

Applying Genetic Algorithms to the Optimization of Production Planning in a Real-World Manufacturing Environment

Roland Braune, Stefan Wagner, Michael Affenzeller

Institute of Systems Theory and Simulation

Johannes Kepler University

Altenbergerstrasse 69

A-4040 Linz, Austria

email: roland.braune@gmx.at, {sw,ma}@cast.uni-linz.ac.at

Abstract

In this paper we present an optimization approach to a real-world production planning problem. Based on raw data that have been extracted from the Production, Planning and Control System of a company which produces special purpose vehicles and equipment, we have developed an architecture of an optimization system for production planning and scheduling in the manufacturing line of this company. The paper itself is divided into two major parts. The first part mainly deals with the theoretical background of production planning problems. In the second part of the paper we give an overview of the concrete scenario which is the subject of our research. Based on these fundamentals, we describe our approach to the problem, the modelling process and the architecture of the optimization system we plan to implement.

1 Introduction

Many modern industrial production environments still rely on pure expert knowledge for production planning and scheduling purposes. Optimization in an algorithmic sense is rare and mostly not performed at all. This may seem amazing at first glance since automation and electronic data processing have pervaded modern companies in almost any area of their daily business. For manufacturing lines in particular, there exist so called *Production, Planning and Control Systems* that serve for data storage and as a support tool for production related management activities. Such systems are designed in a highly generic and versatile way in order to be applied in many different companies no matter what products they actually manufacture. The systems try to distill the most common and essential aspects of production data. Customization is of course possible but it mostly requires deep insight and an immense effort.

Therefore, the concept of Production, Planning and Control Systems is, at present, not applicable for the optimization of production planning and scheduling in most of the cases. On the one hand a possible optimization approach will have to consider *especially* the

domain dependent characteristics, specific properties and constraints that distinguish production systems from each other. On the other hand, the objectives of optimization may vary from company to company, they can be of various different kinds (makespan minimization, maximizing machine occupation, etc.), individually or combined, equally weighted or associated with priorities. All these facts make it very difficult to develop a generic optimization tool that meets the manifold requirements of the production industry.

The problem of (optimal) production planning and scheduling is not new, however. Due to its increasing relevance, this issue early attracted the attention of academic research. Yet in 1960, Giffler and Thompson described the solution of production scheduling problems from an algorithmic point of view [Giffler and Thompson, 1960]. Considerable theoretical advances have been made since that time, leading to the research area of *Job Shop Scheduling Problems* (JSSP) [French, 1982] which provides the fundamental theory and mathematical methodology for production planning and scheduling problems in general. The JSSP and similar scheduling problems are combinatorial optimization problems and commonly classified as NP-hard ordering problems [Garey and Johnson, 1979]. Due to the NP-hardness it is almost impossible to solve these problems exactly, even for relatively small problem instances. Exact methods exist, like the branch and bound approach by Carlier and Pinson [Carlier and Pinson, 1989], but they are only of theoretical relevance due to their exponential runtime complexity. In real-world production environments, provable optimality is not the criteria a solution must satisfy. Instead it suffices to compute results close to the optimum but in a reasonable amount of time. Exactly the latter is achieved by heuristic methods that are dominating in the field of JSSP. The most important and popular ones are Local Search approaches like Tabu Search [Barnes and Chambers, 1995] and Evolutionary Algorithms, especially Genetic Algorithms (GA) [Yamada and Nakano, 1997].

One may think that based on well-established theory and a multitude of existing solution techniques, the development of a production planning optimization tool for a specific industrial application will proceed smoothly and in a straight-forward manner. This

is, however, a misconception. Real-world scenarios like the one at hand usually tend to differ tremendously from artificial and "designed" JSSP benchmark problems. The standard Job Shop Scheduling Problem in general is based on the formal description of a highly simplified and abstract production system (cf. Section 2). Hence some difficulties may arise when trying to describe an existing, concrete production system in a way that is precise enough for the problem to be analyzed and processed furtherly by existing and possibly customized methods. In the case of our project these difficulties were one of the main challenges especially in the initial phase of our work (cf. Section 3).

Besides the problem analysis, our solution approach itself will be the second important aspect covered in the paper (cf. Section 4). We will describe the modelling and the architecture of the target system, optimization specific details such as the chosen method and problem encoding issues. Furthermore, we will outline the planned implementation phase in a short manner (cf. Section 5).

2 The Standard Job Shop Scheduling Problem

In order to provide a common basis for the following sections, we describe the most elementary concepts and formalisms with respect to the standard Job Shop Scheduling Problem (JSSP).

The classic $n \times m$ *minimum-makespan* Job Shop Scheduling Problem is given by a finite set J of n jobs $\{J_i\}_{1 \leq i \leq n}$ and a set M of m machines $\{M_i\}_{1 \leq i \leq m}$ [Jain and Meeran, 1999]. Each job J_i has to be processed on every machine. For this purpose, a job is subdivided into a set of m_i operations $\{o_{ij}\}_{1 \leq i \leq n, 1 \leq j \leq m_i}$ which have to be scheduled in a strictly sequential way according to a given technological order, also referred to as the *precedence constraint*. Thus o_{ik} denotes the operation of job J_i that has to be processed on machine M_k for a certain uninterrupted processing time τ_{ik} where each machine can process only one job at a time (*capacity constraint*). The time span needed to complete all operations of all jobs is known as the *makespan* C_{max} . Given the *starting times* $t_{ik} \geq 0$ for each operation the definition of the minimum makespan C_{max}^* with respect to all feasible schedules can be informally written as

$$C_{max}^* = \min(C_{max}) = \min_{\text{feasible schedules}} (\max(t_{ik} + \tau_{ik}) : \forall J_i \in J, M_k \in M).$$

In order to illustrate the formal specification we will present now an example of a 3×3 JSSP (3 jobs, 3 machines). The technological order of operations (machine sequence) is shown in Table 1. The first number in each column represents the machine M_i on which the operation has to be processed whereas the number within the parentheses denotes the required processing time t_{ik} . Figure 1 depicts one out of many feasible schedules for this problem instance. This kind

Table 1: A 3×3 JSSP instance

Job	Machine Sequence		
1	1 (3)	2 (3)	3 (3)
2	1 (2)	3 (3)	2 (4)
3	2 (3)	1 (2)	3 (1)

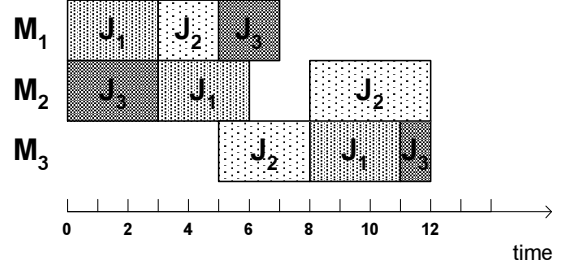


Figure 1: A schedule for a 3×3 JSSP instance

of graphical representation is referred to as the *Gantt Chart* of a schedule.

3 Description of the Actual Problem

3.1 The Main Entities of the Production System

- *Customer Orders:*

Customer orders are the most important elements in the production system and the company as a whole. They determine *what* has to be produced and *when* it has to be delivered. In our case, we distinguish two main types of customer orders: (1) vehicle orders and (2) component (equipment) orders.

- *Production Jobs:*

A production job is the top level entity in the production system. The purpose of production jobs is the manufacturing of *parts*. Hence they define the final product, the amount to be produced and a due date. There is a clear analogy between production jobs and customer orders and we will also regard customer orders as jobs. However, the main difference is that the scope of production jobs is restricted to the internal production management. A production job never directly corresponds to a customer order nor does it necessarily have to be related to a particular one. Production jobs just produce parts, either to satisfy needs that arise from customer orders such as (sub-)components or they simply serve for stocking purposes.

- *Production Processes:*

For the stepwise accomplishment of a production target, both customer orders and production jobs involve production processes which have to be performed one by one according to a given sequence. Each process is to be performed on one

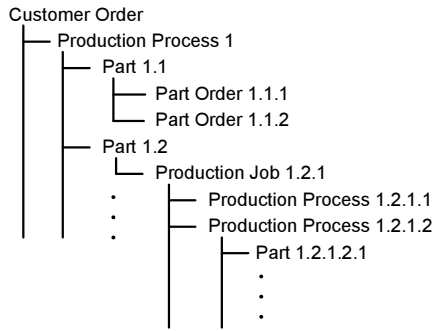


Figure 2: Tree-like view on the production system

single workplace by a specified number of workers. It has a definite duration (workload) and may require a setup time. Usually, a production process is also associated with a list of parts which have to be available before the process can start.

- **Parts:**

As already mentioned above, the availability of parts can be a prerequisite for the execution of production processes. Hence parts and their well timed availability are issues that are of central importance in the production system. There exist two different types of parts: Production parts on the one hand are manufactured in the company itself. Typically, one production part can be associated with one or more production jobs, possibly with different due dates. Purchase parts on the other hand are ordered from suppliers. A list of one or more orders may exist for one purchase part determining amount and date of each corresponding delivery. Of course there is a stock for both kinds of parts and usually most of the parts are in stock up to a certain amount.

Certain complex products such as vehicles are manufactured not only using ready-made parts but also composite parts which have to be assembled themselves from other parts. These parts can again be simple parts or composite parts. The level of nesting can be arbitrarily deep, depending on the complexity of the final product. This fact suggests a tree-like view onto the production system as shown in Figure 2.

3.2 Workplaces and Allocation Scheme

Workplaces are characterized by a definite per-day capacity in hours which is the sum of the capacities of all workers assigned to this workplace. A single workplace might be able to perform *more than one* production process at the same time if its capacity and the number of available workers is sufficient.

3.3 Optimization Objectives

The main task of our optimization tool for the given manufacturing system is to find an optimal schedule or, in other words, an optimal sequence of production processes for each workplace. The optimization objectives can be formulated as follows: (a) Minimizing

the mean deviance from the customer orders' predetermined delivery dates, thus the mean earliness and tardiness and (b) maximizing the workplace utilization.

4 Solution Approach

In this section, we will present our solution approach to the problem described above. As our optimization system is meant to be purely software-based, we discuss the modelling and architectural aspects in a somewhat software-oriented manner.

4.1 Data Representation and Abstractions

In Section 3.1, we have already identified the main entities of the production system as well as its structure. For software based processing, it is necessary to represent these data in an adequate way. Due to the fact that we have chosen an object oriented programming language for our optimization tool, we model the main entities as *classes*. This offers several advantages: Firstly, we reduce complexity and save memory by including only the *relevant* attributes of a customer order, production job, part or part order. Secondly, by mapping each customer order, production job, part and part order in the production system to an *object* (instance) of the corresponding class, we can easily represent the tree-like structure of the production system by linking objects appropriately. Finally, by transforming the raw data from the company's production planning and control system into a structure of objects, we have created the data basis for our software system. As we will describe in the following, this data basis serves as the starting point for further abstractions in our architecture.

It is evident that our input data differs significantly from the data layout the standard JSSP is based on. On the one hand, the standard JSSP only knows *Jobs* and *Operations* and on the other hand, jobs are autonomous in the sense that there are no sequential dependencies between two or more of them. Existing approaches to job shop scheduling problems like the ones we will use (in a modified form) rely on this standard data scheme. The problem was to modify the structure of our input data in order to conform at least basically to the standard scheme. As already stated in Section 3.1, there is a close relationship between customer orders and production jobs, both incorporate production processes and both have a production target. Therefore we make an abstraction here and introduce a class *Job* representing either a customer order or a production job. The associated production processes are also described by a corresponding class, namely *Operation*. However, for better understanding, we will still use the terms "customer order", "production job" and "production process" where appropriate.

One of the most important issues that remain, is the correct reproduction of the tree-like structure of the production system, thus the dependencies between jobs that emerge from the possible demand for (com-

plex) subcomponents in production processes. Using a real tree of objects as the corresponding representation, hence a nested structure of *Job*-instances, would again violate the restrictions imposed by the standard JSSP data model. The idea is to resolve the dependencies in question through mere temporal relationships or, to be more precise, by providing each production process with a starting time. This starting time may be affected by several factors (assuming that the production process requires at least one part for being started): If all required parts are in stock, the production process can start immediately. In the contrary case, as far as purchase parts are concerned, we are confined to the fixed delivery dates of the existing part orders, so we rely on absolute time information. Concerning production parts that are not in stock and thus launch a separate production job, the situation is more complicated. The new job has to be finished in order for the part to become available, but the finish time of the job cannot be determined in advance. This is a characteristic of the schedule building algorithm (see Section 4.2), which is executed in a step-wise way and only "sees" the operations that are *currently* schedulable at a certain time. If we launch a new production job, it will take the schedule building algorithm a particular number of steps until all operations of the new production job are scheduled. Now the part is available, and the production process that has been waiting for the part can possibly start (become schedulable), of course only if all other part requirements are supplied as well.

It is a matter of fact that production processes belonging to customer orders have to be scheduled primarily and the ones associated with production jobs (usually) will not become schedulable until they produce a part needed by another production process already considered by the schedule building algorithm. Therefore we categorize jobs by their scheduling priority. Customer orders are regarded as *primary jobs*, the set of *secondary jobs* consists of all jobs that produce a part required by another job's production process. The *tertiary jobs* at last, are formed by the remaining jobs that either still have no other job to produce a part for or just serve for stocking purposes. The distinction between primary and secondary jobs is a matter of convenience. It simplifies the test for the quality of a solution which is mainly influenced by the finish dates of the customer orders as already stated in Section 3.3. The exact way how the schedule building algorithm proceeds with these three categories of jobs will be the subject of the following section.

4.2 Schedule Building

In 1960, when Giffler and Thompson published their article on algorithms for solving production scheduling problems [Giffler and Thompson, 1960], optimization was not yet of such a great importance as it is nowadays. The primary objective was to find a correct schedule for operations of given jobs in consideration of precedence and capacity constraints (in the sense of the standard JSSP for example). Of course, the resulting schedules are of practical quality though they are

1. Put the first schedulable operation of each job into the set C .
2. Choose an operation o' from C with the earliest possible starting time.
3. Determine the machine M' on which o' has to be processed and build the set G consisting of all operations in C executing on M' .
The set G is called the *conflict set* for machine M' .
4. Remove operations which start later than o' from the set G .
5. Select one operation from the conflict set G , place it into the schedule and remove it from the set C .
6. Insert the successor of operation o' into the set C (if available).
7. If $C \neq \emptyset$ goto step 2.

Figure 3: The non-delay scheduling algorithm

in no way optimal with respect to a particular objective. The classic Giffler and Thompson algorithm, also called the GT algorithm, produces schedules in which no operation can be scheduled earlier without delaying some other operation. Schedules of this kind are called *active* schedules. The constraints active schedules have to satisfy can even be furtherly reinforced. The so called *non-delay* schedules are active schedules in which operations are arranged such that the machine idle time is minimized, meaning that no machine (or workplace) is ever left idle if one or more operations are schedulable on that very machine. For the minimum-makespan JSSP described in Section 2, the optimal solution is known to be among the set of active schedules. In the case of our optimization objectives (see Section 3.3), we claim that the optimal solution will be found in the set of non-delay schedules. The non-delay scheduling algorithm is structurally similar to the GT algorithm and its mode of operation is outlined in Figure 3.

In the architecture of our optimization system the schedule builder is modelled as a separate class. As an input, the schedule builder takes jobs of the different categories described in the preceding section (see Section 4.1) and the output schedule represents a by-the-day allocation plan for each workplace in the manufacturing line. Operations of primary jobs and secondary jobs are tried to be scheduled whenever possible whereas operations of tertiary jobs become scheduling candidates only if there is no other operation schedulable and thus free capacities are available. Of course it is also the task of the schedule builder to check the availability of parts for each operation (production process) and to determine whether a new production job is required for a part and thus has to be migrated from the group of tertiary jobs to the group of secondary jobs.

4.3 Optimization Aspects

The most essential aspect of non-delay (and also active) schedule building is the resolution of the conflict between one or more jobs on a machine [Vazquez and Whitley, 2000]. The standard mechanisms for conflict set resolution are of a heuristic nature and can either be random in the simplest case or involve fixed *scheduling rules (priority rules)* [Panwalkar and Iskander, 1977; Morton and Pentico, 1993] for choosing one job from the conflict set. These techniques are, however, generally insufficient for finding an optimal or near-optimal solution since searching for it would simply mean trying out different random seeds or scheduling rules in a completely undirected and in no way target-oriented fashion. What simple mechanisms are not able to perform, can be achieved by classic optimization methods as they were already mentioned in Section 1 in the context of the JSSP. They provide a directed search and a stepwise improvement of solutions in a systematic way.

As far as our solution approach is concerned, we have chosen Genetic Algorithms (GA) [Holland, 1975] as the underlying heuristic optimization concept. Intensive research related to GA approaches to job shop scheduling problems has been done in the last decade and therefore we can build upon well developed theoretical fundamentals and practical experiences in this area. As the quality of solutions a GA might generate is heavily affected by the problem encoding and the associated crossover operators, these issues were one of the most important theoretical constituents of our work until now. In fact it is the *solution* of the problem rather than the problem itself which is encoded for the genetic algorithm, since - speaking in the terms of GAs - an individual corresponds to a solution. However, in the case of scheduling problems the term "solution" may have two different meanings. Many GA based approaches are designed in a way that a solution in the sense of the genetic algorithm only provides a guidance for resolving the conflict set on a given machine such that the resulting schedule - the solution of the JSSP itself - becomes preferably optimal. Some other encodings are based on the direct representation of an (active) schedule by an individual and some variants even rely on completely different scheduling building procedures. According to [Yamada and Nakano, 1997], the existing encoding variants can be roughly classified into the categories presented in Table 2.

We assessed the existing encoding schemes for suitability for our special problem, their practicality and simplicity (ease of use) and their overall performance. We came to the conclusion that permutation representations like the *job sequence matrix* and the *permutation with repetition* were the most promising ones and fit best into the scope of our problem. However, we cannot definitively decide in advance which GA approach will in fact yield the best results afterwards. For this reason the schedule building part of our architecture has been designed to provide independence of a certain optimization method in general and of a

Table 2: Classification of common JSSP representations for GAs

Category	Representatives / Crossovers
Binary representation	Disjunctive graph based encoding
Permutation representation	Job sequence matrix, Permutation with repetition
Heuristic crossover using an active schedule builder	GT based crossover
Genetic enumeration	Priority rule based GA, Shifting bottleneck GA
Genetic local search and multi-step crossover fusion	Neighbourhood search crossover, MSXF

special encoding in particular. Of course we are restricted to optimization methods and encodings that have been developed to be applied to the classic non-delay (or GT) schedule building process.

The proposed architecture is framework-like: The schedule builder itself is able to use different implementations of an interface named *ConflictSetResolver*. Concrete implementations such as a job sequence matrix based one can be easily "plugged" in by calling the schedule builder with an instance of the corresponding class as an argument. The schedule builder generates a schedule guided by the conflict set resolving mechanism a job sequence matrix provides for example. The schedule is evaluated by a *fitness function* and the computed quality value is the fitness value of the associated individual, thus the actual job sequence matrix for instance.

As far as crossover operators are concerned, we plan to implement at least the most popular variants that already exist for the respective encoding scheme. In case of the job sequence matrix, these will be the *subsequence exchange crossover* (SXX) [Kobayashi *et al.*, 1995] and the *job based order crossover* (JOX) [Ono *et al.*, 1996]. In [Bierwirth, 1995], Bierwirth suggests a modified order crossover (OX) operator, the *Generalisation of OX* (GOX), for his permutation approach. We refer to the literature for further details on the crossover operators mentioned above.

5 Further Proceeding

In Section 4 we have discussed the modelling and architectural aspects of our optimization system. However, the implementation itself is still in the starting phase and a first running version is meant to be finished within the next few months. Nevertheless we will give a brief outline of the most important implementation aspects in the following.

As a programming language we have chosen C#, for several reasons. Firstly, C# is object-oriented and provides features that are essential for the realisation of our architecture. Secondly, C# shows good performance, even compared to C++. Finally, using C# enables us to build upon an existing software framework for heuristic optimization algorithms. This framework, called *HeuristicLab*, is still under development and the current version already includes a broad spectrum of ready-to-use genetic algorithm variants that can easily be integrated into individual applications. Thereby we have the advantage to be able to use the most recent advanced GA concepts e.g. SASEGASA [Affenzeller and Wagner, 2003] which has been designed to avoid premature convergence by sophisticated selection pressure steering mechanisms and yields excellent results despite its generic and problem independent nature.

Due to the huge amount of input data caused by the production system we intend to optimize, we are clearly aware of the high computational complexity of the associated optimization process. Therefore it is a long-term target to extend our optimization tool with regard to parallel processing on multiple computation facilities. The most obvious underlying parallel computing concept for this purpose is *Grid Computing*. In the context of our optimization system it would be conceivable that grid clients run e.g. on the company's office computers during the night when they are normally idle. This is a quite realistic scenario and we are optimistic that the company's organisational policies will be in our favour in this matter.

References

- [Affenzeller and Wagner, 2003] M. Affenzeller and S. Wagner. SASEGASA: An evolutionary algorithm for retarding premature convergence by self-adaptive selection pressure steering. In *Computational Methods in Neural Modeling*, Lecture Notes of Computer Science 2686, pages 438–445. Springer Verlag, Berlin, 2003.
- [Barnes and Chambers, 1995] J.W. Barnes and J.B. Chambers. Solving the job shop scheduling problem with tabu search. *IIE Transactions*, 27:257–263, 1995.
- [Bierwirth, 1995] C. Bierwirth. A generalized permutation approach to job shop scheduling with genetic algorithms. *OR Spektrum*, 17:87–92, 1995.
- [Carlier and Pinson, 1989] J. Carlier and E. Pinson. An algorithm for solving the job-shop problem. *Management Science*, 35(2):164–176, 1989.
- [French, 1982] S. French. *An Introduction to the Mathematics of the Job Shop*. Chichester, Horwood et al., 1982.
- [Garey and Johnson, 1979] M.R. Garey and D.J. Johnson. *Computers and Intractability, a Guide to the Theory of NP-Completeness*. W. H. Freeman Company, San Francisco, CA, 1979.
- [Giffler and Thompson, 1960] B. Giffler and J.L. Thompson. Algorithms for solving production scheduling problems. *Operations Research*, 8:487–503, 1960.
- [Holland, 1975] J.H. Holland. *Adaption in Natural and Artificial Systems*. University of Michigan Press, 1975.
- [Jain and Meeran, 1999] A.S. Jain and S. Meeran. Deterministic job-shop scheduling: Past, present and future. *European Journal of Operational Research*, 113(2), 1999.
- [Kobayashi et al., 1995] S. Kobayashi, I. Ono, and M. Yamamura. An efficient genetic algorithm for job shop scheduling problems. In *Proceedings of the 6th ICGA*, pages 506–511, 1995.
- [Morton and Pentico, 1993] T.E. Morton and D.W. Pentico. *Heuristic Scheduling Systems*. John Wiley and Sons, 1993.
- [Ono et al., 1996] I. Ono, M. Yamamura, and S. Kobayashi. A genetic algorithm for job-shop scheduling problems using job-based order crossover. In *Proceedings of 1996 ICEC*, pages 547–552, 1996.
- [Panwalkar and Iskander, 1977] S.S. Panwalkar and W. Iskander. A survey of scheduling rules. *Operations Research*, 25(1):45–61, 1977.
- [Vazquez and Whitley, 2000] M. Vazquez and L.D. Whitley. A comparison of genetic algorithms for the dynamic job shop scheduling problem. In *Proceedings of GECCO-2000*, pages 169–178, 2000.
- [Yamada and Nakano, 1997] T. Yamada and Ryohei Nakano. Genetic algorithms for job-shop scheduling problems. In *Proceedings of Modern Heuristic for Decision Support*, pages 67–81, London, March 1997. UNICOM seminar.