Differential Evolution: A Method for Optimization of real Scheduling Problems

Martin Rüttgers[1]

TR-97-013

March 1997

ABSTRACT

A new method for optimization of scheduling problems with nonlinear objective functions and multiple dependent restrictions is presented. This method is based on an Evolutionary Algorithm but has a special changing operators for a leaded search over the entire solution space. It can be implemented for solving real problems very fast, it requires only few control variables, it is robust, easy to use and lends itself very well to parallel computation. The implementation to solve a model representing a real scheduling problem in foundries is presented. This application shows good results and the comparison a method based on a stochastic Evolutionary Algortihm, having the reputation for being very powerful, shows that the new method converges faster and with more certainty.

---

[1] FIR - Research Institute for Operations Management at Aachen University of Technology (RWTH), Pontdriesch 14/16, 52062 Aachen, Germany, e-mail: rt@fir.rwth-aachen.de. Research partially supported by the International Computer Science Institute, Berkeley, California and by the AiF, Proj.-Nr.: 10247.

PART I INTRODUCTION

Detailed planning as part of production planning processes is not very well supported by computer based methods. One reason for this fact is the complex structure of practical planning problems which are characterized by nonlinear objective functions and multiple dependent and time varying restrictions. Scheduling problems represent an important part of such problems and are of a major importance for the improvement of production process. Many different models exist for solving these problems but very often they are based on unreal assumptions (DOMSCHKE/SCHOLL/VOSS 1993). So the solution of the models is possible but not of the real problem.

On the other hand, customer requirements force many companies to optimize their production processes to reduce production times and costs. For this reason the optimization of scheduling processes in complex environments is an important field of research. The development of new methods is necessary which lead not only to optimized solutions but can also be adapted to real problems. These methods should be robust and easy to use.

The method of Differential Evolution considers these requirements (STORN/PRIECE 1995). Good optimization results can be achieved with short computation times. In this paper the model for a real scheduling problem, the concept of this method as well as first computation results are presented.

PART II APPLICABILITY OF EXISTING ALGORITHMS

In the field of Operations Research many heuristic methods have been developed to optimize scheduling processes. An extensive overview of these methods is given in the work of DOMSCHKE/SCHOLL/VOSS (1993, p. 249ff). In this book the authors mention the efficiency of some of these methods for solving complex scheduling problems but also emphasize the douptfulness of aplying these methods on real problems. This opinion is stressed by NISSEN (1995, p. 11ff). He gives two basic reasons for the afore mentioned problem:

- Most methods are based on special models not representing real situations and
- most methods are inflexible.They cannot be used while the underlying model changes in time which is very often necessary to represent real situations.

Because of these two facts, in the last years extensive research has been done in the area of Evolutionary Algorithms. These kinds of algorithms are by their nature very flexible in their application to different models. Many different types of these algorithms have been developed and used for optimization of scheduling processes. All these types are based on the same idea: A set of solutions is created and the solutions are changed in an iterative procedure by means of special operators. In each iteration the objective function for each solution is calculated and only the best solutions (by means of the objective function) are taken into the next iteration. The change of the solutions is similar to mutation and replication procedures, and the decision for taking a solution to the next iteration similar to a selection procedure. These basic ideas show the vicinity to evolution process in nature.

Many different types of Evolutionary Algorithms are described by NISSEN (1995, p.13ff). The main difference between them is based on the way the solutions are represented, the operators that are used for changing the solutions from one iteration step to the next and the objective function (SCHOENBURG/HEINZMANN/FEDDERSEN 1994).

In the last few years the application of these algorithms to combinatorial optimization problems have shown their efficiency (RECHENBERG 1973, p. 40ff, SCHWEFEL 1977). The work of MUSIER/EVANS (1989, p. 229ff), CHEN/VEMPATI/ALJABER (1996), LEE/KIM (1995), GRAHAM (et al. 1979), CHENG/SIN (1990) and NEPPALLI/CHUEN-LUNG/GUPTA (1996, p. 356ff) give many promising results. Without going into details, we can say that by the application of this algorithm to models representing real problems a new chapter of optimizing theory has began (REEVES 1993). But mainly two facts could be seen as critical for applications to real problems and made further research necessary.

- The speed of convergence is slow in many applications so that the necessary time to achieve good solutions becomes very large and
- many control parameters have to be adjusted to achieve good results which make the real use of the algorithms questionable.

From a theoretical point of view the first fact doesn't lead to any problem because Evolutionary Algorithms lend itself very well to parallel computation which allows the reduction of processing times. This is different for real scheduling problems: The computers being used in most real situations for the optimization don't support parallel processing.

Differential Evolution is a new kind of Evolutionary Algorithm which has been developed for optimization over continuous spaces (STORN/PRIECE 1995). The application to this kind of problems has shown that it converges faster and with more certainty than other methods. By its nature Differential Evolution is a flexible optimization procedure which can be used under multiple conditions. The implementation is easy and the method is easy to use. For this reasons further research is necessary to make the algorithm applicable to combinatorial optimization problems.

PART III CONCEPT

Differential Evolution represents an iterative solution procedure: A number of individuals are changed by special changing-operators and by the means of an objective function only the best individuals survive in the next generation. The main difference to existing Evolutionary Algorithms is the construction of the changing operator. The concept of the method is shown in Figure III-1.

First of all, the user configurates the system, i.e. he makes all the adjustments which are necessary to describe the existing situation. After creating start solutions, the objective function for each individual is calculated and the solution is compared to other solutions by means of the objective function. Only if the value of the objective function of the new solution is better than this of the other solutions the individuals will be taken into the next generation. If the criteria for stopping the algorithm are not fulfilled, the individuals will be changed by means of a mutation and recombination operator and the algorithm jumps back to the calculation of the objective function. At the end, the solution is presented.

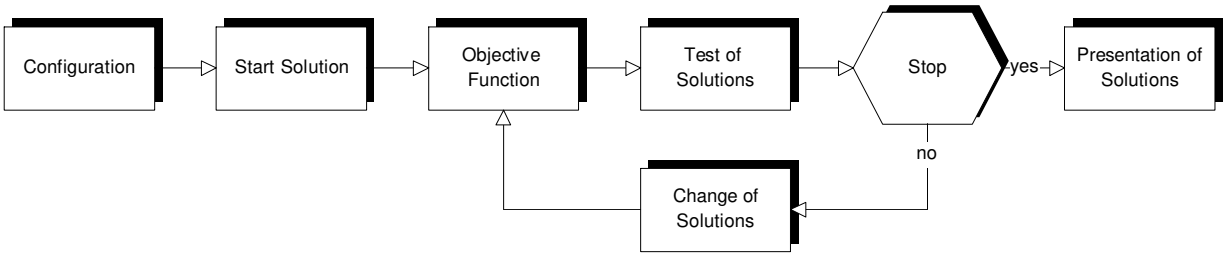

Figure III-1: Concept of method

The mutation operator for changing the individuals is used in each generation and the recombination operator only if a selected number of iterations no improvement of the best solution can be achieved.

The mutation operators are based on a rule which is not like the basic rules for other mutation operators only stochastic. By the concept of Differential Evolution the number of mutations depends on the "distance" between the individuals. This rule leads to a search over the entire solution space which depends on the shape of the objective function. If the individuals of one generation are spread over the whole solution space, then the individuals of the next generation will be the same. But if the individuals are concentrated at a special point in solution space, then the individuals of the next generation stay at this point until a solution somewhere else in solutions space can be found that has a better value of the objecitve function. This leads to a precise search of these parts of solution space promising good results.

Consider an individual being represented by a matrix A_j with

$$A_j = \begin{pmatrix} a_{11} & \cdots & a_{1l} \\ \vdots & \ddots & \vdots \\ a_{k1} & \cdots & a_{kl} \end{pmatrix} \text{ with } j = 1,\ldots,n \; .$$

The columns of the matrix represent the different machines on which the jobs should be scheduled and each line represents a minimum planning time unit Δt. The variable j counts the individuals of one iteration.

To calculate the distance between the individuals in solution space the Hamilton Distance $D(A_i,A_{i'})$ is used. It can be calculated by counting the number of differences in two matrices A_i and $A_{i'}$ with $i \neq i'$ by

$$D(A_i, A_{i'}) = \sum_{alle\ a_{k,l}} \Theta_{k,l,k',l'}$$

with $\Theta_{k,l,k',l'} = 1$ if $a_{kl} = a_{k'l'}$, otherwise $\Theta_{k,l,k',l'} = 0$.

A new individual A_{new} is generated from an old A_{old} by putting $C_{mut} \cdot D(A_i, A_{i'})$ stochastic changes on this individual. $C_{mut}>0$ is a constant which can be adjusted by the user. The changes can be done in two ways. Either $C_{mut} \cdot D(A_i, A_{i'})$ jobs are exchanged stochastically (then we talk about an exchange) or $C_{mut} \cdot D(A_i, A_{i'})$ jobs are inverted (then we talk about an inversion).

In both cases the mutation procedure can be expressed by

$$A_{new} = A_{old} + C_{mut} \cdot D(A_i, A_{i'}).$$

By the inversion a set of $C_{mut} \cdot D(A_i, A_{i'})$ jobs is chosen stochastically and the sequence is inverted:

$$a_{1,1}, a_{1,2}, \cdots, a_{n,m}, \underbrace{a_{n,m+1}, \cdots, a_{n,m+C_{mut} \cdot D(A_i, A_j)}}_{\text{Inversion}}, a_{n,m+C_{mut} \cdot D(A_i, A_j)+1}, \cdots, a_{k,l}.$$

The recombination operator on the other hand leads to a new solution by combining to solutions A_i and $A_{i'}$ with $i \neq i'$. The operator separates the pool of jobs of each individual A_i and $A_{i'}$, into two groups X and Y by

$$\underbrace{a_{1,1}, a_{1,2}, \ldots, a_{n,m}}_{X}, \underbrace{a_{n,m+1}, a_{n,m+2}, \ldots, a_{k,l}}_{Y}.$$

All jobs from group X remain in the new solution A_{new} at the old position of solution A_i. The jobs of group Y are put on this points in the new solution A_{new} which corresponds to the positions they had in solution $A_{i'}$.

PART IV REAL PROBLEM

1 Problem Formulation

The power of the new method is tested in its application to the scheduling problem of core blowers which is characterized by a nonlinear objective function and many complex, dependent and time varying restrictions. The problem is of major importance for foundries because core blowers are strongly connected to automatic molding plants with high production costs. If the cores are not produced early enough, then the molding plants must be stoped. On the other hand cores grow older and lose quality if they are produced to early. The scheduling problem model for core blowers in foundries can be described by the following premises:

Premise 1: The core blowing process is a four stage process. The stages are core-blowing, core-assembling, core-sleeking and core-drying.
Premise 2: The sequence of process stages is not changeable.
Premise 3: On each stage more than two parallel processors exist.
Premise 4: Some of the stages can be skipped for special jobs.
Premise 5: Not all processors can be reached bei each job.
Premise 6: Each processor can processe only one job each time.
Premise 7: Ressources of materials are not limited.
Premise 8: Special jobs can only be processed on special processors.
Premise 9: After each stage storage in a common depot with limited capacity is possible.
Premise 10: Production times are processor and job dependent.
Premise 11: Productions times are set at the beginning of scheduling process.
Premise 12: Set-up times are processor and job dependent.
Premise 13: Transportation times between the stages can be neglected.
Premise 14: Number of necessary workers depend on the processores.
Premise 15: Job-splitting is allowed.
Premise 16: Job-lapping is allowed.

Premise 17: Job-passing is allowed

The optimization of this model should be done on the job floor several times a day. In most cases there are more than 15 parallel, heterogeneous processors and more than 200 jobs to be scheduled. In order to avoid interrupt of production process, the optimization should be done within a couple of minutes. Hardware configuration is based on common PCs.

2 Module design

The configuration module consists of three major input-parts. In the first part the input of data describing the production situation is necessary:

- Setup of discirptive terms for each core blower: Set-up time, blowing frequency, necessary staff for blowing, production costs, set-up costs, machine load,
- capacities for core-blowing, core-assembling, core-sleeking, core-drying,
- number of working days, number of shifts per day, shift-duration,
- personnel cost,

In the second part, the input of data describing the different jobs is necessary:

- Type of finishing procedure (parallel or successive), rest time after finishing, due date, set-up time, blowing frequency, necessary storage capacity, assortment affiliation.

In the third part, the input of data describing the optimization process is necessary:

- Common Data: Type of initial solutions (stochastic, knowledge-based), number of changes in case of knowledge-based generation of initial solutions, smallest planning time, number of individuals per generation
- Mutation Operator: Changing degree (C_{mut}),
- Recombination Operator: Criteria for initialization of recombination, number of recombinations per iteration,

With this input data all necessary information are collected to start the optimization procedure. After configuration n initial solutions are generated. Depending on the configuration one of the following possibilities for generating the start solutions can be used:

- Knowledge-based Generation: In this case the user himself creates the first schedule A_1, i.e. he puts the jobs on special processors and determines the order of jobs on each processor. From this first start solution the other *n-1* solutions are generated stochastically. The number and the extend of this stochastic changes is given by the configuration.
- Stochastic Generation: In this case all n individuals A_j of the initial population are generated statistically, which means that the user has no influence of putting the jobs to special processors and to determine their order. Also here the number and extend of the stochastic changes is given by the configuration of the system.

In both cases of generating the initial solutions only those individuals can be generated which do not produce any conflicts with one of the restrictions. In the objective function the following aspects can be considered: Exceeding of capacities, deviation from due dates, production costs, set-up costs and deviation from assortment affiliation. Its structure is the following:

$$F_{total} = a_{Capacity} \cdot F_{Capacity} + a_{Date} \cdot F_{Date} + a_{Cost} \cdot F_{Cost} + a_{Set} \cdot F_{Set} + a_{Assort} \cdot F_{Assort}.$$

The variables $a_{capacity}$, a_{Date}, a_{Cost}, a_{Set} and a_{Assort} are constant and take the rating of the different aspects into account which can be changed by the user.

The variable $F_{Capacity}$ describes all exceedings of capacities from staff, storage-place, assembling, sleeking and drying. All this components where considered in the objective function. By calculating the number of planning units where the existing capacities does not satisfy the demand. If K_w with $w =$ *staff, storage, assembling, sleeking, drying* are these numbers then the total exceeding of capacity is

$$F_{Capacity} = C_{Capacity} \cdot \sum_{all\ w} \left(K_w\right)^{\mu_w} \text{ mit } C_{Capacity},\ \mu_w = \text{const.}.$$

Die variable F_{Date} describes the deviation from due date of all jobs in a schedule by calculating the total time difference between due dates and real finishing times:

$$F_{Date} = C_{Date} \cdot \sum_{j} \left(t_{due,j} - t_{real,j} \right)^{\mu} \text{ with } C_{Date},\ \mu = \text{const.},\ j = j\text{th job.}$$

The variable F_{Cost} describes the production costs of the core blowers. If K_i are the production costs for the ith machine and if $\Theta_i(t)=1$ only if the ith machine is occupied by a job, otherwise $\Theta_i(t)=0$, then:

$$\mathrm{F_{Cost}} = C_{Cost} \cdot \sum_{i} \left(K_i \cdot \Theta_i(t) \right)^{\mu} \text{with } C_{Cost},\ \mu = \text{const.},\ i = i\text{th machine.}$$

The variable F_{Set} describes the set-up costs. If R_i are the set-up costs for the ith machine per set-up process and if Z_i counts the number of different set-up processes on the ith machine, then:

$$F_{Set} = C_{Set} \cdot \sum_{i} \left(R_i \cdot Z_i \right)^{\mu} \text{ with } C_{Set},\ \mu = \text{const.},\ i = i\text{th machine.}$$

The variable F_{Assort} describes the affiliation of all cores to different assortments by calculation the time differences between the finishing times and due-dates of all cores of one assortment. If S_v are the different assortments, then:

$$F_{Assort} = C_{Assort} \cdot \sum_{v} \sum_{j \in S_v} \left(t_{due,j} - t_{real,j} \right)^{\mu} \text{ with } C_{Assort},\ \mu = \text{const.},\ j = j\text{th job.}$$

For changing the individuals the described mutation- and recombination-operator are used. The objective function is formulated in a way that it becomes minimized by the algorithm:

$$F_{total} = Min.$$

By the means of this a new individual is taken into the next iteration, if the value of the objective function is reduced in comparison to the value of the old individual:

$$F_{total}(A_{old}) \geq F_{total}(A_{nwu)}.$$

The algorithm terminates if one of the following conditions is fulfilled:

- A number of generations is generated,
- the reduction of the objective function over the last hundred generations is less than a minimum amount.

The results of an optimization procedure are presented in a Gantt-Diagram which is a graphical presentation of the matrix A_i of the individuals with the smallest value of the objective function. In addition to this, the values of considered staff and machine capacities are shown over the entire planing period. So the user is warned of bottlenecks.

PART V RESULTS AND CONCLUSION

The described algorithm is tested on real problems in two different foundries. The results are compared with an Evolutionary Algorithm with a stochastic mutation operator and planning results of a manual planning procedure which is standard in most foundries. For a small number of jobs the optimum solution of the problem can be calculated and handled as a lower bound for the problem. The results are shown in Figure V-1

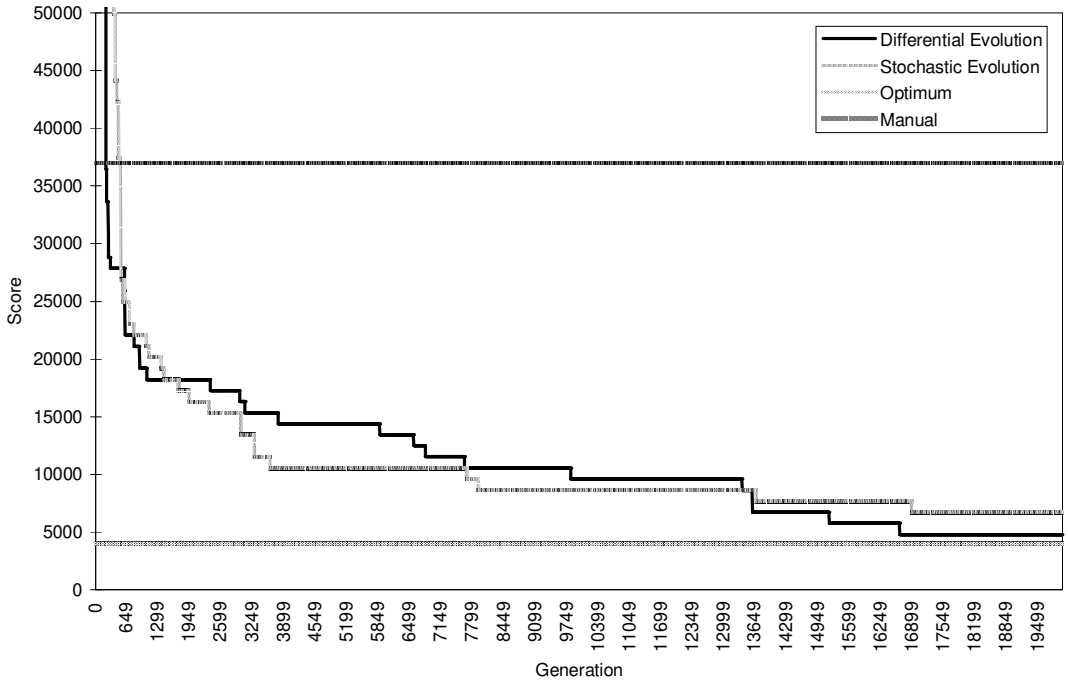

Figure V-1: Optimization results (5 machines, 30 jobs)

The results show that the new method converges faster than a normal Evolutionary Algorithm with stochastic based changing operator. We found out that the solution of manual planning process can be improved from 37000 to about 5000. For the problem, the lower bound can be calculated as 4700. After 17000 iterations which corresponds to a computation time of about 10 minutes on a pentium processor the method finds its best solution which is close to the lower bound.

In addition to this, the personnel in foundries had no problems in using the algorithm instead of manual planning. The most important reason for this is the flexibility of the algorithm which makes adaptation to changing situations because of new jobs or machine breakdown very easy.

PART VI REFERENCES

CHEN, C.; VEMPATI, V.S.; ALJABER, N.: An application of Genetic Algorithms for the flow shop problems. In: European Journal of Operational Research, 80 (1995), S. 389-396.

CHENG, T.C.E.; SIN, C.C.S.: A state-of-the-art review of parallel-machine scheduling research. In: European Journal of Operational Research, 47 (1990), S. 271-292.

DOMSCHKE, W.; SCHOLL, A.; VOß, S.: Produktionsplanung. Ablauforganisatorische Aspekte. Springer, Heidelberg 1993.

GRAHAM, R.L.; LAWLER, E.L.; LENSTRA, J.K.; RINNOOY KAN, A.H.G.: Optimization and approximation in deterministic sequencing and scheduling: A survey. In: Annals of Discrete Mathematics, 5 (1979), S. 287-326.

LEE, C.; Parallel Genetic Algorithms for the earlyness-tardiness job scheduling problem with general Penalty Weights.

KIM, S.: In: Computers Industrial Engineering, 28 (1995) 2, S. 231-243.

MUSIER, R.F.H.; EVANS, L.B.: An approximate method for the production scheduling of industrial batch processes with Units. In: Computers Chemical Engineering, 13 (1989) 1/2, S. 229-238.

NEPPALLI, V.R.; CHEN, C-L.; GUPTA, J.N.D.: Genetic Algorithms for the two-stage bicriteria flowshop problem. In: European Journal of Operational Research, 95 (1996), S. 356-373.

NISSEN, V.: Evolutionäre Algorithmen. Darstellung, Beispiele, betriebswirtschaftliche Anwendungsmöglichkeiten. Deutscher Universitätsverlag, Wiesbaden 1994.

RECHENBERG, I.: Evolutionsstrategie. Frommann-Holzboog, Stuttgart 1973.

REEVES, C.R.: Modern heuristic techniques for combinatorial problems. John Wiley & Sons, Oxford 1993.

SCHÖNEBURG, E.; HEINZMANN, F.; FEDDERSEN, S.: Genetische Algorithmen and Evolutionsstrategien. Eine Einführung in Theorie and Praxis der simulierten Evolution. Addisson-Wesley, Bonn, Paris 1994.

SCHWEFEL, H.-P.: Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie. Birkhäuser Verlag, Basel, Stuttgart 1977.

STORN; PRIECE: Differential Evolution - A simple and efficient adaptive scheme for global optimization over continuos spaces. International Computer Science Institute 1995, Technical Report TR-95-012.