A Multi Species Genetic Algorithm for Scheduling in Semiconductor Manufacturing

Paper21

xxx
xxx
xxx
xxx

Abstract

In semiconductor manufacturing, we face many complex scheduling problems, which are difficult to tackle with conventional approaches. Genetic algorithms are a reasonable approach to solve these problems, however, oftentimes plain genetic algorithms are prone to premature convergence. This is particularly relevant if we consider multi-criteria and multi-constraint problems, which are ubiquitous in semiconductor manufacturing. A typical problem we investigate is offline scheduling for parallel cluster tools, which effectively is a combination of partitioning and sequencing of jobs with constraints. Each tool can be classified as a job shop environment, which can handle multiple jobs limited by the number of load ports. Regarding cluster tools we have to consider sequence-dependent process times, since jobs processed in parallel compete for resources and slow each other down depending on their recipes. In this paper we try to maximize the throughput of a tool group by minimizing the makespan of the scheduled jobs. We propose an extensible genetic algorithm to cope with this problem. The algorithm adheres to a modular design philosophy, which enables us to customize it in a multitude of ways. It supports multiple species and races, different individual selection strategies and further extensions like aging. A multi species approach we employed generated promising results with regard to aforementioned problem.

Introduction

Genetic Algorithms

Genetic algorithms (GA) belong to the class of Evolutionary Algorithms (EA). They are widely used, to generate solutions to optimization and search problems. For this purpose they rely on repeated application of two fundamental concepts - variation and selection (see Fig. 1). Variation generates new solutions derived from existing ones. It increases the diversity and facilitates the exploration of the search space. The variation operators of EAs are mutation and recombination.

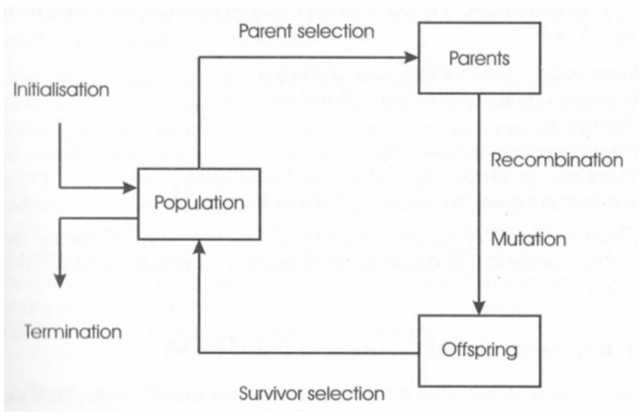


Figure 1: Scheme of an EA (Eiben 07, p. 17)

Selection is the driving force of evolution. Since variation is not goal driven selection is employed to increase the average quality of populations in the course of generations. This is achieved by favoring fitter individuals, which represent better solutions to a given problem. Typical selection methods include selection for reproduction and selection for survival. Fuelled by variation and selection the fitness of individuals gradually improves with passing generations. Obviously this also improves the results generated by the EA.

Natural selection is a mechanism for generating an exceedingly high degree of improbability.

Ronald Aylmer Fisher

Usually GAs generate good results for many complex problems (see Fig. 2).

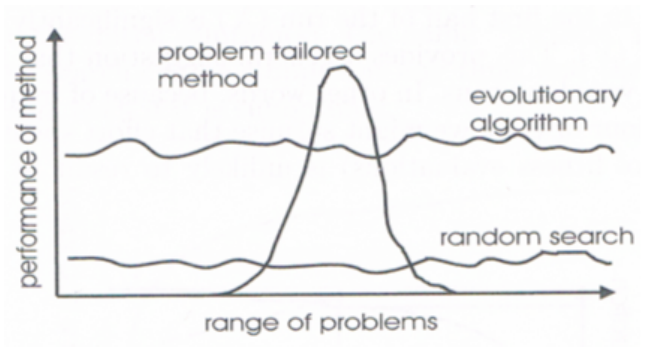


Figure 2: Performance of EAs (Eiben 2007, p. 32)

However, their performance often depends on their ability to avoid premature convergence, caused by genetic drift.

> The population based nature of EAs holds out much promise for identifying multiple optima, however, in practice the finite population size when coupled with recombination between any parents (known as panmictic mix) leads to the phenomenon known as genetic drift and eventual convergence around one optimum. (Eiben 07, p. 155)

This is especially relevant if multimodal problems are considered. Different approaches have been proposed to avoid premature convergence (Eiben 07, p. 158 - 164). A very simple approach is to repeatedly run a simple GA and save the best results. This approach has been improved by employing parallel populations, which exchange individuals at certain points in time. This behavior is inspired by the punctuated equilibria theory (Eldrege 72). Examples for this technique include island model EAs and coarse-grained parallel EAs. The main idea of these algorithms is, that different populations explore diverse niches of the search space and do not converge to a single local optimum. Hence the possibility of discovering the global optimum in one of these niches is increased. Even if the global optimum is not found, populating various niches may lead to a better local optimum in one of them.

The exploration of multiple niches may also be achieved by employing a spatial distribution within a GA. This technique distributes many overlapping subpopulations in the algorithmic space (Husbands 94). Individuals may only mate with other individuals, which are their neighbors or are situated in close proximity. This locality principle leads to a broader distribution of solutions in the search space. Common examples for this method are fine-grain parallel EAs, diffusion model EAs, cellular EAs, or distributed EAs.

Fitness sharing is another method to avoid convergence of individuals to one common optimum. Individuals sharing the same niche have to share their fitness. Consequently individuals tend to be fitter if they have their own niche and do not share it with other ones. This improves diversification significantly. In a similar approach, called crowding, survivor selection is adapted to choose only a limited amount of individuals sharing a niche. This effectively prevents crowding of niches and promotes diverse populations.

Another way to avoid crowding is to implicitly facilitate the spreading of individuals. This can be achieved by favoring new individuals over established individuals. To this end aging in GAs can be employed to reduce genetic drift (Nouioua 09). One possibility is to limit the lifespan of individuals. Consequently a population may not be dominated by a group of elitist individuals since they will die eventually. In their wake new individuals representing promising alternatives may flourish. Simple examples of this approach are GAs using a survival selection method, which does not allow elitism. This effectively limits the lifespan of individuals to one generation. However, this idea should be applied cautious since neglecting elitism implies the potential loss of some very fit individuals. Elitist individuals usually tend to generate very promising offspring. Therefore it might degrade the performance of an algorithm to ignore their genetic information. A larger lifespan reduces this problem. A reasonable approach might assign a greater lifespan to fitter individuals to preserve their genetic code for the following generations. It is also possible to completely ignore explicit survival selection. In this case individuals stay in the population for a number of generations depending on their lifespan. As a consequence the population size varies in the course of generations. An alternative aging approach relinquishes the idea of a given lifespan. Instead it applies some kind of deterioration to the fitness of an individual depending on its age. The fitness of an individual decreases until it is not selected any more as parent or for survival. This approach especially favors recently born individuals with the necessary fitness.

Finally we like to introduce one last method to avoid premature convergence. Speciation is a promising approach to achieve diversification. Different species are restricted from mating. This effectively divides the population into multiple subpopulations, which may explore various niches. A modification to this algorithm uses different races instead of multiple species. These races are not restricted from mating, but they have a bias to select individuals belonging to the same races. This approach is similar to coarse grain parallel algorithms, since there is a certain amount of exchange between subpopulations. Usually species and races are implemented by adding a label to an individual. This label determines to which species/race an individual belongs. For multi species algorithms individuals may only select mates, having the same label. For multi race GAs individuals prefer mates with the same label. This can be called an artificially imposed speciation.

In this paper, we want to introduce a different approach. Instead of artificially labeled species we use individuals, which actually have incompatible DNAs. This is achieved by implementing multiple species, using unique problem representations as genetic code. Previous work has shown the importance off an adequate DNA implementation (Uhlig 09). Some implementations may have an inherent advantage to solve certain scheduling problems. On one hand side this is related to the quality of the genetic code, which models a problem solution. On the other hand varying representations behave different if variance operators are applied. Especially the disruptiveness of crossover may differ for various DNAs.

Oftentimes it is difficult to predict which implementation is suited best for a given problem. A multi species GA can circumvent this problem. When multiple species compete against each other the best representation will eventually prevail. Therefore it may be good to resort to a pool of species instead of relying completely on a single DNA implementation. In this paper we introduce a modular GA, which allows for incompatible individuals (species) in a population.

Scheduling Problems in Semiconductor Manufacturing

Semiconductor manufacturing has become one of the largest industries in the world. Hence there is a great interest in efficient production in this field. In addition to advancements in the fabrication process improving the operational process should significantly increase the effectiveness of wafer fabrication and reduce the production costs. Scheduling is one important method to optimize the operational process. Today usually dispatching rules are utilized in semiconductor manufacturing (Varadarajan 06). However, deterministic scheduling is a promising approach and ultimately may be necessary to further improve the effectiveness of wafer fabrication (Pfund 2006).

This paper will discuss the optimization of schedules for tool groups in a wafer fab. Oftentimes a single tool or tool group is the bottleneck in wafer fabrication. Usually these are very expensive tools operating at a high load. Optimizing the throughput of these bottlenecks will improve the performance of the whole fab and thereby reduce the costs of wafers. In semiconductor manufacturing there are basically three different types of tools. Simple tools processing one job at a time, batch tools and cluster tools. Batch tools can handle a number of parallel jobs. Groups of jobs called batch are processed at the same time. In contrast cluster tools also process parallel jobs, but each job is handled independently. In the context of semiconductor manufacturing a job is a lot containing multiple wafers. The processing time of a job may depend on the number of wafers.

Regarding the three types of tools we can identify some basic scheduling problems. These are sequencing, partitioning, and grouping of jobs. Sequencing deals with the optimal order of jobs, which is relevant for problems with sequence-dependent setup times or sequence-dependent process times. It is also very important for problems with customer oriented objective functions. These functions depend on delivery in time to satisfy certain due dates. The distribution of jobs to parallel machines is called partitioning. Scheduling for batch tools requires grouping of jobs, which are to be processed in a batch. These problems are usually very complex by themselves. However, most actual problems are a combination of more then one of them and therefore exhibit an even greater complexity.

Scheduling problems are often categorized using a $\alpha|\beta|\gamma$ notation by Graham et al. (Graham 79). The α field describes the machine environment, the ß field represents special restrictions and the γ denotes the scheduling objective. According to that notation the problem discussed in this paper is:

$$R \mid lrc \mid C_{max}$$

We face unrelated parallel tools (R) with sequent-dependent process times caused by different load port recipe combinations (lrc), which is typical for cluster tools. The scheduling objective is the makespan (C_{max}). The makespan equates the completion time of the last job and therefore provides a measurement of the throughput. Other common scheduling objectives in semiconductor manufacturing are total weighted Tardiness and maximum Lateness. The given problem is a special instance of Job Shop scheduling in combination with partitioning, which are both known to be NP-hard (Pinedo 02). Hence no algorithm with polynomial time complexity can solve it optimally. Regarding partitioning we also consider dedication. Dedication is a constraint resulting from different machine qualifications. Jobs may only be processed on certain qualified machines.

Cluster Tools

Cluster tools are integrated tools employed in semiconductor manufacturing. A cluster tool is composed of multiple process chambers, arranged around a vacuum mainframe. It processes lots, which usually contain up to 25 wafers. Wafers can be processed in parallel in different process chambers. The number of parallel jobs is limited by the load port count. Load ports are entry points for lots. A robot takes single wafers from a lot in a load port to a chamber were it is processed. A wafer is processed in one or more process chambers according to its recipe. Since different wafers usually compete for limited resources (available process chambers, shared transport robot) they mutually slow each other down (see Figure 2). This effect can be modeled with slow down factors for given recipe mixes. Using a matrix prediction method with these slowdown factors we can predict the completion times for lots processed in a cluster tool (Unbehaun 2007). This method is significantly faster than detailed simulation of a cluster tool and is implemented in our simulator.

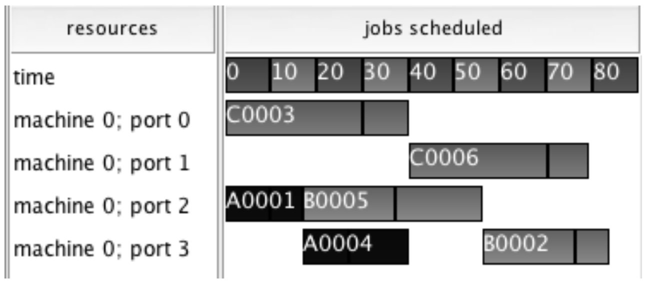


Figure 3: Gantt chart of jobs processed on a cluster tool with four load ports

Methods

The Scheduling Framework

Our framework was implemented to evaluate different metaheuristics for semiconductor scheduling. It consists of three main components, a simulator, schedule evaluation methods and the scheduling algorithms. In general, the different scheduling algorithms call the simulator to run their schedules. Subsequently they ascertain the quality of the generated solutions using the evaluation methods.

The simulation core is a very basic discrete event based simulator. It was developed for rapid simulation of schedules. It lacks nearly all features of commercial simulators and just calculates the processing times for jobs in a given schedule. It basically uses lookup tables assigned to machines to determine the processing time of a job on a machine depending on its recipe. Additionally it also supports tables with sequence dependent setup times for simple tools and batch tools. Cluster tool behavior is simulated with the matrix prediction method (Unbehaun 07) using special lookup tables containing slow down factors for all load port recipe combinations. The simulator can only simulate a schedule for a tool group with parallel tools. The supported tools are simple tools (one job at a time), cluster tools and batch tools. At the moment it does not support sequential process steps and routing of jobs.

Evaluation methods can be applied to processed schedules to calculate different objective functions. Typical objective functions are makespan (C_{max}), total weighted tardiness (TWT), and maximum lateness (L_{max}). The framework currently supports all objective functions relying only on job attributes for calculation. Available job attributes include release dates, due dates, deadlines, weights, processing times, waiting times, and recipes.

The metaheuristics we employ are algorithms, which generate schedules given a test setup. A test setup consists of a number of jobs and parallel tools belonging to one tool group. The resulting schedule contains ordered job sequences allocated to certain machines. This paper discusses a modular genetic algorithm, which is introduced in the next paragraph. Other algorithms we implemented include simulated annealing, hill climbing, ant colony optimization, particle swarm optimization, and many others.

A Modular Genetic Algorithm

To perform our experiments we developed a modular GA. (see Figure 3) In contrast to ordinary GAs it is independent of the actual problem representation and works with any individuals, adhering to a given schema. The individuals are responsible for problem specific details. Methods employed for selection are also designed modularly to allow easy replacement.

```
initialize population
for number of generations do
  update statistics
  parents = Selector(population, n)
  for each parent do
    generate one offspring
    add offspring to children
  end for
  population = SurvivorSelector(population, children)
  store best solution
  apply aging
end for
```

Figure 2: Pseudo code of implementation.

A ***Selector*** chooses n parents from a given population. These parents are used to generate offspring representing new solutions. Usually a Selector favors fitter individuals to apply selective pressure. An individual may be chosen as parent more than once in each generation. A Selector may employ a stochastic or deterministic selection method. Stochastic methods we implemented include among others ranking selection, tournament selection and fitness proportional selection. An example for deterministic method is truncate selection.

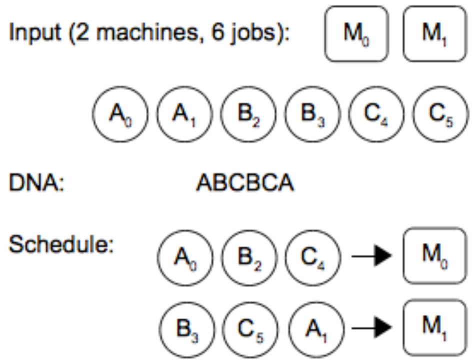
The ***SurvivorSelector*** builds the next generation by selecting individuals from the current population and the children. In contrast to the reproduction selector an individual may only be chosen once for survival. This selection process may once again be stochastic or deterministic. Aside from fitness the SurvivorSelector may use other factors like elitism, age, race, or species for selection.

Resulting from the modular approach we can easily adapt the selection strategy of our algorithm. This raises the responsibility for the experimenter setting up the algorithm since it increases the amount of parameters. Generally a genetic algorithm works if there is some kind of selective pressure in contrast to a purely random selection, but the intensity of selection affects the pace of evolution and the likeliness of premature convergence. A high selective pressure promotes the elitist individuals causing a very fast development. However, it also increases the likelihood of genetic drift towards singe local optima. That is why it is important to utilize a well-balanced selection strategy to ensure a diverse population.

A population consists of a number of unique individuals, which may belong to different species or races. The individuals we employ provide solutions to a given scheduling problem. This solution is their phenotype, generated depending on their genotype. The genotype is a problem dependent representation usually called DNA. Individuals of the same species share a common DNA representation. Individuals of the same race furthermore share a common generator logic for schedules and identical parameters. Races belonging to one species may differ in many characteristics but they remain compatible with regard to recombination. That said most of the times individuals of certain race have a strong bias to select an individual from the same race as mate.

Whenever an individual is selected as parent it generates one offspring. Reproduction may be mitotic or meiotic. Using mitosis a genetically identical offspring is generated and a mutation operator is applied to its DNA. If meiosis is employed the individual uses its own Selector to chose one compatible mate from the current population. Subsequently a new individual is generated using recombination and mutation operators. The most common recombination operator is crossover. Individuals are initialized at the beginning of the algorithm (start population) or during their creation as offspring. During initialization a schedule is generated from the DNA. The resulting schedule is evaluated and a fitness value is assigned to the individual. The fitness value represents the quality of the generated schedule according to a given objective function.

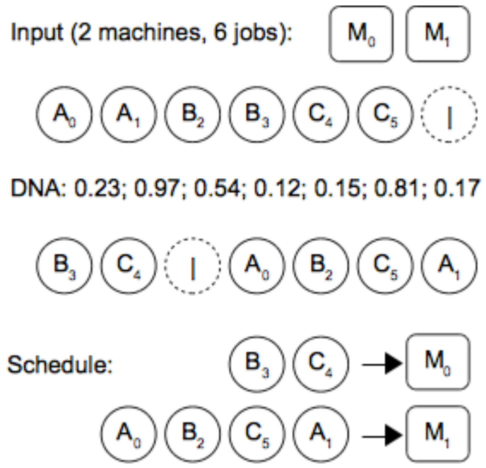
To illustrate the concept of multiple species we will introduce two example species. The first species (S1) uses a text string to encode the genetic information. The DNA is a pattern containing recipes denoted as capital Letters. To generate a schedule all Jobs are ordered according to their recipe to match this pattern (see Figure 4). The resulting job sequence is cut into partitions of fixed size, which are approximately equisized. Recent studies have shown that this arbitrary constraint oftentimes leads to very good results (Uhlig 09). This may be attributed to the reduced size of the search space and the common ground that all individuals can rely on. Especially regarding recombination different sized partitions may lead to significant disruption. Obviously it is easy to find examples were equisized partitions are disadvantageous. For some problems there does not even exist any valid solution with equisized partitions. This individual uses swap mutation and recombination. Recombination is implemented as single point crossover, multi point crossover, or uniform crossover. Obviously DNA generated with simple crossover may be incorrect since the number of certain recipes in the pattern may be wrong. To solve this problem we apply a correction algorithm. It replaces the first occurrence of a too frequent recipe with a recipe occurring too infrequently until the string pattern is valid. This correction can be seen as a special kind of mutation.



Input (2 machines, 6 jobs):
M_0
M_1
A_0 A_1 B_2 B_3 C_4 C_5
DNA:
ABCBCA
Schedule:
A_0 B_2 C_4 → M_0
B_3 C_5 A_1 → M_1

Figure 4: Schedule generation - Individual with recipe pattern as DNA and equisized partitions.

The second species (S2) relies on flexible partition sizes. To achieve this we introduce special marker jobs as cutting points. For sequencing purposes these marker jobs are treated just like all other jobs. For this species we employ random number indices to encode the relative job position. The jobs are sorted according to their corresponding index number (see Figure 5). We employ a stable sorting algorithm. This species employs crossover for recombination and supports two different mutation operators. Swap mutation swaps to values in the DNA, while random resetting replaces one index with a new random value.



Input (2 machines, 6 jobs):
M_0
M_1
A_0 A_1 B_2 B_3 C_4 C_5 |
DNA: 0.23; 0.97; 0.54; 0.12; 0.15; 0.81; 0.17
B_3 C_4 | A_0 B_2 C_5 A_1
Schedule:
B_3 C_4 → M_0
A_0 B_2 C_5 A_1 → M_1

Figure 5: Schedule generation - Individual with random number index DNA and additional marker jobs for partitions

Other species have been implemented as well, but will not be discussed in detail in this paper. Some of them use different approaches for partitioning employing a list containing machine numbers to explicitly assign jobs to certain machines. Furthermore there are combinations of methods introduced for S1 and S2. For example an individual using recipe patterns in conjunction with flexible partitions. The experiments performed for this paper only used S1 and S2.

Design of Experiments

For our experiments we employed 5 different test setups with parallel cluster tools (see Table 1). The tests include experiments with incompatible cluster tools – tools that cannot process jobs with certain recipes. Dedication of tools to certain jobs depending on given machine qualifications is typical constraint in semiconductor manufacturing. Furthermore we have instances were a tool is significantly slower than other ones. As a reference algorithm we tested all problem instances with a Monte Carlo Method. This method returns the best solution off 10,000 random generated schedules. Each experiment has been repeated 100 times to minimize variance.

For this paper, we evaluated a multi species algorithm using two species described previously (S1 and S2). For reference purposes we also run simulations using the GA with only one species for each them.

Table 1: Test Setups

No.	**Jobs**	**Recipes**	**Machines**
1	40	10	4 incompatible Cluster Tools with 4 load ports
2	30	5	2 incompatible Cluster Tools with 4/2 load ports
3	30	10	3 identical Cluster tools with 4 load ports
4	40	10	2 cluster tools with 4 load ports (1 slow)
5	40	10	4 cluster tools with 4 load ports (1 slow, 2 incompatible)

For the genetic algorithm we used the following parameters:

- Fitness: C_{max} as objective function
- 100 generations and 100 individuals per generation (equally distributed among species for multi species instances) resulting in 10,000 simulation calls per run
- Parent selection: stochastic ranking selection
- Survivor selection: deterministic with uncapped elitism (no special treatment for different species)
- Single point crossover for recombination

All experiments were performed on a reasonably modern laptop. Calculation time for one algorithm run, with 10,000 simulation calls, was always well below 5 seconds for one repetition.

Results

The results of our experiments illustrate that all GAs perform significantly better than a Monte Carlo algorithm (see Table 2). On average the multi species algorithm returned the best results. However, with regard to single tests it never was the winner. This can be attributed to the fact, that only two different species were employed and no special selection enforced the surviving of both species. Consequently there were at most two subpopulations exploring different niches. It can be assumed, that in most cases one species became extinct very soon, which effectively resulted in a single species GA. Nevertheless the multi species proved to be very robust, since it found a solution for nearly all test runs. In contrast the GA using only S1 found no valid solution in 1 out of 10 times while S2 found no solution in 1 out of 20 times. Regarding the results it is obvious that S1 and S2 excel in different test cases. Generally S1 is more effective whenever sequencing is difficult and equisized partitions are not a hindrance. In contrast S2 excels at partitioning. The multi species GA can benefit from both species, relying on the one, which is better suited to solve a certain task. Consequently it performed nearly as well as the better single species GA for all problem instances.

Table 2: Average improvement in tests 1-5 in comparison to Monte Carlo Method and percentage of successfully solved problems (100 repetitions per test)

	Individuals		
	S1	**S2**	**S1 + S2**
Test 1	24,1%	21,9%	22,5%
Test 2	9,0%	16,5%	15,9%
Test 3	12,0%	11,2%	12,0%
Test 4	19,0%	15,9%	18,2%
Test 5	21,1%	19,1%	18,4%
Avg. improvement	17,0%	16,9%	17,4%
Solved instances	90%	94,6%	99,6%

Discussion

Genetic algorithms are general-purpose heuristics, which are often outperformed by specialized algorithms (see Figure 1). Regarding cluster tool scheduling a simplex algorithm (Boehl 08) is able to determine an optimal job sequence for many problems considering only a single tool with C_{max} as objective. However, this algorithm cannot be applied to problems with parallel tools. A promising idea would be an individual, which employs the simplex algorithm for sequence generation for cluster tools and uses a genetic code for partitioning. Effectively this would result in a hybrid GA. Often times a GAs with local optimization relying on domain knowledge is called memetic algorithm (Eiben 2007, p. 173-185).

The algorithm introduced in this paper was developed for offline scheduling problems. However, in semiconductor manufacturing scheduling is often neglected in favor of simple dispatching rules, since the actual problem description may change very often. This for example can be caused by arrival of new jobs in the queue of a tool group. Furthermore there are unpredictable events like machine breakdowns. Especially machine breakdowns occur very often in semiconductor manufacturing. These facts seem to imply that scheduling cannot be applied successfully to these problems. However, with regard to our fast Simulator it is easy to recalculate a completely new schedule in a few seconds, whenever a change occurs. Consequently the responsiveness of our proposed scheduling system should be adequate with regard to real world requirements.

At the moment our multi species GA relies heavily on the idea that different implementations of individuals may have an inherent advantage with regard to different scheduling problems. However, future development might lead to an implementation, which outperforms other individuals for every problem instance. Consequently no further benefit would be expected if other inferior species were employed in conjunction with this super-individual. In this case one might resort to a multi species algorithm employing a labeling approach instead off actual different implementations for speciation.

Conclusions

The multi species GA proposed in this paper achieved reliable good results, although we employed only two species. The limited number of species prevented the algorithm from exploring multiple niches in the search space effectively. Hence it did not improve the results with regard to specific experiments in comparison to conventional GAs. On average it returned the best results since it could always rely on the genetic representation which generated better results for a given problem instance. This behavior led to a very high robustness. Given a certain problem it is the best solution, if we do not know in advance which species is suited best to solve this problem instance.

In the future experiments using more species will be performed. Furthermore special selection mechanisms will be implemented to avoid early extinction of species. Hopefully this will lead to further improvements, by emphasizing the idea of exploring multiple niches.

References

Pinedo, M. 2002. *Scheduling: Theory, Algorithm and Systems*. Prentice Hall.

Eiben, A. E., and Smith, J. E. 2007. *Introduction to Evolutionary Computing*. Springer.

Nouioua, F., and Hamdi-Cherif, A. 2009. Using age, competition and dynamic parameters to control the population's diversity of an evolutionary algorithm. *In INISTA 2009*. 432-436.

Unbehaun, R., and Rose, O. 2007. Predicting Cluster Tool Behavior with Slow Down Factors. In *Proceedings of the 2007 Winter Simulation Conference*. 1755-1760.

Varadarajan, A., and Sarin, S. C. 2006. A survey of dispatching rules for operational control in wafer fabrication. In *Information Control Problems in Manufacturing Vol. 1, Part 1*.

Pfund, M., Mason, S., Fowler, J. W. 2006. Dispatching and Scheduling in Semiconductor Manufacturing: State of the Art and Survey of Needs, *Handbook of Production* Scheduling, J. Herrmann, (eds.), 213-241,

Husbands, P. 1994. Distributed Coevolutionary Genetic Algorithms for Multi-Criteria and Multi-Constraint Optimisation *Evolutionary Computation, AISB Workshop,* Springer, 150-165

Boehl, E. 2008. *Einsatz des Simplexverfahrens in einer Clustertoolanalyse,* unpublished.

Uhlig, T. 2009. *Employing Evolutionary Algorithms to Minimize the Makespan of Schedules with Sequence-Dependent Process Times*. Thesis, Department of Applied Computer Science, Dresden University of Technology.

Graham, R. L., Lawler, E. L., Lenstra, J. K., Rinnooy Kann, A. H. G., 1979. Optimization and Approximation in Deterministic Sequencing and Scheduling: a Survey, *Annals of Discrete Mathematics 5*, 287–326.

Eldrege, N., and Gould, S. J., 1972. *Models of Paleobiology,* 82-115. Freeman Cooper.