Proceedings of the 2013 Winter Simulation Conference
R. Pasupathy, S.-H. Kim, A. Tolk, R. Hill, and M. E. Kuhl, eds.

REDUCING COMPUTATION TIME IN SIMULATION-BASED OPTIMIZATION OF MANUFACTURING SYSTEMS

Matthias Frank

Chair of Modeling and Simulation
Technical University of Dresden
Nöthnitzer Straße 46
Dresden, 01187, GERMANY

Christoph Laroque

Business Computing, esp. CIM
University of Paderborn
Fürstenallee 11
Paderborn, 33102, GERMANY

Tobias Uhlig

Department of Computer Science
Universität der Bundeswehr München
Neubiberg, 85577, GERMANY

ABSTRACT

The analysis of production systems using discrete, event-based simulation is wide spread and generally accepted as a decision support technology. It aims either at the comparison of competitive system designs or the identification of a best possible parameter configuration of a simulation model. Here, combinatorial techniques of simulation and optimization methods support the user in finding optimal solutions, but typically result in long computation times, which often prohibits a practical application in industry. To close this gap, this paper presents a fast converging procedure combining a Genetic Algorithm with a material flow simulation including an interactive analysis of simulation runs. An early termination of simulation runs is used for unpromising parameter configurations. The integrated implementation allows automated, distributed simulation runs for practical, complex production systems. A use-case shows the proof of concept with a reference model and demonstrates the resulting speed-up of this approach.

1 MOTIVATION

Modern business computing, especially in the area of operations research, offers a wide variety of methods for complex problem solving for planning, scheduling and control of production and logistic processes. Those processes, which are to be either designed or improved, are typically projected to models and then optimized by the use of simulation and/or optimization technologies in order to improve decision variables and resulting key performance indicators under a given set of restrictions. In simulation, this improvement is usually achieved by the iterative evaluation of multiple scenarios and their subsequent simulation results (Law and Kelton 2000). In the case of optimization, the optimal configuration is achieved by mathematical optimization algorithms or (meta-) heuristic approaches (Rardin 1998). Due to the high computational demand of both iterative evaluation and mathematical optimization, specific procedures as a combination of both simulation and optimization were derived. They combine both advantages: an optimization algorithm can be used to automatically generate a specific model configuration, which can be evaluated by simulation runs (Fu 2002). Especially for simulation models with stochastic influence factors, which need a high amount of simulation runs, these procedures can lead to faster identification

of improving model configurations than standard methods for the design of simulation experiments (Fu 2002). It remains a challenge to further improve the performance of finding a good solution with a high global quality, especially in the area of production. The given complexity of the underlying systems, and thereby the simulation model, is very high, so that the application of standard combinatorial approaches of mathematical optimization and material flow simulation is infeasible for industrial applications due to high computation costs. Our approach applies a simulation-based optimization approach of a Genetic Algorithm and a material flow simulation, that are employed in a distributed matter. We speed up computation through distributed processing and achieve fast convergence by stepping-out from worse parameter configurations as early as possible during the execution of the simulation run. Our implementation is integrated with the material flow simulator d^{3}fact and manages initiation of nodes and data exchange between them. By using generic interfaces to d^{3}fact, we are able to apply our method in a practical, industrial environment. The paper presents in short the necessary state of the art in simulation based optimization and design of experiments in the following section. The conceptual approach of the procedure is presented in section 3, followed by the implementation. The first evaluation results of the procedure are shown in section 4. The paper closes with an outlook on future work in this area.

2 STATE OF THE ART

2.1 Design of Experiments

Design of experiments (DOE) refers to the use of statistical techniques to create an efficient, systematic set of controlled experiments for collecting data in order to estimate relationships between independent and dependent variables through measurement. In the area of simulation, DOE is used for the systematic evaluation of simulation models in order to identify a set of model parameters, which leads to the desired simulation results. Each simulation run hereby evaluates a concrete set of parameters. Typically, the simulation models include stochastic influence factors, so that a single simulation run is not sufficient for the evaluation of the parameter set, and multiple simulation runs for each of the configuration sets are to be performed. Efficient procedures like 2k-factorial-Design (Banks 2001), Plackett-Burman-experiments as well as response-surface method (RSM) or evolutionary optimization (EVOP) (Fu 2002) are used to reduce the number of required simulation runs by determining parameter sets that will likely lead to a good result.

2.2 Simulation-based Optimization

Simulation-based optimization is an iterative process to determine a solution to a given problem. Each iteration consist of three basic steps: the generation of a candidate-solution, simulation of the proposed solution, and finally the evaluation of the simulation (see Figure 1). Feedback gained from simulation and evaluation is used to steer the process of generating new candidate-solutions. With this approach the proposed solutions improve iteratively until the optimization terminates. Typical termination criteria depend on the attained solution quality, the recent optimization progress, or the passing of a given amount of time or iterations.

Simulation-based optimization can be seen as a movement through the search space of simulation parameters, where the simulation maps a point in that space to a point in the space of performance indicators. The optimization therefore needs to ensure that the found solution is not just a local optimum, and it needs to find a good solution (optimality is generally not guaranteed) in a small number of simulation runs, i.e. converge quickly (Kabirian and Olafsson 2007). Many different types of optimization strategies exist (see Hachicha et al. (2010) for a classification approach) and have also partially been implemented in commercially available tools (Fu 2002, Law and Kelton 2000). We focus on meta-heuristics, specifically Genetic Algorithms (Russell and Norvig 2010). They work generically without knowledge of the concrete problem, making them universally applicable using standard implementations. GAs have been successfully applied to simulation based optimization (Paul and Chanev 1998; Krug 2002; Krug et al. 2002), and Laroque et al. (2012) shows the applicability of meta-heuristics to design or configure manufacturing

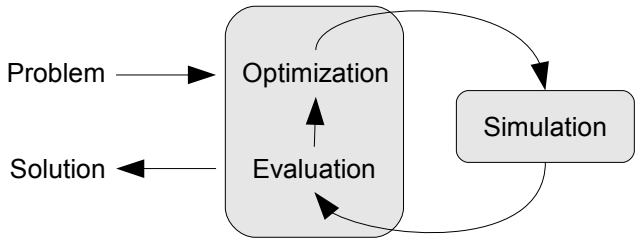

Figure 1: The basic optimization cycle using simulation to evaluate solutions generated by the optimizer.

plants. General implementations of these meta-heuristics are available in various open software libraries (e. g. ECJ (Luke 2011)), providing an efficient implementation of the optimization component.

However the typical bottle-neck of simulation-based optimization is simulation speed. Evaluation of complex model can be quite time consuming, while the generation of new candidate solutions usually requires little calculation time. To operate effectively we need to complete as many iterations as possible for optimization. Fast simulation is therefore of great benefit. In section 3 we will discuss an approach to reduce the time spend for simulation.

2.3 Optimization using a Genetic Algorithm

To optimize the model from section 4.1 we use a genetic algorithm. As an evolutionary algorithm it is a population based meta heuristic, using variation and selection to iteratively generate better solutions. Variation is used to derive new candidate-solutions from already existing ones, relying typically on operations like recombination and mutation. Selection steers the evolutionary process by propagating better solutions and eliminating the bad ones. The selection process has a bias, favoring fitter individuals. The fitness measures the adequateness of a candidate solution, in our case the quality of a parameter set determined by simulation. Each population consists of many individuals representing possible solutions to the problem and with each passing generation on can expect an increase in average fitness. Eventually this should lead to a satisfying solution.

We use a modified (μ, λ)-evolution strategy with truncation and tournament selection, 2-point-crossover and Gaussian convolution. Figure 2 shows the scheme of the used GA. Every model configuration is one individual in the GA. Meaning the genome of an individual corresponds to the set of input parameters for the model.

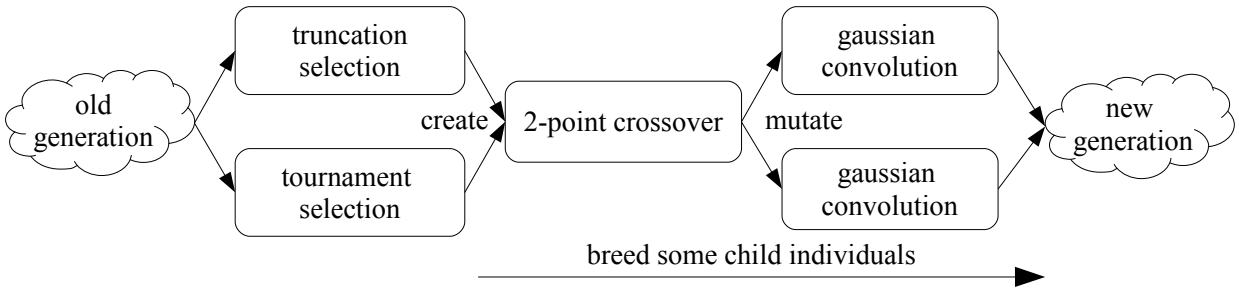

Figure 2: Scheme of the genetic algorithm.

We use $\mu = 120$ individuals in a population, evaluated by simulation. For breeding the next generation, truncation and tournament selection each select ten individuals ($\lambda = 2 \cdot 10$). The tournament selection uses a tournament size of 15. Each pair of parent individuals produces 6 children using a 2-point-crossover.

The new individuals are mutated using Gaussian convolution. The mutation probability is 0.3333 for each gene. The (μ, λ)-evolution strategy produces for every generation a completely new population. We do not preserve good individuals to avoid getting trapped in a local extrema.

The parameters and operators of the Genetic Algorithm were obtained by optimizing the Rastrigin function (Törn and Zilinskas 1989; Mühlenbein et al. 1991 see (2)) with the same number of parameters as the model. We assume this will result in a good configuration to optimize the model. While this strategy does not guarantee to find the optimal Genetic Algorithm and parameters, we attained at least an useful configuration. There probably is a better set of parameters and evolutionary operators, i.e. selection, cross over or mutation. With regard to the high computational complexity, it is however infeasible to experimentally analyze a huge number of potential parameter stes. A more detailed introduction to Genetic Algorithms is given in Luke (2009).

2.4 Discrete Event Simulation with d^{3}fact

d^{3}fact is a discrete, event based material flow simulation framework, designed and implemented at the Heinz Nixdorf Institute of the University of Paderborn, Germany. Designed as a multi-user environment, it allows simultaneous, collaborative modeling and simulation of a model by multiple simulation experts. d^{3}fact consists of a modeling tool, a simulation server, that runs the simulation and few visualization options from 2D to 3D. The open source software is based on the Eclipse Rich Client Platform (RCP) and is implemented in Java (Dangelmaier et al. 2005, Laroque (2007)). The project provides a Java API to program material flow simulation models independently from the modeling interface.

3 CONCEPT: REDUCING COMPUTATION TIME BY EARLY EXITS AND SIMULATION PROFILES

Typically a parameter set is evaluated using multiple replications and long runs to get accurate simulation results. The genetic algorithm, especially the truncation selection compares the fitness of different individuals, selecting the good ones and discarding the bad individuals. As long as we guarantee comparability of individuals the genetic algorithm operates effectively. Therefore an accurate fitness value is not required as long as the individuals are still comparable. The presented approach, model and experiments are based on Frank (2013).

Presumption 1 The further away the genetic algorithm is from finding an acceptable solution, the greater the tolerable inaccuracy when evaluating individuals.

Typically, the diversity of the population at the beginning of an optimization process is significantly higher than at the end. Using this assumption we can initially tolerate bigger inaccuracies, since the huge differences in fitness values outweigh the inaccuracies, when we compare individuals. At the end of an optimization the population contains a lot of possible solutions with similar fitness scores. Accordingly the evaluation has to be more accurate to reliably identify the fitter individual.

In this work we implement this proposition 1 by using simulations profiles, defining a run length and number of replications for every phase of the algorithm (see Figure 3). These profiles (see Table 1) are experimentally adjusted to the model's stochastic system behavior.

During the exploration phase the diversity should be high enough to find improved candidate-solutions very often. Since large improvements are expected a single simulation replication should suffice to identify promising new candidates. Therefore we use the *fast* profile for the initial phase. The simulations profile for the exploitation phase is similar to profile *fast*. Because of the typically smaller diversity, compared to exploration phase, a few more replications and longer runs are necessary. In the end of the optimization the Genetic Algorithm finds acceptable solutions. The individuals should be evaluated accurately to choose reliably identify the best solution. Therefore multiple simulation replications with adequate runs lengths are essential.

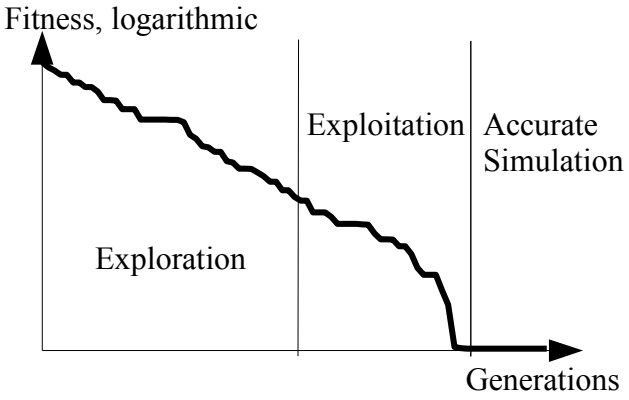

Figure 3: Different phases on a logarithmic fitness scale.

Table 1: Simulation profiles.

phase	simulation profile	accuracy	replications	length of replication
exploration	fast	raw evaluation of an individual	1	250 min
exploitation	medium	small impreciseness tolerable	3	625 min
accurate simulation	accurate	multi replications and longer runs for required accuracy	10	1000 min

In this work, we switch between the simulations profiles depending on the slope of the fitness graph. Let $f(g)$ the best fitness of generation g and h a step size. The resulting slope is:

$$\alpha = \arctan\left(\frac{f(g-h)-f(g)}{h}\right).$$

The angle α is defined in the half open interval $[0^\circ, 90^\circ)$ where $\alpha = 0$ stands for no optimization progress within the last h iterations. An angle with near 90° represent a fast convergence (see Figure 4).

Although it is possible to use the actual value of the slope, we prefere the use of angles, since they are more intuitive and illustrative.

We use the threshold of 87° to switch between simulation profile *fast* and *medium*, and 25° to switch between profile *medium* and *accurate*. Defining the step size h should be done carefully. If for some reason the genetic algorithm does not generate an improved solution for some generations the angle α gets smaller eventually. A slower simulation profile might be chosen prematurely, causing an unnecessary high accuracy and accordingly a waste of computation time. On the other hand, if the step size h is too large the switching of the simulation profiles will occur too late. Here we use an empirically determined step size of $h = 7$.

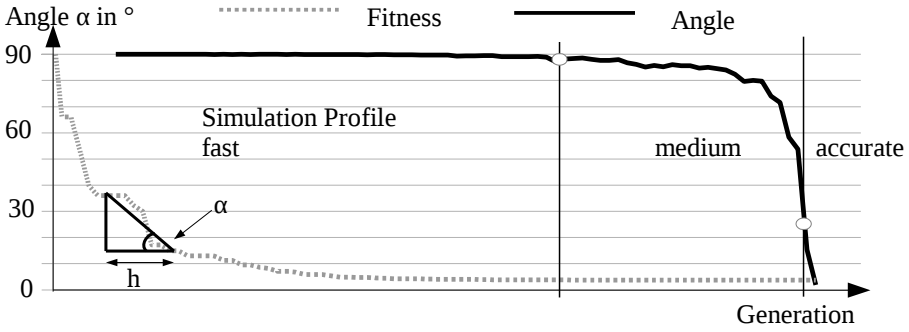

Figure 4: Angle α over the whole optimization progress.

4 EXPERIMENTS

Here we describe the optimization of the employed production model in section 4.1. Due to the limited computation time the experiments for this model were performed only once. Therefore we implemented additionally theoretical experiments to validate the results.

4.1 Model

We use a model of a manufacturing plant to demonstrate the optimization approach. It consists of five independent types of sources and nine different kinds of processors with their respective buffers. Certain kinds of processors are dedicated to the respective production steps. Using a three staged production process five input tokens are stepwise assembled to a single output token. The stages are connected by conveyors, that have a given speed. Each source releases one specific kind of token. Interarrival times of tokens are drawn from a Gaussian distribution with mean and variance set for each type of source. Processing times for the processors are also normally distributed. Processors have their own buffers with a given capacity to store waiting tokens. The input parameters of the model, adapted during optimization, are

- Number of available sources for each type.
- Number of available processors for each type.
- Capacity of buffers.
- Speed of conveyors.

The tokens are transported via a point-to-point conveyor network to the next process stage. Depending on the number of sources or processors, the conveyors intersect each other. The intersection point routes the incoming token in a probabilistic way to one of the target buffers. The position of these intersection points co-determine the length of the conveyors. Under the assumption, that the length of the conveyors don't change the (stationary) system behavior, the intersection points can be chosen with a hill climbing algorithm.

The model is evaluated using the discrete event simulator d^3fact and the resulting performance indicators are mapped to a cost function. For a fully automated run, we need an automatic criteria to remove the initial bias of each simulation run. In Hoad et al. (2009) an overview about warm-up length estimation criterion is given. We use the MSER-5 criteria (Spratt 1998; White Jr et al. 2000), which is commonly accepted as a best practice (White Jr et al. 2000; Hoad et al. 2008; Hoad et al. 2009).

4.1.1 Optimization Objective

The challenge is to find a configuration leading to minimal costs for a desired production rate. To evaluate a certain configuration we map the costs c and the production rate p to a fitness value using a fitness function $f(p,c)$. Let p_d be the desired production rate.

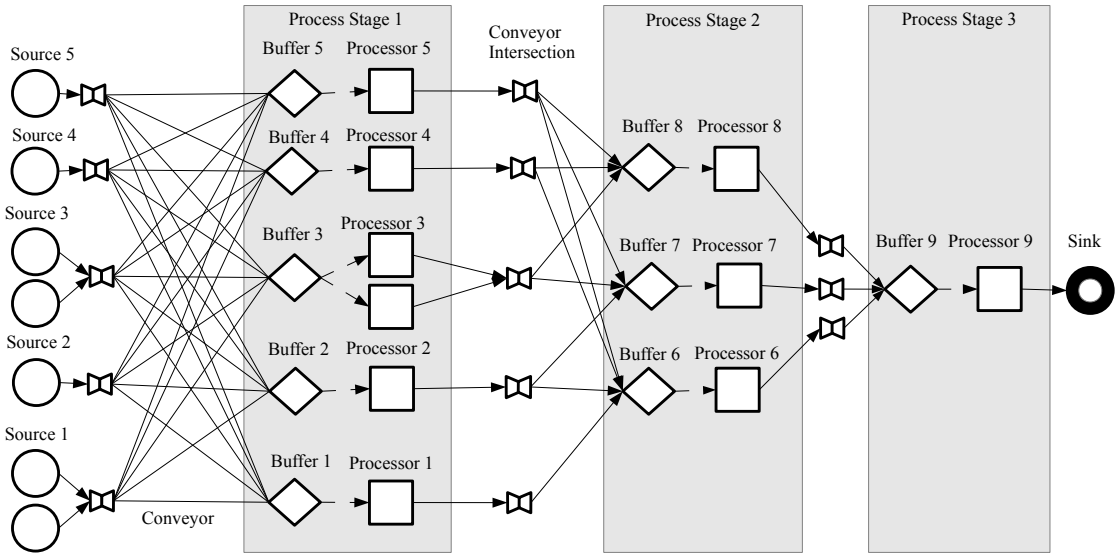

Figure 5: Scheme of the used model.

The fitness function should have a minimum for the desired production rate and minimal costs, so the parabola (1) seems to be well-suited therefore.

$$f(p,c) = (p - p_d)^2 + c. \tag{1}$$

The first term of (1) has its minimum with $p = p_d$. Minor differences of p and p_d are less punished then major differences through the quadratic term. Adding up the costs of a model instance should lead to minimal costs by minimizing of (1). This fitness function $f(p,c)$ could also expressed as a multi criteria optimization using two separate functions $f_1(p)$ and $f_2(c)$. By adding up the production term and the cost term the problem can be easily handled as a single criterion problem.

4.2 Optimization of the Model

To implement the genetic algorithm, we use the Java framework ECJ (Luke 2011). To evaluate the individuals, the grid middleware JPPF (Cohen 2013) generates a task for each individual and ships it via one or more drivers to the according nodes. The node launches d^{3}fact with the model and starts the simulation. During the simulation the MSER-5 criteria looks for the end of the initial bias. According to the simulation profile the node simulates the required replications with the required run length. After completing the task, the node calculates the fitness of the individual and returns the result.

JPPF is a self managed peer-to-peer grid middleware for pc clusters, desktop grids or cloud environments. The build-in fault tolerance mechanisms allow also a geographical distribution over large distances. In this work, we use JPPF nodes, placed in Amazon EC2 instances in North Virginia, USA together with some small business servers and workstations in Dresden, Germany.

As a starting point, we search for a model instance capable of producing $p_d = 10,000$ tokens a day. So the fitness function (1) has a minimum at $f(p_d,c)$ with minimal costs.

We start the optimization with 120 randomly created individuals. During the exploration phase the GA found better individuals in nearly every generation. As it was expected, the diversity is big enough to evaluate with profile *fast*. At generation 30 the slope of the fitness flattens and our heuristics switches to simulation profile *medium*. In the exploitation phase, the GA finds some slightly better individuals but not as fast as in the exploration phase, accordingly the simulation profile is switched to *accurate* at Generation 45. To save computation time, we cancel the GA after one accurate evaluated generation. The

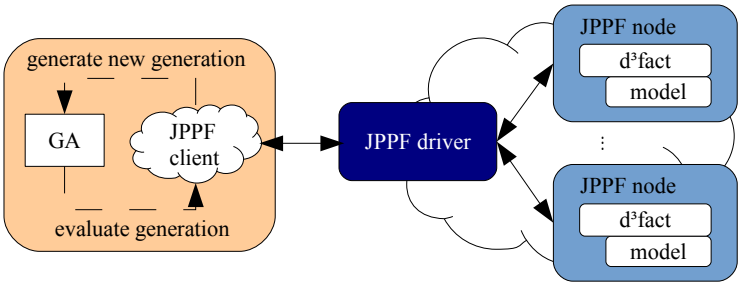

Figure 6: Optimization and simulation in a distributed environment.

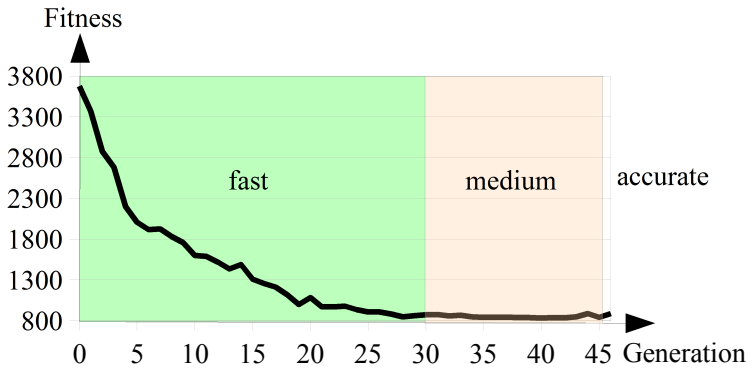

Figure 7: Progress of the fitness with simulation profiles.

best solution we found has a production rate of 10189 tokens a day. This small difference of desired and found production rate is quite acceptable. However the GA can not guarantee whether the found solution is the best solution. Neither can we prove optimality or portability with regard to the chosen operators and parameters. Nevertheless experience shows that GAs are working fine for many problems.

The experiment consumed 3296 CPU hours, most of the time was required for simulation. On average one simulation with profile *fast* needed 14.7 min, with profile *medium* 50.3 min and with profile *accurate* 6.7 h. So a repetition without simulation profiles would need approximately 38000 CPU hours, which is to much for our study. Therefore we designed some experiments to validate the simulation profile approach.

4.3 Additional Validating Experiments

In section 4.2 we showed, that our approach with reduced computation time worked. But we did not explain why it works. In this section we perform some theoretical experiments to provide a deeper understanding.

For these experiments we use the same GA and parameters as in section 2.3. To evaluate the individuals we replace simulation with the generalized multidimensional Rastrigin function f_{Ra} to save computation time and to get reasonable fitness values. The Rastrigin function has a lot of local extrema and one global minimum with $f_{Ra}(\vec{0}) = 0$.

To model the inaccuracy in simulation, we add an error to the Rastrigin function (Frank 2013).

$$f_{Ra}(\vec{x}) = An + \sum_{i=1}^{n} \left[x_i^2 - A\cos(2\pi x_i)\right] \text{ with } A = 10 \text{ and } n \text{ size of } \vec{x}. \tag{2}$$

$$f_{Ra}^{e}(\vec{x}) = f_{Ra}(\vec{x}) + f_{Ra}(\vec{x}) \cdot e \cdot \mathbb{N}(0,1). \tag{3}$$

This error is a product of a normal distributed random number with mean of 0 and standard deviation of 1 and a proportional part of f_{Ra} The parameter e adjusts the magnitude of the error.

To show the influence of an error containing fitness function, we compare first the selection of individuals using fitness functions f_{Ra} and f^e_{Ra}. Secondly, we compare the behavior of these fitness functions within the GA.

4.3.1 Selection Operators

In the GA of section 2.3 we use a truncation selection and a tournament selection. To show how the selection operators works with f^e_{Ra}, we randomly generate a 120 individual population and evaluate them with f_{Ra} and f^e_{Ra}. Now we can determine if a selection operator selects the same individuals for the fitness function with an error in comparison to the unbiased fitness function. If a selection operator selects an individual, evaluated with f^e_{Ra}, which also is correctly selected using f_{Ra}, we call this a *hit*.

The individuals are generated in a range of $[-10.0, 10.0]^d$ in a d-dimensional search space. For every combination of (d,e) with $e \in [0.005, 0.2]$ (step size 0.005) and $d \in [2, 30]$ we perform 2000 tests and count for each test the number of its hits.

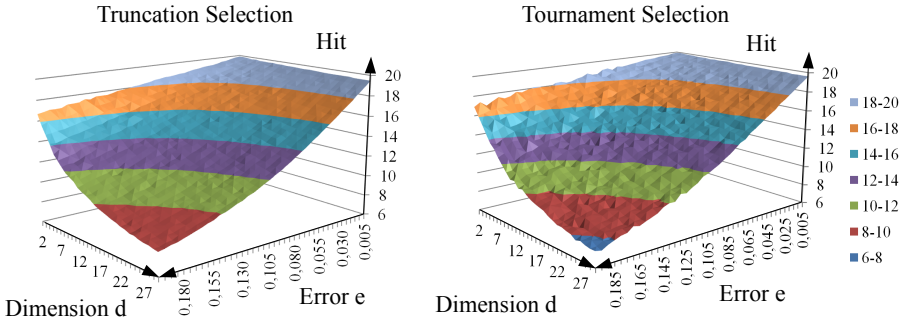

Figure 8: Number of hits for truncation and tournament selection (Frank 2013).

The selection operator selects 20 of 120 individuals. So the aim is to select all individuals correctly in spite of an error containing fitness function. The left chart shows the hits of the truncation selection depend on d and e, the right chart shows the results of the tournament selection with tournament size of 15. Both charts in Figure 8 show quite similar behavior. With increasing d and e more and more individuals are wrongly selected. But if the error is small, independent of d nearly all individuals are correctly selected. So a small error is tolerable for these selection operators (Frank 2013).

4.3.2 Tests with the Genetic Algorithm

In section 4.3.1 we have seen, that a small error is tolerable for truncation and tournament selection. Here, we examine the whole genetic algorithm from section 2.3 with an error containing fitness function. To model the simulation profiles, we change e for each profile. For the profile *accurate*, the error must be $e = 0$ to find unbiased solutions in the end of the optimization. Table 2 shows some values for e to play around with some values.

Experiment 1 is used as reference, no error is used. In Experiments 2 to 5 we have tried some error values. For each experiments we perform 10 runs. In Table 2 are the results with smallest, biggest and average fitness. The global minimum of the Rastrigin function is zero. So no experiments hit is exactly, but approximately. Within the scope of the available precision, experiment 1 to 4 reaches the same results. Only experiment 5, which uses the same error for profile *fast* and *medium* does not reach that values. These experiments show that also a genetic algorithm can tolerate an error containing fitness function.

Figure 9 shows the best fitness of each generation of the unbiased experiment 1 and for experiment 3. We observe the effect that the error containing fitness of experiment 3 seems to converge faster than unbiased fitness. But it converges not really faster. The error is modeled as a random value (see (3)), so it can be smaller or greater than zero. If a good, maybe not the best, individual gets a negative error, so the

Table 2: Experiments with an error containing fitness function within a genetic algorithm (Frank 2013).

	Error e Simulation Profile			Results		
Experiment	**fast**	**medium**	**accurate**	**smallest fitness**	**mean**	**biggest fitness**
1	0	0	0	1.06	2.69	6.10
2	0.08	0.03	0	1.06	2.00	4.11
3	0.2	0.1	0	1.06	3.14	5.08
4	0.5	0.3	0	1.08	2.52	5.09
5	0.2	0.2	0	1.81	5.08	11.05

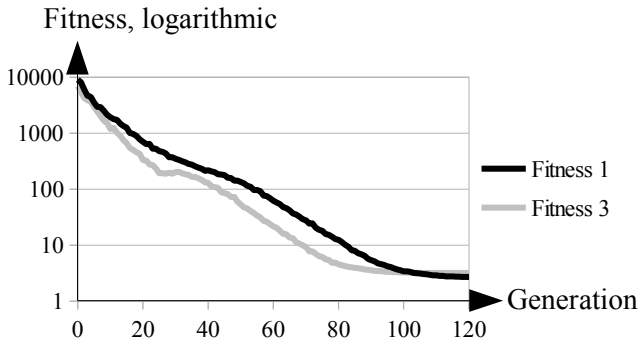

Figure 9: Comparison of fitness with and with an error.

fitness gets smaller and it seems better than an individual with nearly the same fitness but, with positive error. In the end of the optimization, when the error gets small, this effect disappear.

5 CONCLUSION & OUTLOOK

This work presents a concept to reduce the required computation time of simulation-based optimization with a Genetic Algorithm. This is accomplished by reducing the run length and number of replications of a simulation depending on the slope of the fitness. The accuracy of the simulation is adapted during the optimization progress. To demonstrate the concept, a model of a manufacturing plant is configured using a genetic algorithm. To validate the approach, we employed experiments with an error containing fitness function.

The current approach relies on simulation profiles adjusted to a specific model. Future works could determine the required accuracy depending on the diversity. Instead of defining a certain number of runs in advance, the simulation profiles adapt automatically to the desired level of accuracy. An automatic replication estimator could determine the number of replications for the required accuracy (Hoad et al. 2007).

We consider only one Genetic Algorithm and one configuration. The main focus is to reduce computation time with short simulations runs and fewer replications whenever the fitness of individuals is easily distinguished. Presumably this approach is not limited to the presented Genetic Algorithm. Other selection operators must be evaluated, especially fitness proportional selection operators might behave differently in comparison to the employed truncation or tournament selection. Furthermore other error models should be considered. Finally this approach should be examined using other optimization algorithms, e.g., covariance matrix adaptation evolution strategy, particle swarm optimization or variance reduction.

REFERENCES

Banks, J. 2001. *Discrete Event System Simulation*. 3rd ed. Upper Saddle River, NJ: Prentice Hall.

Cohen, L. 2013. "Java Parallel Processing Framework". http://www.jppf.org/, Accessed at March 25, 2013.

Dangelmaier, W., D. Huber, C. Laroque, and B. Mueck. 2005. "d^{3}FACT insight - An Immersive Material Flow Simulator with Multi-User Support". In *Proceedings of the 2005 Summer Computer Simulation Conference*, edited by A. Bruzzone and E. Williams, 239–242. SCS Press.

Frank, M. 2013, January. "Konzeption und Implementierung eines schnell konvergierenden Verfahrens zur verteilten, simulationsgestuetzten Optimierung von Fertigungslinien". Diploma Thesis, Faculty of Computer Science, Technical University of Dresden, Germany.

Fu, M. C. 2002. "Optimization for Simulation: Theory vs. Practice". *INFORMS Journal on Computing* 14 (3): 192–215.

Hachicha, W., A. Ammeri, F. Masmoudi, and H. Chachoub. 2010. "A Comprehensive Literature Classification of Simulation Optimisation Methods". In *The 9th International Conference on Multi Objective Programming and Goal Programming (MOPGP)–May*. Sousse, Tunisia.

Hoad, K., S. Robinson, and R. Davies. 2007. "Automating DES Output Analysis: How Many Replications to Run". 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, 505–512. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Hoad, K., S. Robinson, and R. Davies. 2008. "PART 1: Review of Methods and Shortlisting". Warwick Business School, Coventry, UK, http://www2.warwick.ac.uk/fac/soc/wbs/projects/autosimoa/warmup/warm-up_writeup_part1_review_of_methods_and_shortlisting.doc – Accessed at March 24, 2013.

Hoad, K., S. Robinson, and R. Davies. 2009. "Automating Warm-up Length Estimation". *Journal of the Operational Research Society* 61 (9): 1389–1403.

Kabirian, A., and S. Olafsson. 2007. "Allocation of Simulation Runs for Simulation Optimization". 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, 363–371. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Krug, W. 2002. *Coupling ISSOP with other Simulation Systems. Modelling, Simulation and Optimization for Manufacturing, Organisational and Logistical Processes*. SCS European Publishing House.

Krug, W., T. Wiedemann, J. Liebelt, and B. Baumbach. 2002. "Simulation and Optimization in Manufacturing, Organisation and Logistics". In *Simulation in Industry - Modeling, Simulation and Optimization - Proceedings of the 14th European Simulation Symposium*, edited by A. Verbraeck and W. Krig. SCS Press.

Laroque, C. 2007. *Ein mehrbenutzerfähiges Werkzeug zur Modellierung und richtungsoffenen Simulation von wahlweise funktions-und objektorientiert gegliederten Fertigungssystemen*. Ph. D. Thesis, Universität Paderborn, Germany.

Laroque, C., A. Klaas, J.-H. Fischer, and M. Kuntze. 2012. "Fast Converging, Automated Experimen Runs for Material Flow Simulations using Distributed Computing and Combined Metaheuristics". In *Proceedings of the 2012 Winter Simulation Conference*, edited by C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A. Uhrmacher, 2887–2898. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Law, A. M., and W. D. Kelton. 2000. *Simulation Modeling and Analysis*. 3rd ed. Boston: McGraw-Hill.

Luke, S. 2009. *Essentials of Metaheuristics*. Lulu. Available for free at http://cs.gmu.edu/~sean/book/metaheuristics/.

Luke, S. 2011. "A Java-based Evolutionary Computation Research System". http://cs.gmu.edu/~eclab/projects/ecj/, Accessed at March 25, 2013.

Mühlenbein, H., D. Schomisch, and J. Born. 1991. "The Parallel Genetic Algorithm as Function Optimizer". *Parallel Computing* 17:619–632.

Paul, R. J., and T. S. Chanev. 1998. "Simulation Optimisation Using a Genetic Algorithm". *Simulation Practice and Theory* 6 (6): 601–611.

Rardin, R. L. 1998. *Optimization in Operations Research*. Upper Saddle River, NJ: Prentice Hall.

Russell, S. J., and P. Norvig. 2010. *Artificial Intelligence / A Modern Approach*. 3rd ed. Boston, Munich: Pearson Education, publ. as Prentice Hall.

Spratt, S. C. 1998. "Heuristics for the Startup Problem". Master Thesis, Department of Systems Engineering, University of Virginia.

Törn, A., and A. Zilinskas. 1989. *Global Optimization*, Volume 350. Springer-Verlag.

White Jr, K. P., M. J. Cobb, and S. C. Spratt. 2000. "A Comparison of Five Steady-State Truncation Heuristics for Simulation". In *Proceedings of the 2000 Winter Simulation Conference*, edited by J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick, 755–760. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

AUTHOR BIOGRAPHIES

MATTHIAS FRANK studied Computer Science at Technical University of Dresden until February 2013. This work is based on his diploma thesis. He is interested in simulation-based optimization related on mechanical engineering. His email address is matthias.frank@tu-dresden.de.

CHRISTOPH LAROQUE studied business computing at the University of Paderborn, Germany. From 2003 to 2007 he has been a PhD student at the graduate school of dynamic intelligent systems and, in 2007, received his PhD for his work on multi-user simulation. He is team leader of the "simulation & digital factory" at the chair of Business Computing, esp. CIM. He is mainly interested in the simulation-based decision support for operational production and logistic processes. His email address is laroque@upb.de.

TOBIAS UHLIG is a PhD student at Dresden University of Technology and a research assistant at the Universität der Bundeswehr München, 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.