Annealing Algorithm. The main idea of the algorithm is that the elements in the search space of an optimization problem are equivalent to the atom configurations in a material. Hence finding a good solution can be seen as the process of establishing a state with low internal energy. The algorithm starts with an initial solution as its current configuration. In each iteration a new configuration is derived from the current one. This is done by applying a function to the current state, that applies some random changes to the present configuration and returns the changed state. A probabilistic decision is made whether the new or the old state is used for the next iteration. This decision is skewed to favor better solutions. Hence given multiple iterations the algorithm moves towards an improved solution. The actual probability of accepting a transition from the current state (s) to the derived state (s') depends on the internal energy of the states $(E(s))$ and the current temperature (T). It is calculated with an equation by Kirkpatrick et al. (1983):

$$P(s,s',T) = \begin{cases} 1 & \text{if } E(s) \geq E(s') \\ \exp\left(\frac{E(s)-E(s')}{T}\right) & \text{otherwise} \end{cases} \tag{7}$$

Per definition an improvement (state of lower energy) is always accepted while a regression is accepted with a certain probability depending on the current temperature and the energy difference. Allowing regression under certain circumstances has the benefit, that the algorithm is able to escape suboptimal local optima. In the beginning the temperature is high and even large regressions are possible. In the final stages, with a temperature close to zero, only improvements are accepted. At this point the algorithm turns into a hill climbing algorithm (since the internal energy is minimized it searches the deepest valley instead of the highest mountain). SIMA uses an annealing schedule to emulate the cooling during the annealing process. Regarding problems with a finite search-spaces Granville et al. (1994) showed that the probability of determining a global optimum with simulated annealing will eventually approach one given that the annealing schedule is extended appropriately. However for real world applications the iterations required to guarantee a global maximum exceed the iterations used in a brute force approach. Although we cannot warrant an optimal solution SIMA has been successfully applied to many problems.

2 THE SIMULATED ANNEALING ALGORITHM

A simulated annealing algorithm is a special instance of an evolutionary algorithm. In general an evolutionary algorithm uses selection and variance operators to generate a solution over multiple generations. This is essentially what simulated annealing does. Traditionally evolutionary algorithms are inspired by natural evolution while SIMA is inspired by metallurgy. However, Table 1 shows how the different metaphors correspond to equivalent domain specific entities in scheduling. An atom configuration representing a schedule can be seen as an individual which encodes a schedule. Both are just abstract representations of a possible solution. Atoms and genes are just different metaphors for the jobs which should be scheduled. Transition from one state to the next is equivalent to reproduction of an individual. Selection of states or individuals is based on probabilistic functions which are skewed to favor better solutions in both cases.

Regarding evolutionary algorithms this skewness is called selection pressure. In SIMA this pressure directly depends on the changing temperature. In a nutshell, the classic SIMA is an evolutionary algorithm with a population limited to one individual and a temperature dependent selection function. The next chapter introduces our evolutionary algorithm framework and discusses the necessary adaptation to build a simulated annealing algorithm.

Table 1: Equivalences between Simulated Annealing and Evolutionary Algorithms and their relation to the scheduling domain.

Scheduling Domain	**Simulated Annealing**	**Evolutionary Algorithm**
Schedule	State/configuration	Individual
Generation of new schedules	Transitions between states	Reproduction (using mutation)
Jobs in Schedule	Atoms	Genes
Quality of Schedule according to given objective(s)	Internal Energy	Fitness
Similarity of schedules	Neighborhood relation	Genetic relatedness
Algorithm control parameter	Temperature	Selective Pressure

2.1 The Evolutionary Algorithm Framework

Our framework is an object-oriented extensible framework to build evolutionary algorithms. It can be adapted to implement many different optimization heuristics (i.e., random walk algorithms, genetic algorithms, particle swarm optimization and, of course, simulated annealing). The most important part of the framework are the individuals. They contain most the of algorithm specific information and all of the problem related information. Different implementations of individuals can be applied to various problems and use different strategies to handle them. Each individual supports three basic methods: *getGenotype*, *getPhenotype*, and $generateOffspring$.

The *getGenotype* method allows us to access the genotype of an individual. The genotype is a problem-specific representation encoding a possible solution for the problem. Inspired by natural evolution, we often refer to the genotype as DNA. In general, the genotype is not observable. Accessing it is only necessary during the generation of offspring whenever recombination is employed. During recombination an individual selects one or more mates to generate an offspring derived from the combined genetic information of all of them. The implementation of the recombination can vary between different individuals ranging from different crossover methods to certain goal driven adaptation methods employed for particle swarm optimization.

The *getPhenotype* method returns the phenotype of an individual, which is its observable characteristic. Basically the phenotype is the proposed solution to a certain problem. In our case the phenotype is a schedule, which can be simulated and evaluated. Evaluating the phenotype, we determine the fitness of an individual. It is important to note, that our approach does not assign a fixed fitness value to an individual. The fitness is just a function applied to the phenotype by an observer. This observer may be an individual selecting a mate or a global selection method which selects individuals for reproduction or survival. It is not necessary that all entities in our framework use the same fitness function (e.g., individuals in a population may select their partners using different criteria).

Finally, to create a new individual the $generateOffspring$ method is employed. An individual may use mutation, recombination or any other method to generate an offspring. The derived offspring represents a new solution to a given problem. The core of our framework is a central loop containing steps which are universally valid for all iterative optimization methods. Each iteration in the loop represents one generation. For each generation three basic steps are performed.

1. The **ReproductionSelector** selects individuals (parents) from the current population.
2. Each parent generates one offspring.
3. The **SurvivorSelector** forms a new population from current population and offspring.

} *Generation*

During the first step of each iteration, we select parents which are members of the current population representing promising solution. Different versions of the *ReproductionSelector* method may implement different strategies for selecting the parents. In general, the method has a bias to prefer fitter individuals. The *ReproductionSelector* may have its own independent fitness function to evaluate the individuals. An individual may be chosen more than once for reproduction in one generation. In the second step each parent generates one child which is added to the offspring. If an individual was chosen more than once as parent it will generate one child for each time it has been chosen.The final step uses a *SurvivalSelector* to generate the population for the next iteration. It selects the individuals from the existing population and the newly generated children. Just like the *ReproductionSelector* it generally has a bias towards fitter individuals and can have its own evaluator. The previously mentioned optimization algorithms are realized by using a certain type of individuals and employing the appropriate selection methods.

2.2 Implementing Simulated Annealing

To implement a plain simulated annealing algorithm we do not need a special kind of individual. Any individual that supports mutation for the generation of offspring is sufficient. This mutation is equivalent to a transition between states during the annealing process. Therefore the employed mutation method implicitly defines a neighborhood-relation between states. We implemented already three different encoding strategies for the scheduling problem.

- Individual A uses random indexes to encode the order of jobs. The mutation operator randomly switches two random indexes or shifts one index. Partitioning of jobs to multiple machines is achieved by employing special marker jobs with no processing time. These jobs act as cutting point in a global job sequence and each resulting subsequence is assigned to a tool.
- Individual B also uses random indexes to encode the order of jobs, but additionally assigns a number to each job encoding the tool this job should be processed on. The mutation operator of individual A is extended to include random reassignments of jobs to new different tools.
- Individual C uses a recipe pattern to encode the position of jobs in a sequence according to their recipe. Mutation is implemented as a random switch of recipe values in the pattern. This individual uses a rigid partitioning approach by enforcing approximately equisized subsequences for all tools.

Regarding the *ReproductionSelctor* method implementing simulated annealing is trivial. The population size for simulated annealing is one, hence there is only one individual which can be selected as parent. This individual represents the current state of the algorithm. According to the algorithm exactly one child is generated. The *SurvivalSelector* is a little more complex. It compares the fitness of the parent and child and probabilistically selects one for the next generation. The decision is based on Equation 7 and depends on the temperature given by an annealing schedule. This approach results in three implementations of a plain simulated annealing algorithm using these three species. In the next section, we will propose some extensions to this basic algorithm.

2.3 Extending the Algorithm

Our framework supports population size larger than one. Hence, we are able to generate a parallel simulated annealing algorithm (pSIMA). During each iteration all individuals are selected in same order. Each individual generates one offspring. During survivor selection we employ a coupled selection method which compares each parent directly with its offspring. The selection probability is calculated in exactly the

same way as the plain SIMA. Essentially this results in parallel runs of SIMA that do not mutually interact. Using the same number of simulation calls as plain SIMA will result in a broader but less deep search. Apart from the added convenience there is no advantage in using pSIMA instead of multiple instances of a plain SIMA.

However, we can further adapt the algorithm by introducing a modified recombination concept. In this case we use recombination for some instances of offspring generation. The probability of recombination is very low (i.e., p=0.01) to not destroy the concept of simulated annealing. Whenever recombination occurs an individual uses a special selector to determine the fittest individual. It then copies the genetic information and generates an exact clone of the fittest individual. Using this approach many individuals independently search the most promising area of the search space.

Going a step further it is possible to use recombination just as for a regular genetic algorithm. This would result in an approach that behaves like a genetic algorithm but that would still use the temperature dependent selection strategy. Consequently, we would have a genetic algorithm with increasing selective pressure. Furthermore the coupled selection method avoids genetic drift. Genetic drift is a phenomenon known from conventional genetic algorithms (Eiben and Smith 2003). It leads to convergence in one local optimum. Avoiding genetic drift facilitates a widely spread search.

Our algorithm framework handles all individuals in a standardized way, without any special consideration of their actual implementation. This allows us to use multiple species (implementations) at the same time. Given the nature of evolutionary algorithms the representation matching the problem best will prevail in this context. This approach was investigated by us with regard to a multi-species evolutionary algorithm (Uhlig and Rose 2011) and leads to very robust results. Using our modular approach, we can think of many other possible extensions to simulated annealing, including new individual implementations, different selection methods, effects like aging of individuals or the use of changing population sizes.

2.4 Parameters

Determining reasonable algorithm parameters for a given problem is a challenging task. Regarding simulated annealing we have to consider two parameters. The first one is the number of performed iterations. Obviously more optimization steps will lead to better results. However eventually the algorithm will converge and further iterations will not lead to significant improvements. We use fixed numbers of iterations for our algorithms based on the idea that there is only a fixed amount of time available for optimization. Other approaches depend on the quality of the generated results. Terminating when a solution is better than a given reference value or whenever there is no further improvement for a given number of iterations. With a fixed number of iterations it is slightly easier to define an appropriate annealing schedule, which is the second parameter of simulated annealing. The annealing schedule dictates the temperature for each step of the algorithm. Effectively it assigns a certain selection pressure to each iteration of the optimization loop. We use an exponential cooling scheme proposed by Kirkpatrick et al. (1983). For this annealing schedule the temperature is calculated recursively using an absolute term α, which is usually close to, but smaller than, 1. Given the current temperature the temperature for the next iteration is calculated as:

$$T_{n+1} = T_n \cdot \alpha, \qquad \underbrace{0 < \alpha < 1}_{\text{cool down factor}} \tag{8}$$

Given an initial temperature T_0 we can therefore calculate the temperature for any iteration using the following equation:

$$T_n = T_0 \cdot \alpha^n \tag{9}$$

Using this annealing scheme we generally observe three phases of simulated annealing (see Fig. 3). The first phase is characterized by a high temperature. Therefore regressions are accepted nearly all of the time. Assuming there are more mediocre than good solutions in the search space chances are high that the

algorithms tends to produce mainly suboptimal results in this phase. But fueled by the high temperature we have a great chance to escape bad local optima. A bad local optimum is a point in the search space representing a mediocre or bad solution which is surrounded by even worse solutions making it hard to escape from it. Although there is a slight bias towards better solutions the algorithm often returns to average ones. Since good solutions are surrounded by more average than better solutions we face a very high probability of regression. With decreasing temperature we gradually enter the second phase of the algorithm. This phase is very productive. It is characterized by a stronger bias towards good solutions leading to a more focused exploration of the search space. Nevertheless there is still a good chance to escape suboptimal local optima. Eventually the temperature drops under a certain threshold. In this phase temperatures are so low that only very small or no regressions at all are accepted. At this time the algorithm basically becomes a stochastic hill climbing strategy. If we already reached a local optimum the algorithm is basically stuck. Otherwise the algorithm tries to gradually improve the current solution. Commonly there are only small gains in quality in this phase.

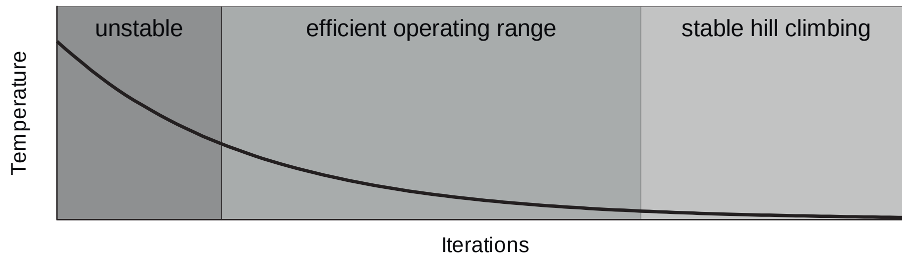

Figure 3: Three phases of simulated annealing algorithm using a exponential cooling scheme. First unstable phase with behavior like random walk. Second phase efficient searching with possibility of escaping local optimum. The stable third phase resembling a hill climbing algorithm.

The challenge is to balance the three phase to generate optimal results. Generally this requires much experience and calibration. However reasonable results can be achieved using some basic guidelines to automatically generate an exponential annealing schedule. A very simple approach would be to start with a very high (too high) temperature and use a huge amount of iterations. In this case we get pretty long phases one and three since we do not know in which temperature range the second phase is situated. The cool down factor (α) is chosen to ensure that the temperature is close to zero in the end. This can be done by using a visual representation of the function to select the appropriate value. A more sophisticated approach is based on certain desired acceptance probabilities. If we define a probability (P) to accept a regression (ΔE) we can use Equation 7 to calculate the necessary temperature.

$$P(\Delta E, T) = \exp\left(\frac{\Delta E}{T}\right) \quad \rightarrow \quad T = \frac{\Delta E}{\ln P} \tag{10}$$

We can use Equation 10 to define the temperature for two points in the annealing schedule. The first point is the initial temperature (T_0). At the beginning of the algorithm we want to be able to accept nearly every regression. To estimate the values of possible regressions we use the standard deviation of a sample consisting random solutions to the given problem. Accordingly we define: $P_0(\sigma(x)) = 0.99$. The second point is the temperature (T_N) at the end of the annealing schedule. Here we will only accept small regressions. We assume that even values as small as 1% of the standard deviation should only be accepted rarely. We define this point as $P_N(\sigma_X \cdot 0.01) = 0.01$.

$$T_0 = \frac{\sigma_X}{\ln P_0} \quad , \quad T_N = \frac{\sigma_X \cdot 0.01}{\ln P_N} \tag{11}$$

Having defined two points in the annealing schedule it is easy to generate the whole schedule starting with the initial temperature and calculating a cool down factor (α).

$$\alpha = \left(\frac{T_N}{T_0}\right)^{\frac{1}{N}} \tag{12}$$

For our extended algorithms, we have to consider further parameters like the number of individuals and the actual implementation chosen for them. There is no easy rule how to determine these parameters appropriately. Regarding parallel individuals, we face a decision between deep or broad search. A possible compromise could be to start with a broad search and later decrease the population size to focus on a few promising individuals. Generally it is worth considering to use at least some parallel individuals to exploit modern multi-core processors, since the simulation routine is single threaded. Determining the right individual implementation is also not easy. If tests in advance cannot determine the best implementation it is worthwhile to consider a multi-species approach.

3 EXPERIMENTS

We performed an extensive analysis to evaluate the implemented algorithms, using more than 10 generated tool groups in various test instances. An representative excerpt of the generated results can be found in Table 2. For this excerpt we used three different test setups with the following properties:

- Tests setup one (ts1) consists of one cluster tool with four load ports. 16 jobs with 8 different recipes (2 jobs for each recipe) have to be scheduled.
- The second test setup (ts2) also included one cluster tool with four load ports. However in this case we had 20 jobs with 5 different recipes.
- The third test setup is the most complex one. However, we had no valid due dates for the jobs, and therefore could use only certain objectives. The setup consists of 4 cluster tools with 4 load ports. One special constraint is machine dedication, since two of the tools were not qualified to process all jobs. The two remaining tools were able to process all jobs, but one tool was slower than the other. This setup contained 40 jobs uniformly distributed to 10 different recipes.

Scheduling was done with respect to three different objectives. The minimization of the makespan (C_{max}) which effectively results in an maximized throughput. The minimization of the maximum Lateness (L_{max}) and minimization of the total weighted tardiness (TWT). Both objective measure the quality of a schedule with regard to meeting the given due dates.

For our test we employed our algorithms to generate schedules for the test setups. All algorithm runs were repeated 100 times to ensure valid results. Algorithms are named using the scheme $[extension.description]SIMA_s$. The optional description signifies the used extension while the index (s) contains the employed individual implementations. Therefore $SIMA_s$ is a plain Simulated Annealing algorithm. $pSIMA_s$ is a parallel simulated annealing using multiple individuals. For this study population size was set to 10. The the multi-species $pSIMA_{abc}$ approach used only three individuals one for each species. For algorithm $pxSIMA_s$ we extend the parallel approach using the discussed recombination strategy. And finally $gaSIMA_s$ is the proposed hybrid of genetic algorithm and simulated annealing. We do not include other simulation-based optimization heuristics as a reference. For L_{max} and TWT we use an earliest due date dispatching rule (EDD) as comparison.

Regarding the results of our test it is easy to see that there is no single best algorithm instance. For different problems different algorithms returned the best results. However algorithms relying on individual implementation B ($SIMA_b$) performed bad in all cases. This may be attributed to a high disruptiveness of the mutation operator which should be reconsidered to generate better results in the future. All other implementations returned very good results and clearly outperformed the simple dispatching heuristic (EDD) which is not able to cope with the complexity of cluster tool scheduling. This can be attributed

Table 2: Results of Tests (best results marked bold). All values are mean results of tests. Each test was repeated 100 time to ensure statistical significance.

Algorithm	n=20000						n=1000					
	C_{max}		L_{max}		TWT		C_{max}		L_{max}		TWT	
	ts1	ts3	ts1	ts2	ts1	ts2	ts1	ts3	ts1	ts2	ts1	ts2
$SIMA_a$	390.6	385.1	16.6	3.3	44.7	10.0	396.7	505.5	**41.7**	10.2	**101.3**	18.3
$SIMA_b$	401.0	467.2	68.7	22.3	212.2	35.1	419.3	499.0	123.8	41.7	390.7	85.1
$SIMA_c$	393.4	**383.2**	35.5	5.3	75.9	10.0	396.1	**458.8**	53.6	11.4	119.0	16.8
$pSIMA_a$	380.6	402.4	**6.9**	**0.1**	**7.7**	**0.2**	397.5	648.8	49.4	11.7	117.7	15.6
$pSIMA_c$	381.0	402.2	22.2	0.4	41.6	1.2	397.2	630.6	54.1	12.4	128.4	17.0
$pxSIMA_a$	378.4	397.7	**6.9**	**0.1**	**7.7**	1.0	397.2	642.9	49.5	11.3	116.8	**15.1**
$pxSIMA_c$	380.9	401.0	22.0	0.2	41.5	1.3	397.5	511.6	52.4	12.3	123.3	16.1
$gaSIMA_a$	**377.5**	397.6	7.0	**0.1**	**7.7**	3.8	396.8	618.2	44.8	11.1	103.1	16.3
$gaSIMA_c$	378.0	397.7	22.3	0.5	40.1	2.3	396.2	498.6	53.5	11.9	128.2	18.5
$pSIMA_{abc}$	387.3	389.1	16.5	2.3	33.8	9.0	**395.7**	508.8	48.2	**9.6**	106.5	16.8
EDD	463.7	-	90.8	81.0	333.8	169.5	463.7	-	90.8	81.0	333.8	169.5

to the fact, that optimizing with regard to L_{max} or TWT still depends very much on finding a schedule with a low makespan to avoid delays for all jobs. The biggest influence to the results can be found if you compare plain SIMA approaches to approaches relying on more than one individual. Obviously the decision between a broad or a deep search strategy has a big influence on the results. But we can not reliably predict for which cases which strategy is better. In general, a deep search strategy is better suited to more complex problems like test setup 3. The complexity of this setup is reflected in the average results for algorithms using only 1000 iterations. In comparison the results for test setup 1 are very good even with a low number of iterations.

In a nutshell, we can see that different implementation have different strengths. This is in accordance with Little's Theorem which predicts that any nonrevisiting black box algorithm solving a certain problem better than other ones must be lacking with regard to other problems. Regarding all possible problems one can on average expect the same performance for all black box algorithms. (Eiben and Smith 2003, 201-202). The only way to circumvent this is to violate the black box criteria and include problem specific knowledge.

4 CONCLUSION AND OUTLOOK

Using simulated annealing for simulation-based optimization is not a revolutionary approach. However, in this particular example we showed how a fast simulator can enable us to successfully apply this technique in a field generally dominated by fast dispatching heuristics. Furthermore, we showed that embracing the evolutionary character of simulated annealing leads us to many possible extensions for the algorithms by combining the established simulated annealing with techniques from the field of evolutionary algorithms. Those new hybrid algorithms showed some potential during our experiments however further evaluation is needed before they can be widely accepted. In general we cannot expect any of the introduced implementations to outperform all other for all scheduling problems. The new algorithms are another tool to tackle difficult problems.

With regard to the future we will further evaluate and extend the algorithms. One important goal is to increase the automatisation of parametrization for the algorithms. Reasonable algorithms that automatically adjust to a given problem or use universally adequate default parameters can comfortably be used in a larger context. For example we consider multi-agent networks for optimization of whole fabs. In this scenario, a single agent uses an evolutionary algorithm to manage the jobs for a tool group. In contrast to local optimization, the agents are able to communicate and adjust their schedules with regard to global interests.

REFERENCES

Dümmler, M. A. 1999, December. "Using Simulation and Genetic Algorithms to Improve Cluster Tool Performance". In *Proceedings of the 1999 Winter Simulation Conference*, edited by P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. Evans, 875–879. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Eiben, A. E., and J. E. Smith. 2003. *Introduction to Evolutionary Computing*. Springer Verlag.

Granville, V., M. Křivánek, and J.-P. Rasson. 1994. "Simulated Annealing: A Proof of Convergence". *IEEE Transactions on Pattern Analysis and Machine Intelligence* 16:652–656.

Hastings, W. K. 1970. "Monte Carlo sampling methods using Markov chains and their applications.". *Biometrika* 57:97–109.

Kirkpatrick, S., J. C.D. Gelatt, and M. P. Vecchi. 1983. "Optimization by Simulated Annealing". *Science* 220:671–680.

Kohn, R., and O. Rose. 2011, December. "Automated Generation of Analytical Process Time Models for Cluster Tools in Semiconductor Manufacturing". In *Proceedings of the 2011 Winter Simulation Conference*, edited by S. Jain, R. R. Creasey, J. Himmelspach, K. P. White, and M. Fu, WSC '11. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Pfund, M. E., S. J. Mason, and J. W. Fowler. 2006. "Semiconductor Manufacturing Scheduling and Dispatching". In *Handbook of Production Scheduling*, edited by J. W. Herrmann, Volume 89, 213–241: Springer US.

Uhlig, T., and O. Rose. 2010. "Solving Scheduling Problems with Sequence-Dependent Process Times by Evolutionary Algorithms". In *Proceedings of the 2010 Industrial Engineering Research Conference*, edited by A. Johnson and A. Miller.

Uhlig, T., and O. Rose. 2011. "A Multi Species Evolutionary Algorithm for Tool Group Scheduling in Semiconductor Manufacturing". In *Proceedings of the 5th Multidisciplinary International Conference on Scheduling: Theory and Applications*, edited by J. Fowler, G. Kendall, and B. McCollum, 459–468.

Unbehaun, R., and O. Rose. 2007, December. "Predicting cluster tool behavior with slow down factors". In *Proceedings of the 2007 Winter Simulation Conference*, edited by S. G. Henderson, B. Biller, M.-H. Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, WSC '07, 1755–1760. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Varadarajan, A., and S. C. Sarin. 2006. "A survey of dispatching rules for operational control in wafer fabrication". *Information Control Problems in Manufacturing 2006* 1:709–720.

Weigert, G., S. Horn, and S. Werner. 2006. "Optimization of manufacturing processes by distributed simulation". *International Journal of Production Research* 44:3677–3692.

AUTHOR BIOGRAPHIES

TOBIAS UHLIG is a PhD student at Dresden University of Technology and a research assistant at the University of the Federal Armed Forces Munich, Germany. He received his M.S. degree in Computer Science from Dresden University of Technology. His research interests include evolutionary computation and its application to scheduling problems. He is a member of the IEEE RAS Technical Committee on Semiconductor Manufacturing Automation. His email address is tobias.uhlig@unibw.de.

OLIVER ROSE holds the Chair for Modeling and Simulation at the Department of Computer Science of the University of the Federal Armed Forces Munich, Germany. He received an M.S. degree in applied mathematics and a Ph.D. degree in computer science from Würzburg University, Germany. His research focuses on the operational modeling, analysis and material flow control of complex manufacturing facilities, in particular, semiconductor factories. He is a member of IEEE, INFORMS Simulation Society, ASIM, and GI, and General Chair of WSC 2012. His email address is oliver.rose@unibw.de.