Agile factorial production for a single manufacturing line with multiple products

Wolfgang Garn,James Aitken

The Surrey Business School, University of Surrey, Guildford, Surrey, GU2 7XH, United Kingdom

Abstract

Industrial practices and experiences highlight that demand is dynamic and non-stationary. Research however has historically taken the perspective that stochastic demand is stationary therefore limiting its impact for practitioners. Manufacturers require schedules for multiple products that decide the quantity to be produced over a required time span. This work investigated the challenges for production in the framework of a single manufacturing line with multiple products and varying demand. The nature of varying demand of numerous products lends itself naturally to an agile manufacturing approach. We propose a new algorithm that iteratively refines production windows and adds products. This algorithm controls parallel genetic algorithms (pGA) that find production schedules whilst minimizing costs. The configuration of such a pGA was essential in influencing the quality of results. In particular providing initial solutions was an important factor. Two novel methods are proposed that generate initial solutions by transforming a production schedule into one with refined production windows. The first method is called factorial generation and the second one fractional generation method. A case study compares the two methods and shows that the factorial method outperforms the fractional one in terms of costs.

Keywords: lot-sizing; production schedule; ELSP; genetic algorithm

1. Introduction

Food manufacturers are experiencing increases in the variability of demand received and the variety of products required (Squire et al., 2009). It is imperative for operations to manage the increasing dynamics of the situation to avoid excessive inventory. Through the application of an agile approach food manufacturers are increasing the flexibility of their operations and finished goods stocks to meet customer demand (Taylor and Fearne, 2006). This increased flexibility supports the production of goods as they are needed. The resulting manufacturing schedules are directed by the objective to minimize the holding, setup and shortage cost. This study focuses on a single, agile, production line accommodating multiple products. This single production line perspective is a reality for many industries. However, for several it is a simplification. Nevertheless the resulting production schedule is a useful strategy that can be used to derive a refined schedule.

In this article we discuss the literature on lot-sizing models and its relationship to agile manufacturing. The classic Economic Order Quantity (EOQ) model is the starting point of the review, which leads to the Economic Lot Sizing Problem (ELSP) and finishes with a review of research on the Stochastic ELSP (SELSP). The analyzed classifications are shown in figure 1. This section also defines some of the later used terminology.

Our overall approach is to use lot sizing techniques to derive production schedules. A *lot-size* is the

☆

Tel.: +44(0)1483 68 2005; fax: +44(0)1483 68 9511.
Email address:
{w.garn,james.aitken}@surrey.ac.uk (Wolfgang Garn,James Aitken)

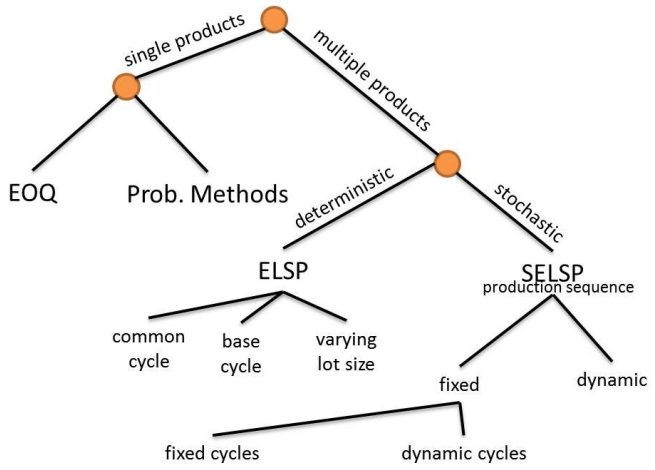

Fig. 1. Literature classifications.

quantity of units that is (was, will be) produced for one or more products. A *product* is raw material that is manufactured into a finished good. Products are sometimes referred to as items.

Lot sizing techniques are often divided into two categories. The first category is concerned with individual-product manufacturing and inventories. The second one looks at multiple products (Silver et al., 1998). Lot-sizing techniques for the single product category have been extensively studied. Harris (1913); Harris and Sollis (2005) introduced the EOQ model, which finds the optimal order quantity Q in terms of minimizing order cost $\frac{D}{Q}c_o$ and holding costs Qc_h subject to several constraints such as constant demand D. Note that order cost in a manufacturing context is the setup cost. A number of EOQ derivatives were introduced using additional factors such as shortage costs, production rates and many more. Erlenkotter (1990) gives an interesting documentation of the EOQ history. The original EOQ models were extended to incorporate focus on probabilistic demand (predominantly assuming normal distributed demand). Policies such as Order-Point & Order-Quantity(s,Q), Periodic-Review & Order-Up-To-Level (R,S) and (R,s,S) became common practice. The (s,Q) system is a continuous review system with a fixed quantity Q which is ordered whenever inventory drops below s. The (R,S) system is a replenishment cycle system where the inventory is increased to level S every R time units. The (R,s,S) system is a combination of the previous two systems. Every R time units the inventory level is checked, and when it is below s the level is raised to S. Details of these methods are explained in Silver et al. (1998).

In food manufacturing environments the multiple product category is typical challenging the capability and flexibility of the organizations to forecast demand (Christopher, 2011). The ELSP is an extension of the EOQ model that supports several products with differing demand D_i and also introduces change over costs c_{ij} from product i to j. The first popular work was done by Rogers (1958) and is now known as an **Economic Lot Scheduling Problem (ELSP)**. The usual objective of the ELSP is to find cyclical scheduling policies such that the setup and holding costs are minimized. The assumptions are that there is little demand fluctuation (no seasonality and trend) for each D_i and that there is a single machine that allows only one product at a time to be produced. Cyclical means a repetition of the production. In a two product scenario there will be production times p_1 and p_2 and setup (change over) times s_{12} and s_{21}. The timings for a possible production cycle are $p_1\ s_{12}\ p_2\ s_{21}$, which is then repeated. To be more precise the *production time* for one product is the amount of time required for all machines in use to manufacture a specified quantity of the product, including the run time and down time. The *run time* is the time machines are active. The down time is used to describe any possible machine inactivity. That means the cycle time is longer than the production time.

Silver et al. (1998) introduce an approach which finds a feasible solution for the ELSP. Osman and Demirli (2012) have recently contributed to the classic ELSP work. They gave a quadratic assignment formulation and introduced an algorithm that found optimal solutions - even for large problem instances. The ELSP is non-deterministic polynomial-time (NP) hard as Hsu (1983) proved. The recent review of the ELSP done by Chan et al. (2013) suggests classifications of the ELSP according to the schedule cycles. Their investigation showed that the common cycle approach is most often found in literature (41%). That means there is a single cycle for all products. The next significant class is based on the basic period approach with a frequency of 28%. Here each product has it's own cycle time, but each

cycle must be a multiple of the basic period. The third major approach has varying lot sizes per production cycles, but only got 11% attention in the literature (Chan et al., 2013). From an agility perspective the varying lot size (VLS) is particularly relevant to manufacturers in an agile environment due to its practical applicability. The limited research in this area indicates a gap in the body of knowledge and the authors found no publications that address the use of VLS combined with varying production intervals in the food manufacturing sector.

Important work in the time varying lot sizes class was done by Dobson (1987), which led to time varying cycles. However it is questionable whether the cycle term should continue to be used, because of the possible absence of repetitiveness. Moon et al. (2002) have looked at the ELSP with an "imperfect" production process, which also uses a time varying lot size approach. The last two mentioned papers reveal similar characteristics to the work, but these papers give priority to theoretical solutions based on numerous assumptions, whilst the approaches in this paper focus on a practical approach. Time-varying lot-sizes belong into the ELSP class rather than the SELSP category, because although demand varies - these variations are supposed to be known. However, one could put them into the category of stochastic ELSP claiming zero noise in the predicted demand. If noise cannot be explained - or if other random influences are present - stochastic models need to be regarded.

Winands et al. (2011) survey the **stochastic economic lot scheduling problem (SELSP)**. The term stochastic is primarily associated with random demands, (possible) random setup times and (possible) random production times. They classify the approaches according to production sequence, which could be fixed or dynamic.

The fixed production sequences are further divided into those with fixed or dynamic cycle length. The fixed production sequence with fixed cycle length can fulfill the demand on a global scale by dividing the "stochastic" demand accordingly. This class can be associated with the common-cycle class mentioned in the ELSP classification. Bradley and Conway (2003) explain characteristics of cyclic inventory. The cycle time is defined here by the production times, the changeover times and slack time. This cycle is repeated. They found that the average cyclic inventory is directly proportional to changeover times. Cycle length is directly proportional to changeover times. Two common operations errors were identified: (1) interrupting a production run (lot-splitting) to fulfill sudden urgent demand requests; (2) increasing machine utilization. The fixed production sequence with dynamic cycle length adapts the cycle length to meet the stochastic demand without changing the production sequence. For this class Wagner (2004) proposes a local search algorithm. The demand is assumed to be a stationary renewal process, and the schedule minimizes the long-run average costs. Product scheduling cycles may vary in multiples of the base period. The algorithm finds feasible solutions, and its performance was tested on several instances. Other important work in this class include Gallego (1990, 1994); Federgruen and Katalan (1996, 1999).

On occasions it may occur that a product is not produced. In this case the production sequence may reinstate later. However, allowing changes to the production sequence (called dynamic production sequence) is another main category Winands et al. (2011) identified. These changes reflect the dynamic situation that agile manufactures face in altering schedules to meet changing demand (Harrison and Hoek, 2011). Gascon et al. (1994) research falls into this class, as they provide heuristics that find schedules given stochastic demand, multi-items and a single-machine. Zipkin (1986, 1991) considered dynamic production sequences as well. Their demand and production processes have stochastic characteristics and are based on queueing models, which lead to interesting theoretical results. Sox et al. (1999) is another important work in this field.

The above literature review and the Bradley and Conway (2003) paper confirm that almost all literature assumes that stochastic demand is stationary. The work presented here was particularly developed to deal with the non-stationarity of demand to cover the gap currently in the lot-sizing literature. Bradley and Conway (2003) give a simple example that demonstrates some of the adverse effects of non-stationary demand in respect to stock-outs and service levels. In practical situations we need to assume

that demand is non-stationary. The case study in this paper uses such demand patterns. As mentioned above most literature deals with cyclic approaches with fixed production sequences. There is a need for methods and algorithms that create dynamic production schedules. The research in this paper can be associated to the SELSP class dynamic production sequence which reflects the agile environment that food manufacturers operate within, if the demand is specified as non-deterministic. From the ELSP point of view our study and algorithms is situated in the varying lot size and varying lead time category, assuming deterministic "dynamic" demand.

The main contributions and objectives of this paper are:

- to provide a lot-sizing technique that can deal with non-stationary and varying demand;
- to introduce a method for planning agile manufacturing schedules;
- to generate production schedules via novel methods;
- to apply and compare these new methods on the basis of a real world case study;
- to propose an efficient parallel genetic algorithm configuration;
- to provide practitioners with tools to create production schedules.

The remainder of the paper is structured as follows. An illustrative example motivates the general mathematical formulation of the problem (section 2). This problem is solved with the genetic algorithm introduced in section 3. The Genetic Algorithm terminology is linked with those from production. The new factorial and fractional methods are proposed in section 4. These methods will be trialled on an industrial scenario dominated by it's demand data (section 5). In section 6 the results of the case study are presented and linked to operations issues. The paper concludes with a discussion of the results and future directions of research (section 7). The main contribution of this paper is the development of an efficient way for practitioners to create agile production schedules. This is achieved through the generation of production schedules via novel methods which were subsequently tested in a real world situation.

2. Mathematical formulation

In the previous section various inventory models were introduced with a focus on the ELSP. It was found that the formulation of the ELSP, with varying lot size, was closest to the problem we sought to address. This section introduces and defines the problem under investigation. A solution procedure will be proposed in the next section.

2.1. Problem definition

The objective is to find a production schedule for multiple products such that the total cost is minimized. The cost is constructed by the holding, shortage and changeover cost.

Finding the schedule is subject to:

- varying demand (non-stationary);
- a single manufacturing line;
- constant production rate;
- smallest production time window fixed;
- cost are product dependent.

The variability in demand of the products motivates the agility of the manufacturing process. Here, agility means that production changes according to demand. For production only a single manufacturing line is available. We assume that production rates for each product remain the same over the entire manufacturing period. The smallest production time window is set, but the production run period itself adapts in an agile way. The holding and shortage costs are product dependent. The changeover (setup) cost occurs when changing to a different product. We will assume that the changeover time is neglect-able in comparison to the production run time.

Example 2.1 (principal approach). *Our goal is to create a production schedule that satisfies the demand and minimizes the costs. Demand for products one (d_1) and two (d_2) are known between day 1 and*

	Products	1	2
cost	holding	\$2	\$3
	changeover	\$1	\$1
	shortage	\$8	\$6

Table 1. Cost factors per batch.

20, $d_1 = (0\ 1\ 0\ 0\ 3\ 0\ 0\ 0\ 2\ 0\ 1\ 1\ 0\ 0\ 0\ 0\ 0\ 2\ 0\ 0\ )$ *and* $d_2 = (0\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 3\ 0\ 0\ 0\ 0\ 0\ 0\ 2\ 0\ 0\ 1\ )$*. For instance the demand during day 5 for product one is three batches.*

There are three cost factors, which will be considered to minimize the overall cost. The holding, setup and shortage cost factors for the two products are shown in table 1. That means the daily holding cost c_1^h *for one batch of product one is \$2. The changeover cost* c_1^o *to product one is \$1. The shortage cost* c_1^s *per batch is \$8 and represents lost sales.*

The decision variable x_i *is the product batch to be produced on a given day, i.e.* x_i *can take on the values 1 or 2. There are 20 decision variables, i.e.* $x = (x_1, x_2, \ldots, x_{20})$*.*

The aggregated number of produced product batches are $p_{k1}(x) = \sum_{i=1}^{k}[x_i = 1]x_i, p_{k2}(x) = \sum_{i=1}^{k}[x_i = 2]x_i$*.* [1] *The aggregated number of consumed product batches are* $r_{k1} = \sum_{i=1}^{k} d_{i1}, r_{k2} = \sum_{i=1}^{k} d_{i2}$*. That means the inventory (in batches) for product one for day i is* $q_{i1}(x) = p_{i1}(x) - r_{i1}$*. This leads to daily holding costs* $c_{i1}^h(x) = [q_{i1}(x) > 0]q_{i1}(x)c_1^h$ *and the total holding cost* $\hat{c}_1^h(x) = \sum_{i=1}^{n} c_{i1}^h(x)$ *for product one; similar for product two* $\hat{c}_2^h(x)$ *resulting in* $\hat{c}^h(x) = \hat{c}_1^h(x) + \hat{c}_2^h(x)$*. The total shortage costs for product one and two are determined in a similar way* $\hat{c}_j^s(x) = \sum_{i=1}^{n} -[q_{i1}(x) < 0]q_{i1}(x)c_j^s$*, resulting in* $\hat{c}^s(x) = \hat{c}_1^s(x) + \hat{c}_2^s(x)$*. Changing to product one on day i causes a cost of* $[x_i \neq x_{i-1}]c_1^o$*,* $i > 0$*. The total of all setup costs are determined by:* $\hat{c}^o(x) = c_{x_1}^o + \sum_{i=2}^{n}[x_i \neq x_{i-1}]c_{x_i}^o$*.*

Thus the objective function is $f(x) = \hat{c}^h(x) + \hat{c}^s(x) + \hat{c}^o(x)$*. A solution can be found using an evolutionary algorithm such as the genetic algorithm. A standard genetic algorithm finds the solution* $x = [1\ 1\ 1\ 1\ 2\ 2\ 1\ 1\ 2\ 2\ 1\ 1\ 2\ 1\ 1\ 2\ 2\ 1\ 2]$ *with value* $f(x) = 80$*, where x is the production schedule and* $f(x)$ *is the cost of this schedule.*

The above example demonstrated the approach in principle. This will be used to derive a more general approach.

2.2. General aspects

In example 2.1 we looked at daily demand d_1 and d_2 for two products for 20 days in "batch" units. We will abbreviate the basic time period with t_b. So this could represent a week, a day or an hour. Let n denote the *number of production windows*, and m the *number of products*. A *production window* is a time slot (with duration t) during which units of one product are manufactured. If the production window's duration is t_b then we say it has window size 1. A window size of two means that we halve the basic time period. In general the *window size* w_l is defined as the fraction $\frac{1}{l}$ that divides the basic time period. For instance if t_b is 6 days then $w_3 t_b$ leads to a two days production window. w_l is used to refine the basic production window. A *production run* can spread over several adjacent production windows. The quantity produced within the production window is the batch size. The *batch size* b_j is the number of units of one product j worked on in one process step. In this paper the batch size is dependent on the size of the production window. The *supreme* batch size is a multiple of batches that are produced within neighboring production windows of the same product. Note the differences to production time (window includes downtime and changeover time) and cycle time (window is non repetitive and only covers one product). The production rate is product dependent and is obtained by $\frac{b_j}{t}$.

The *demand matrix* $D = (d_{ij})$ is given. Here $d_{ij} \in \mathbb{R}$ is the demand in production window (e.g. week) i for product j. If only one index is used then this addresses a column vector of D, e.g. $d_2 = D_{:2}$ is the demand profile for product two. In case demand is negative ($d_{ij} < 0$) there is no net demand during i. Negative demand will be called returned goods and may reduce the amount to produce, assuming the goods are reusable. In practical situations

[1] Iverson (1962) introduced these brackets for true-or-false statements $[S] := \begin{cases} 1 & \text{if } S \text{ is true;} \\ 0 & \text{otherwise.} \end{cases}$. For instance the Kronecker delta is defined by $\delta_{ij} := [i \stackrel{?}{=} j]$.

given demand may be in aggregated form although it actually occurs in shorter time intervals. For instance weekly demand might be given, but actually stock is depleted on a daily basis. There are two typical scenarios for the planning. The first scenario is that the demand can only be fulfilled in *periodic* (e.g. weekly) batches, due to warehouse and transportation arrangements. The second scenario is that demand could be satisfied "*continuously*" (e.g. on a daily basis). We will call the way the demand is satisfied as *depletion scheme*.

We assume linear production and possible consumption during the production run. The *decision variable* x_i is the product, which is produced during production window i. For instance $x_5 = 2$ means that product two is produced in production window (e.g. week) five, in general $x_i \in \{1, 2, \ldots, m\}$. The case of no production can be implemented by allowing $x_i = 0$ (alternatively a product with zero demand can be used). The *decision vector* is defined as $x = (x_1, x_2, \ldots, x_n)$. The decision variable combined with the product's batch size b_j specifies the produced units in production window i, i.e. $[x_i = j]b_j$. *Initial stock* for product j is abbreviated by s_j. However, if there is no initial stock available then it can be created by "shifting" the demand matrix, i.e. adding zero demand on top of the matrix. This might be suitable if the company would like to create initial stock.

2.3. Mathematical program

We will now develop a mathematical program based on the general aspects mentioned above. The following dimensions, parameters and variables will be needed:

Dimensions and indices:

m	number of products
n	number of production windows
j	product
i	production window
k	aggregated index

Parameters:

t	duration of production window
c_j^h	holding cost (per unit per t)
c_j^s	shortage cost (per unit per t)
c_j^o	changeover cost from any product to product j
b_j	batch size for product j per t
s_j	initial stock of product j
d_{ij}	demand for product j during production window i
q_{ij}	inventory at production window i for product j

Decision variables:

x_i	product to be produced in production window i

In order to obtain the objective function the inventory state and its related costs need to be derived.

The aggregated number of produced units in the first k weeks for product j is:

$$p_{kj}(x) = \sum_{i=0}^{k} [x_i = j] x_i b_j + s_j. \tag{1}$$

We start with the production window $i = 0$, to represent additional initial stock $p_{0:}$ on top of existing inventory s_j; and hereby allow the production of one additional batch for a single product j before units are consumed.

The aggregated number of consumed units is the same as the aggregated positive demand:

$$r_{kj} = \sum_{i=1}^{k} [d_{ij} > 0] d_{ij}. \tag{2}$$

Note if returned goods are reusable then $r_{kj} = \sum_{i=1}^{k} d_{ij}$, which reduces the quantity to produce.

That means the units of *inventory* for product j at production window i is:

$$q_{ij}(x) = p_{ij}(x) - r_{ij}. \tag{3}$$

This motivates the *holding cost* for product j:

$$\hat{c}_j^h(x) = \sum_{i=1}^{n} [q_{ij}(x) > 0] q_{ij}(x) c_j^h, \tag{4}$$

where c_j^h is the holding cost per unit per production window. One may want to choose to add the "initial stock" holding costs.

The total *shortage cost* for product j is determined in a similar way:

$$\hat{c_j}^s(x) = \sum_{i=1}^{n} -[q_{ij}(x) < 0] q_{ij}(x) c_j^s, \tag{5}$$

where c_j^s is the shortage cost per unit per production window. Note that these product quantities cannot be back-ordered.

The changeover (setup) cost is obtained by:

$$\hat{c}_j^o(x) = \sum_{i \in K} c_j^o, \tag{6}$$

with $K = \{i \in \{1, \ldots, n\} \,|\, x_i = j \wedge x_i \neq x_{i-1}\}$ and c_j^o is the changeover cost from any product to product j. $x_i \neq x_{i-1}$ is true if a different product was produced in the previous production window. We assume that the change-over (setup) time is negligible in comparison to the production window duration.

Thus the objective function is:

$$f(x) = \hat{c}^h(x) + \hat{c}^s(x) + \hat{c}^o(x). \tag{7}$$

The agile manufacturing problem can be formulated as the mathematical program:

$$z^* = \min_x f(x) = \min_x \hat{c}^h(x) + \hat{c}^s(x) + \hat{c}^o(x) \tag{8}$$

subject to the set of constraints:

$$1 \leq x_i \leq n, \; x_i \in \mathbb{N} \tag{9}$$

A feasible solution for this mathematical program can be found with the Genetic Algorithm.

3. Genetic Algorithm

The first section will give a brief explanation about the underlying biological concepts. The second section introduces the genetic algorithm (GA), and the third section explains how the GA is used in the context of production schedule creation.

3.1. Background - Biological Evolution

Genetic algorithms are based on the natural selection of the survival of the fittest. The idea of these algorithms was first introduced by Holland (1975) and their practical usage was demonstrated by Goldberg and Holland (1988).

The biological background is shortly discussed. It is assumed that a human being is made out of 10^{14} cells (e.g. the diameter of a red blood cell is 9μm). A *cell* contains a linear DNA string. A *chromosome* is a continuous piece of DNA string. The diameter of a chromosome is between 0,2 and 20μm typically. A chromosome is built up by sequence of linearly ordered genes. A *gene* contains the information about the characteristics and shape of a macro molecule.

Each living being has got a *genotype* and a *phenotype*. The genotype is made up by the chromosomes which identify the "whole" of a being. The phenotype describes the appearance of the individual. A set of individuals make up a population. Individuals survive because they are fit or lucky. But fit individuals are more likely to keep on living and to be selected as parents for the next generation. The selected parents reproduce with their strongest gens. But complete unexpected and unforeseen events happen occasionally and change parts of the genetic information. We call this *mutation* of the gens. A better generation (species) replaces the old one, which is part of the evolution process.

3.2. Algorithm

The principals of a genetic algorithm are shown in algorithm 1.

Algorithm 1 Genetic algorithm - principle.

Input: fitness function f
Output: population P
1: set initial population P
2: **while** generation reasonable **do**
3: $I := S(P, f)$ select fit individuals
4: $N := C(I)$ crossover of individuals
5: $P := M(N)$ mutate
6: **end while**

We begin with deriving an initial population (i.e. set of initial production schedules). This is a set of diverse feasible solutions. Usually individuals (schedules) are chosen/created with an opening procedure. The fittest of these create the next generation, i.e. fitter individuals are more likely to be selected. These individuals represent parents and generate children, which "combine" the characteristics of their parents. This process is called crossover

operation, which are often implemented by merging binary representations of the individuals. Mutation randomly changes some individuals. This completes a generation change over. This procedure repeats itself (i.e. the population evolves) until a certain number of iterations or other stop criteria have been reached.

Genetic algorithms (GA) have previously been used for solving lot-sizing problems. Chang et al. (2006) used a GA for solving a fuzzy economic lot-size scheduling problem. They paid attention to perturbations of demand leading to "fuzziness" in the cost function. In contrast to our approach their search is governed by cycles, which they have to accommodate in the decision variables for the production schedule (individual). Yokoyama and Lewis (2003) developed a GA to optimize the stochastic dynamic production cycling problem. This problem differs from the ELSP by using more than one machine (production line). The representation of the decision variables is similar to the one chosen in this study, with the difference that a machine index was necessary. Furthermore, Yokoyama and Lewis (2003) did not investigate the refinement of production windows. Khouja et al. (1998) created a GA for solving the basic period ELSP. Trials on some of the algorithm settings were done. In general finding appropriate settings for the GA is essential for the solution quality. Most authors state only one configuration, but it is advisable to set up an experimental design and test several factors as described in the following sub-section. Torabi et al. (2006) looked at an extension of the economic lot and delivery scheduling problem (ELDSP) using a hybrid GA to find production sequences and other aspects. The hybridism addresses the issue that GAs are "slow" in the local search by introducing a local improvement method for each child. Similar reasons motivated the "guidance" algorithm proposed in section 4. A GA is one of many possible meta-heuristics to be used for scheduling. Almeder and Mönch (2010) have analyzed various algorithms such as the Ant Colonization Optimization (ACO), Variable Neighborhood Search (VNS) and GA in manufacturing/scheduling context. Their study suggests that the VNS is superior to the ACO and GA. Another study by Raza and Akgunduz (2008) compared heuristics in regards to the ELSP. Moreover the study proposed a Simulated Annealing (SA) algorithm to solve the ELSP. SA outperformed other heuristics such as the hybrid GA and neighborhood heuristics. Their SA converged faster than Tabu Search (TS) having similar solution quality as TS. Gaafar (2006) implemented a GA for the dynamic lot sizing problem with batch ordering, which was compared against the modified Silver-Meal (MSM) heuristic. Their results indicate that the GA outperforms the MSM heuristic in regards to solution quality. From the mentioned literature no general statement about the superiority of a particular heuristic can be derived. Alternative approaches to GA will be discussed in conclusion section on further research. The overall framework of dealing with increased complexity introduced in section 4 allows the GA to be substituted with any heuristic that accepts initial solutions.

This section showed the basic GA and reviewed related literature associated to heuristics in regards to lot-sizing problems.

3.3. GA for production schedule

We will now relate the GA to the production schedule creation. In particular we will describe the GA's settings in more detail. These settings are applied to run a traditional and parallel GA algorithm. In section 2 we have defined the decision variables x_i as the product manufactured during production window i. These decisions variables are the *genes* that make up an *individual* (also called *genome*) x. That means a feasible production schedule x is an individual in GA terminology. The *fitness* of an individual is determined by the value of the objective function ($f(x)$, see eq. 7), which are the accrued holding, setup and shortage costs. A set of feasible production schedules represents the current population (generation).

In general an initial set of production schedules (initial population) are determined with an opening procedure. These schedules should be diverse in x and usually vary in $f(x)$ as well. This is important for creating better production schedules. In our algorithm x is obtained by applying a uniform random distribution on the produced products (genes) fulfilling the constraints of equation (9). It would be interesting to initialize production sched-

ules (genomes) with longer production runs (strings of genes), e.g. that each production schedule produces just one product or mixtures with longer production runs. This should improve the run-time, but might affect the quality of the solution. As mentioned above more complex production schedules are based on previously simpler ones. For instance assume two products are produced over five weeks, e.g. $x = (2, 2, 1, 1, 2)$. This will be transformed into a production schedule (parent) that deals with 3 products and 10 half-weeks by doubling each entry in x, i.e. $x' = (2, 2|2, 2|1, 1|1, 1|2, 2)$.

To control the variation of $f(x)$ we scaled it to a predefined range. In our approach we have used rank fitness scaling, i.e. we ordered $f(x)$ and assigned value 1, 2, ... s. The number of candidate schedules (size of the population) is determined by

$$s = \min\left\{\max\left\{n, 40\right\}, 200)\right\}, \tag{10}$$

where n is the number of decision variables (number of production windows), i.e. $s \in [40, 200]$. For the case study this means that the population is initially 104 (number of weeks in two years), and will increase to 200 for refined production windows. The choice of the population size (number of production schedules to investigate) is a trade off between solution quality and performance. A large population will avoid more local optima and increase the chance of the GA finding a global optimum. However, the performance duration will increase. Several authors have document this population size issue, e.g. Alander (1992), Roeva et al. (2013) and Pasandideh et al. (2011). Based on linear algebra, n base vectors (i.e. initial production schedules, parents) are required to span the entire search space (i.e. all possible production schedules). However, if there are less production schedules, mutation would eventually create missing base vectors. Literature acknowledges a relation between the size of the search space and the population size. Our literature review indicates that there is no conclusive method that determines the population size depending on run time or number of decision variables, and that most authors using the GA have experimentally chosen the population size.

Fit individuals (possible production schedules) are selected for generating the next generation (new production schedules). There are several popular selection methods such as the tournament, roulette and stochastic-uniform method. Single factor experimentation indicated that the tournament method leads to the best production schedules out of the three tested methods. In the deterministic tournament method k "players" (individuals) are chosen randomly and the best one becomes a parent. This is repeated until sufficient parents have been chosen. Additionally we allow migration. That means a fraction m_f of the "best" schedules (fittest parents) replaces the worst schedules (i.e. individuals from a sub-population). Single factor experimentation suggested $m_f = 30\%$, other alternatives tested were 10%, 20% and 40%. These two fractions are the schedules for the next generation. Note that we use a percentage e_c that ensures the "best" schedules (fittest parents) reach the next generation, i.e. they by-passed crossover and mutation. We have set e_c to 5%, other less successful options were 10% and 30%.

Our crossover operation takes scheduled production windows (genes) randomly (uniform) from schedule 1 and merges them with schedule 2. For instance $p_1 = (2, 2, 1, 3, 3)$, $p_2 = (1, 2, 2, 3, 2)$ and the random production windows are $(2, 4, 5)$. This means we merge $(\_, 2, \_, 3, 3)$ into p_2, which gives us $c = (1, 2, 2, 3, 3)$ as the new production schedule. The previously mentioned GA mechanism ensures that better schedules are used. Another crossover operation is to randomly choose the cut index. In the previous example, let the cut index be three, then the created schedule is $c = (2, 2, 1|3, 2)$. We could also cut the schedule in two positions, e.g. two and four resulting in $c = (2, 2|2, 3|3)$. Usually not all schedules of the population are created using the crossover operation, but rather a percentage c_f. Our configuration had $c_f = 80\%$.

After the crossover operation the mutation operation is applied. Initial generations are mutated more than later generations. Individual's genes (i.e. parts of the production schedule) are changed by choosing a random number from the normal distribution with mean 0 and decreasing variance. The variance is given by $\sigma_k^2 = \sigma_{k-1}^2(1 - \frac{3k}{4g})$, where g is the total number of generations and k the current generation.

Our primary interest is in the quality of the solution rather than the speed. However, we have limited

the run time to 10 minutes. Furthermore, the GA stops if the improvement (fitness weighted average) over $s_g = 50$ generations is less than $t_f = 10^{-7}$. The genetic algorithm (in context of the fractional schedule generation) terminated in 64.8% of all cases because there was no improvement in the best fitness value for a period of $s_g = 50$ generations. 35.0% of terminations were due to exceeding the generation limit of 100. The rest was due to exceeding the time limit. Increasing the generation limit to 200 resulted in a 96.4% no improvement termination. The previously discussed parameters have been summarized in table 5.

The GA implementation takes advantage of parallelism. In our case study we used 32 cores and shared memory. Furthermore, we used a multi-start approach, which has been integrated in algorithm 2 proposed in the next section. The number of multiple starts of the parallel GA (pGA) was set to 30. The run-time increased as the production schedule was refined. The decision variable could take on values between 0 and 10, i.e. 10 products (plus no production possible). We observed that on average the run-time to find a pGA solution for a decision vector of size 208 was 65.5 seconds (i.e. on average 2.18 seconds for a single pGA run utilizing 32 cores). This increased to 499.7 seconds for a production schedule with 2080 decision variables (i.e. on average 16.7 seconds for a single pGA run). The pGA for refined production schedules uses initial solutions (i.e rougher schedules) found in prior runs. This approach will be explained in detail in the next section. In this section we have discussed the configuration of the pGA. We found that the pGA algorithm can be used to find the feasible production schedules even without initial solutions. However, the solution quality is not as good as using an iterative schedule creation. That means complexity is added iteratively.

4. Iterative schedule creation

The iterative algorithm will be explained first. Followed by an introduction to the factorial and fractional approach.

4.1. Iterative Algorithm

This study investigates how complex schedules are derived via an iterative algorithm that uses parallel genetic algorithms (pGAs). Initially the authors attempted to use the parallel genetic algorithm directly, i.e. without the guidance of the iterative algorithm. This led to high schedule costs. Our algorithm increases the complexity (i.e. number of production windows and products) step by step. The proposed iterative generation of feasible production schedules is shown in algorithm 2. The input for this algorithm

Algorithm 2 Iterative generation.

Input: demand D, see table 4
Output: production schedules P
1: g finds initial production schedules P
2: **for** window size w in W **do**
3: $B = f(P, w)$ set new initial schedules
4: demand depletion transformation
5: adapt batch size
6: **parfor** $k = 1$ to number of runs **do**
7: $p_k = g(B)$ get schedule via GA
8: **end parfor**
9: **end for**
10: **for** #products = 3 to m **do**
11: find schedule based on #products-1
12: **end for**

is the demand of all products. This demand should be ordered according to the product's importance (see section 5). The importance can be identified by the total volume, holding cost, shortage cost or profit. In the case of volume importance this means:

$$\sum_{i=1}^{n} d_{i1} \geq \sum_{i=1}^{n} d_{i2} \geq \cdots \geq \sum_{i=1}^{n} d_{im},\ m > 1. \qquad (11)$$

In the case of a single product the reader is advised to use techniques mentioned in section 1. As mentioned previously the demand may vary, be non-stationary or be discontinued. A detailed discussion of demand for a case study is given in the data section 5, demonstrating numerous real world issues. Other relevant input values and options such as costs or depletion scheme are specified in the input of the algorithm, which are detailed in table 4.

We will now explain the steps in the algorithm in more detail. [Line: 1] The algorithm begins with finding several production schedules P by using the GA explained in section 3 multiple times. This step

focuses on the first two products only. [Line: 2] The for-loop iterates (hence the name iterative algorithm) through a set of production windows W. The set is dependent on the factorial or fractional approach, which are explained in detail in subsequent subsections. [Line: 3] Setting the initial production schedules (population) is the "heart" for iterative improvements. This paper proposes and investigates the factorial and fractional approach. In the algorithm these two initial schedule generation methods are abbreviated with $f(P, w)$, where P are existing production schedules and w is the new window size. f derives a schedule basis B such as $\left\{x_1^{32}, x_2^{32}, \ldots, x_s^{32}\right\}$. Here, s is the number of runs defined in line 6.

[Line: 4] In section 2.2 two depletion schemes: periodic and continuous were introduced. The given demand has to be accordingly transformed. In the periodic depletion scheme the demand is requested at the end of a given period. As an example assume a 12 days schedule with demand aggregated over three days is 12, 3, 9 and 6. The end of period depletion on a daily basis is $(0, 0, 12, 0, 0, 3, 0, 0, 9, 0, 0, 6)$. If we apply the continuous depletion scheme the average depletion is $(4, 4, 4, 1, 1, 1, 3, 3, 3, 2, 2, 2)$. In the case study periodic depletion was used.

[Line: 5] In order to fulfill all demand on average the production rate must be $p_{avg} = \frac{1}{n}\sum_{i,j} d_{ij}$. As mentioned in result section 6 it is recommended that the actually used planning production rate is at least twice p_{avg}. The permission of production windows with zero output can be interpreted as adjusting the production rate according to demand. Here, we set the batch size to:

$$b = \frac{2}{n}\sum_{j=1}^{m}\sum_{i=1}^{n}[d_{ij} > 0]d_{ij}. \tag{12}$$

Note, that the batch size depends on the number n of production windows, e.g. n increases when the window size increases. It is also possible to set batch sizes for "individual" products: $b_j = \frac{2}{n}\sum_{i=1}^{n}[d_{ij} > 0]d_{ij}$.

[Line: 6 to 8] The iterative algorithm starts the GA multiple times. The GA was discussed in section 3 and returns a single schedule (individual) p_k via $g(w, B)$, since the GA includes some random operations the returned solutions are expected to vary. Thus, we are obtaining several production schedules $\{p_1, p_2, \ldots, p_s\}$, where s represents the number of runs. These schedules are used as part of the initial population (line 3) for finding a refined production schedule (e.g. windows of size 6 for two products). Later y_k^{w2} will be used to describe $p_k \in P$ with production window size w ($w > 1$) for two products. In the special case that P consists only out of one element we will drop the index k. So $f(y^{32}, 6)$ would have created x^{62} using the factorial design, which $g(x^{62})$ turned into y^{62}.

[Line: 10 to 12] These lines add additional products one at a time, i.e. the values a decision variable can take on are increase incrementally. Again the GA is used to achieve this. This leads to a growing product variety as shown in figure 2.

Subsection 4.2 and 4.3 will explain in detail how fractional and factorial schedules are created.

4.2. Factorial Generation

The basic principle is that a "rough" schedule is refined, i.e. a top down approach where the number of products is fixed (see figure 2). For instance an initial schedule may consist of fixed weekly (6 days) production windows. This duration is linked to window size w_1. The first refinement is half-a-week (3 days and window size w_2). Followed by a third-week (2 days and w_3), and so on. The half-a-week ("division-by-two") schedule uses the one-week slots as initial solutions. The same applies for a division-by 3, 5 and 7 schedule. In general any division-by a prime number schedule can be derived from the schedule with window size 1. The other schedules are derived from the highest factor, e.g. a divided-by 4 schedule is derived from 2, 6(= $3 \cdot 2$) from 3, 8 from 4, 9 from 3 and so on. We will call this the *factorial approach*. The factorial approach is shown in figure 2. Here we see $(1, 2)$ representing the schedule with window size 1 and two products, which can be derived via the parallel genetic algorithm proposed in section 3. This is the basis to derive (2,2), (3,2) or (ω,2), where ω is a prime number. In general the schedule $y^{\omega 2}$ is created via $g(x^{12})$, where g the genetic algorithm. x^{12} is obtained via $f(y^{02}, 1)$. Here, f is the factorial generation function and y^{02} is an initial schedule (see alg. 2, line 1). g and f are also shown in algorithm 2 in line 7 and 3 respectively.

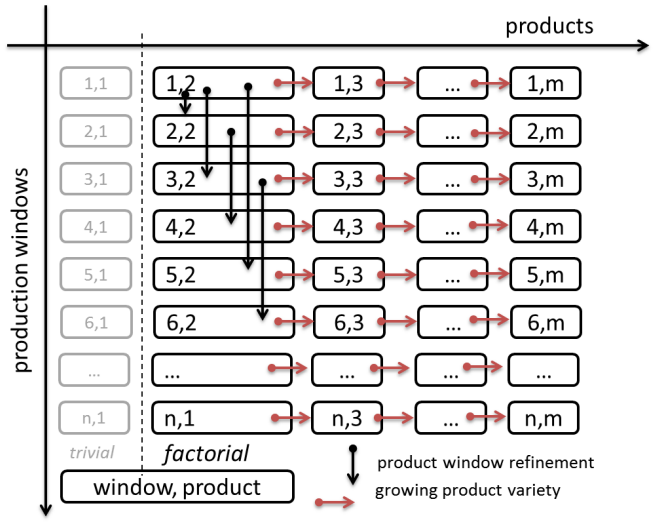

Fig. 2. Factorial solution generation.

Let $y^{12} = (2, 1, 1, 2, 2)$ be a (1,2)-schedule. This is transformed into $x^{22} = f(y^{12}, 2) = (2, 2; 1, 1; 2, 2; 2, 2; 2, 2)$ the (2,2) schedule basis. Similarly into the (3,2) schedule $x^{32} = f(y^{12}, 3) = (2, 2, 2; 1, 1, 1; 2, 2, 2; 2, 2, 2; 2, 2, 2)$. x^{22} and x^{32} are initial schedules, which are improved by the GA to obtain y^{22} and y^{32}, e.g. assume y^{22} evolved into $(2, 2; 2, 1; 1, 1; 1, 2; 2, 2)$. From figure 2 it can be seen that y^{22} is the (2,2) schedule basis that obtains the initial production schedule for the (4,2) schedule via f. For the example that means $x^{42} = f(y^{22}, 4) = (2, 2, 2, 2; 2, 2, 1, 1; 1, 1, 1, 1; 1, 1, 2, 2; 2, 2, 2, 2)$. Now the y^{42} is obtained via $g(x^{42})$. Schedule basis (4,2) will be used to derive schedule y^{43}. That means the number of production windows remains four, but a third product is introduced via g. Furthermore f will not be used because of the "horizontal" direction (see 2). This gives us $y^{43} = g(y^{42}, 3) = (2, 2, 2, 3; 3, 3, 3, 1; 1, 1, 1, 1; 1, 1, 1, 3; 3, 3, 2, 2)$. The second parameter of g specifies the number of used products (see also alg. 2 line 11). We see that g has replaced some of the previously set products with product 3. This iterative approach is based on the highest combined factor α obtained so far. This guarantees that the obtained schedule is better than the α schedule. In general $y^{k\alpha,2} = g(f(y^{\alpha,2}, k))$; and $y^{\alpha,j} = g(y^{\alpha,j-1}, j)$ for $j > 2$. A higher product dimension schedule $y^{\omega 3}$ or $y^{k\alpha,3}$ is derived via g from $x^{\omega 3} = y^{\omega 2}$ or $x^{k\alpha,3} = y^{k\alpha,2}$ respectively. This is used to derive the production window set W specified in algorithm 2. For instance if the required production window is $t = 16$ then $W = \{1, 2, 4, 8, 16\}$. In the next section a second approach is discussed. Section 6 compares the production schedule costs of these methods.

4.3. Fractional Generation

An alternative approach is a fractional generation, this means a production schedule is partially fitted into the refined schedule. For instance a slot 2 schedule with 3 products $y^{23} = (2, 3|2, 2|1, 1|1, 3|2, 2)$ is transformed into a "refined" slot 3 schedule by leaving a production gap $x^{33} = (2, 3, 0|2, 2, 0|1, 1, 0|1, 3, 0|2, 2, 0)$. The gap is defined by setting every third element to zero. Such refined schedules are used in the parallel genetic algorithm as initial solutions. In general the fractional generation process for two products is: $y^{02} \overset{g\circ f}{\rightarrow} y^{12} \overset{g\circ f}{\rightarrow} y^{22} \dots \overset{g\circ f}{\rightarrow} y^{n2}$ or $y^{n2} = (g \circ f)^n(x^{12})$, where g is the genetic algorithm or any production schedule generation function, f is the fractional generation function and y^{02} an initial solution (see alg. 2). This process shows that the set of production windows for algorithm 2 is $W = \{1, 2, \dots, t\}$.

The fractional generation function currently uses production gaps. However, instead of non-production fractions a heuristic interpolation procedure can be used. If product A is produced in the preceding and succeeding production window it can be assumed that product A will be produced during that time slot as well. If two different products surround the gap then the one with the greatest missing demand should be produced. These heuristics shall be investigated in future work.

This section proposed an algorithm and methods to iteratively create a production schedule with higher complexity.

5. Case Study Data

The case study company observed that their inventory levels are too high and detached from the actual demand. So, a change from a push strategy to a pull strategy became of interest. That means that manufacturing has to be done in an agile way. The previous sections introduced a methodology that derives a

schedule for agile production. This section will introduce the company process, the demand data and suggest a cost upper bound.

5.1. Company and Process Overview

The case study company produces a range of 250 food products that service a heterogeneous customer base. The SME had actively pursued growth through expanding its customer base driving a change in manufacturing variables including lead-times and order size. The new demand profile had adversely impacted on the performance of the operations area. The production process that is deployed utilizes a five stage manufacturing route. Raw material inputs are cleaned and sorted leading to the production of eight intermediary semi-finished products which are stored in material handling silos awaiting release to the appropriate production lines for packaging before shipment to the warehouse. The manufacturing process had been faced with the challenges of growing sales, increased inventory cost and falling service levels. Increasing demands on the manufacturing facility challenged the historical approach to production planning and control. The increasingly variable demand on the manufacturing operation weakened the reliability of the planning schedule rendering the plan obsolete within hours of issue. The four week time horizon schedule began to reflect a customer order list. The reactive planning and control modus operandi had forced manufacturing into an increasing number of unplanned changeovers with diminishing performance. The firm had effectively moved from a position of flexible spare capacity to a bottleneck over a three year period due to increased complexity and reduction in schedule stability.

Improving the reliability and responsiveness of the production schedule became the focus of the organization. In order to reduce the costs of unplanned changeovers, lost production capacity, increased downtime and emergency customer shipments the case study firm investigated the possibility of creating focused factories. Products were relocated to specific production lines based on pack size, the primary driver of changeover time, and classified in terms of Make-To-Order (MTO) and Make-To-Stock (MTS) status to develop a more supportive inventory position. These refinements and changes were expected to improve the productivity of the line and the stability of the schedule.

5.2. Demand profiles

In this study we focus on an arbitrary selection of ten products (out of 250). For each product the demand is known on a weekly basis. The time-series of the individual products are shown in figure 3. We see that the quantities of the first two products dominate. In order to optimize the production process greater understanding of the individual product's demand profile has to be gained. We propose to first classify the products:

1. any products revealing a "high" coefficient of variation are flagged red (see "traffic light" in figure 3 and first letter in product code was set to R) - indicating MTO production;
2. time-series with a single significant innovation get an orange traffic-light assigned - requiring the time series to be divided into two sections; an algorithm in Garn and Aitken (2015) shows how the innovation split can be found;
3. all other products are marked as green - suggesting demand profiles that typically include MTS and MTO components.

The orange and green flagged time series usually have a hybrid MTS/MTO demand profile. The Garn and Aitken (2015) splitting method delivers the individual Make-To-Stock (MTS) and Make-To-Order(MTO) time-series. A product code was derived as follows: traffic light (G= green, O= orange, R= red); followed by a sequential number (01, 02, ..., 10); Savings (S= significant, M= moderate, T= tiny, L= low); and back-order information (N= none, L= low, M= moderate, H= high). There is a necessity to improve the scheduling of the production process. In this example we have a variety of time-series and most products have a time-series that can be split into two components.

Let us now return to the total demand. The overall aggregated demand reveals an increasing linear trend (see figure 4). This means we are dealing with a non-stationary production planning process. Furthermore, there is a visible innovation after the first year, which can be obtained using the innovation identification algorithm (Garn and Aitken, 2015). This sug-

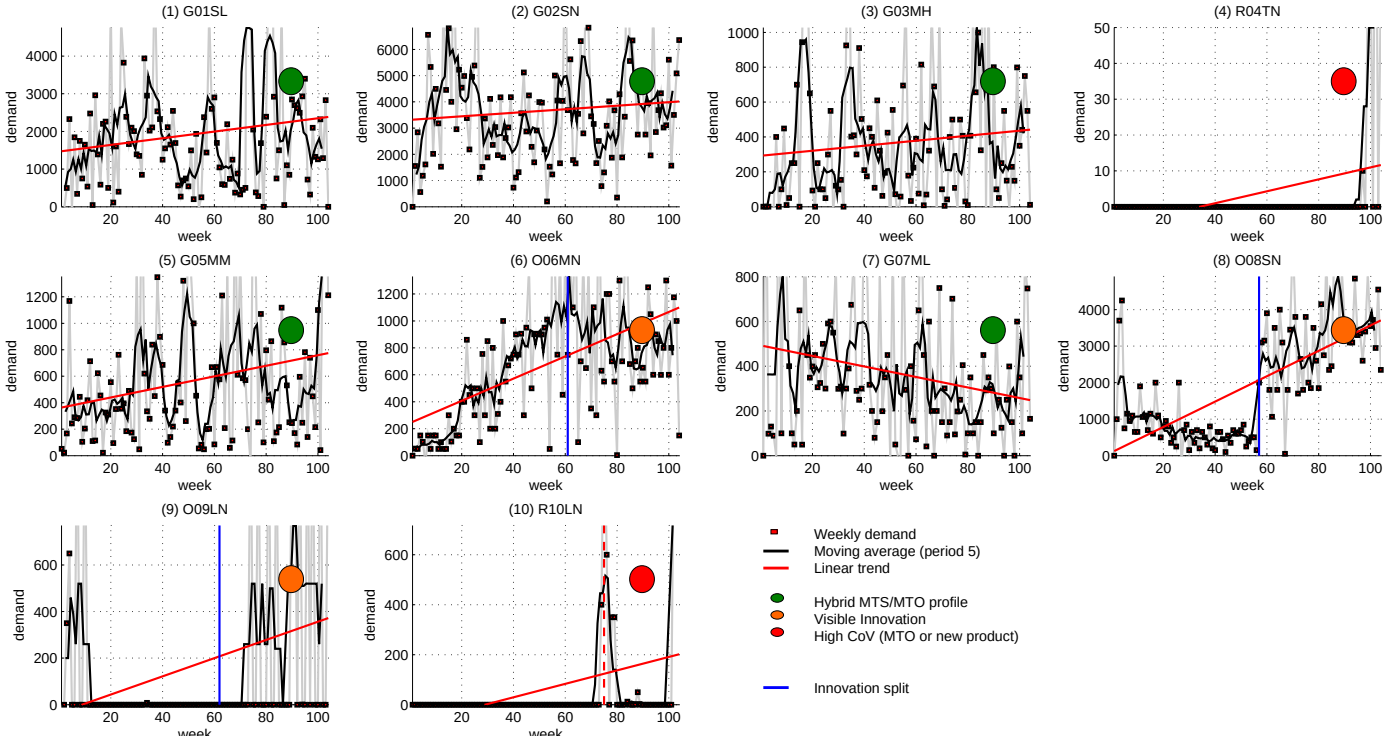

Fig. 3. Time series showing the weekly demand and additional information.

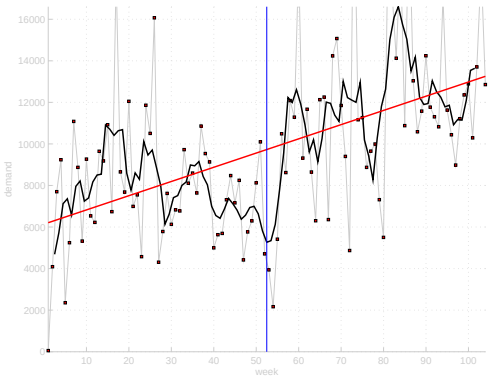

Fig. 4. Total demand per week.

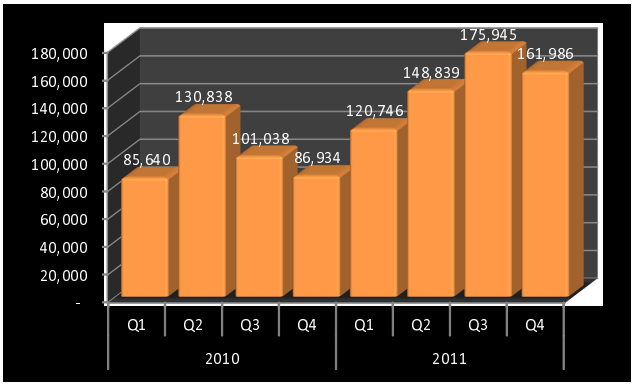

Fig. 5. Aggregated plan.

gests that planning for each year should be done separately. The weekly demand has high variability, indicating cost factors caused by staff scheduling constraints.

5.3. Demand overview and plans

The rough capacity plan was to produce 450k packages in the year 2010. The actual demand in 2010 was 404k packages, which left an excess inventory of 46k packages - 10.1% below plan.

The assumption for 2011 was a growth of 20%, due to sales agreements. The plan was to have 404k plus 20% packages in 2011. That meant the production plan intended to produce 440k packages taking into account the excess inventory from 2010. However, the actual demand in 2011 was 608k packages - this would have meant lost sales of 30%.

Let us consider the demand per quarter (figure 5). We observe significant changes between the quarters (see table 2). The absolute average change of consecutive quarters is 27,439 packages, this is a relative average change of 21.7%. The planned production was 112,500 packages, which was insufficient for 2011 and a capacity management decision was nec-

Table 2. Quarter changes.

	Quarter	Change
2010	Q1	
	Q2	52.8%
	Q3	-22.8%
	Q4	-14.0%
2011	Q1	38.9%
	Q2	23.3%
	Q3	18.2%
	Q4	-7.9%

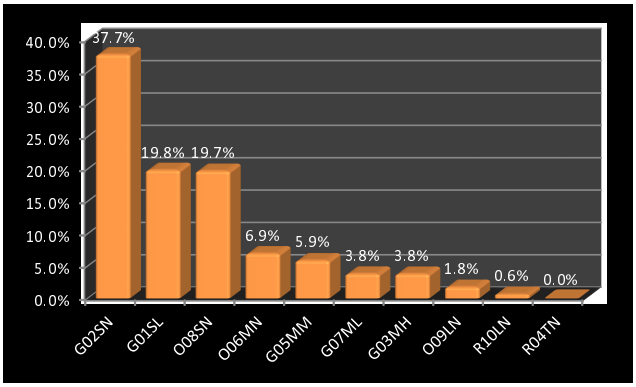

Fig. 6. Product volume importance.

essary. An investment doubled the production rate in the second quarter of 2011.

For the scheduling it is interesting to know which products have the highest demand. Figure 6 shows that product G02SN is causing 37.7% of all demand. Furthermore three products account for 77.2% of all demand. The product R04TN is from a volume and cost perspective not important. A comparison of setup cost against shortage cost (lost sales) shows that this product causes losses, in particular since the demand is at the end of the period. So, practically it should not be manufactured. The generated production schedules (see next section) support this and do not produce R04TN. Thus, reducing product variety and improving factory focus.

5.4. Cost upper bound

Finding an upper bound for the costs is achieved with the following considerations. Holding cost and shortage cost cannot occur concurrently. Setup costs can be avoided by not producing anything at all. That means the maximal cost factors are:

$$c_j^{\max} = [c_j^s \geq c_j^h]c_j^s + [c_j^s < c_j^h]c_j^h, \qquad (13)$$

where j is the product, c_j^s and c_j^h are the product's shortage and holding costs respectively. Usually shortage cost is higher than holding cost, simplifying the cost bound to be the shortage cost. That means an upper bound for the cost is:

$$m = \sum_{i,j} |d_{ij}| c_j^{\max}. \qquad (14)$$

In the case study the maximal cost is \$7,543,580 over two years. The individual cost factors and total demand volume are shown in table 3. The holding cost and shortage cost are in dollar per unit per week. The setup cost occurs when changing from a product to the specified one.

Furthermore, product 2 can cause 38.4% (\$2.89M) damage and product 8, 21.4% (\$1.61M). Note that shortage cost can sometimes be interpreted as negative revenue, i.e. lost sales or possible revenue. So product 2 and 8 could generate 59.8% (\$4.50M) of all possible revenues.

In table 5 we have summarize the data requirements to run the iterative generation algorithm for the case study.

6. Results and Discussion

In section 4 we have proposed an iterative algorithm that generates production schedules. The demand data fed into this algorithm was discussed in section 5. Figure 7 illustrates the factorial and fractional schedule results. The bold brown line represents the average production schedule cost obtained by the factorial approach. The average was formed by calling the iterative algorithm 30 times. Individual results are shown as red dots and their sample standard deviation is indicated by thin brown lines. The bold blue line displays the production schedule cost derived through the fractional approach. It can be seen that the factorial generation outperforms the fractional approach. This is first observable when two production windows occur per week. That means that the half-week gaps cannot be filled with the fractional approach well enough. It would be interesting to start the fractional approach from the factorial w_2 solution onwards. The factorial and fractional approaches reveal challenges of finding better solutions due to the increased number of decision variables.

Table 3. Cost factors and volume.

cost	G01SL	G02SN	G03MH	R04TN	G05MM	O06MN	G07ML	O08SN	O09LN	R10LN
holding	1.3	2.4	1.5	2.2	1.1	1.6	2.3	3	2.3	2.2
setup	5,600	7,900	5,600	6,500	7,000	7,100	5,600	7,700	5,200	8,300
shortage	7.2	7.6	6.6	8.1	4.6	8.6	7.3	8.1	6.8	8.3
volume	200,626	381,381	38,241	320	59,228	70,167	38,429	199,239	17,759	6,576
volume [%]	19.8%	37.7%	3.8%	0.0%	5.9%	6.9%	3.8%	19.7%	1.8%	0.6%

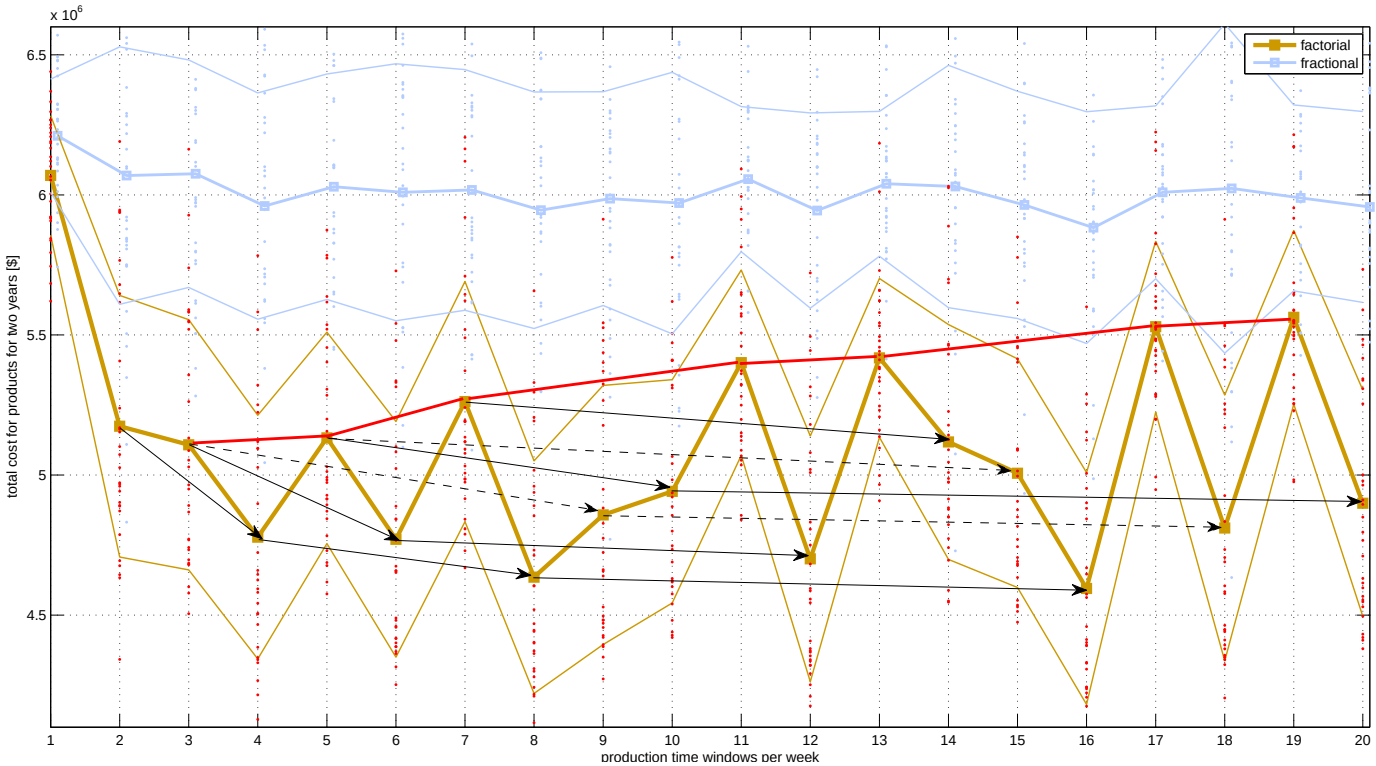

Fig. 7. Production schedules created via factorial and fractional approach.

The factorial solution generation process reveals the following patterns. Two production windows per week (i.e. window size is 2 this is abbreviated with w_2) are based on a schedule with w_1. The production schedule w_4 is derived from w_2 and shows improvement. $w_8 \leftarrow w_4$ and w_{16} follow the iterative improvement process. It can be observed, that the iterative factorial algorithm improves with each refinement of the production window as expected. The results suggest that the cost savings follow a negative exponential function that means initial production window refinements lead to higher savings. If we turn to three production windows per week w_3, an improvement over w_1 is observed however, it is not better than w_2. This is due to increased complexity. That means it is harder for the iterative factorial algorithm to find a better solution due to dimensionality increase. Continuing with w_3 based schedules such as w_6 and w_9 show steady improvements. It can be observed, that the iterative factorial algorithm improves with each refinement of the production window as expected. In general we can observe prime factor schedule cost (fig. 7, red line) increase due to growing complexity. Overall factor derived based schedules improve iteratively. The factorial approach is based on the idea that the best refinement can be obtained from a factorization that uses the largest found schedule as an initial solution. This works well for w_2 However, it can be seen that a better solution could have been found for w_6 by using w_2 instead of w_3. This is a surprising result and indicates that the increased number of decision variables has a major impact on the solution quality.

The results found motivate an improved iterative refinement approach (see figure 8), which makes use of factorial and fractional approach to obtain higher cost savings.

A couple of additional interesting observations were made. Preventing production gaps has a major effect on the costs, e.g. product 2 without production gaps leads to 23.5% higher costs on average. Changing the batch size (production rate) from the average demand to twice of the average demand improves the solution by 39.3% for the fractional approach. This observation was the reason to set batch size (in section 4) to $2p_{avg}$. This behavior is visualized in figure 9. Here the batch size factor is a multiple of the average demand. We have plotted four production windows (abbreviated with pw). This motivates the following questions: What is the optimal batch size (if only one batch size is possible)? Figure 9 indicates that 1.5 times the average demand leads to the minimum. Is a "dynamic (unlimited) batch size" the most cost efficient way to produce? Contrary to intuition we observe that costs start to increase with increased batch sizes.

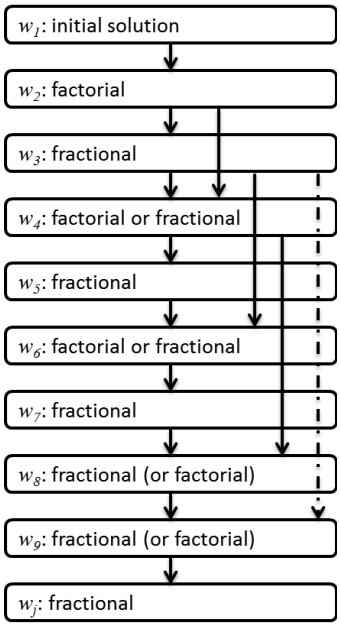

Fig. 8. Improved iterative refinement process

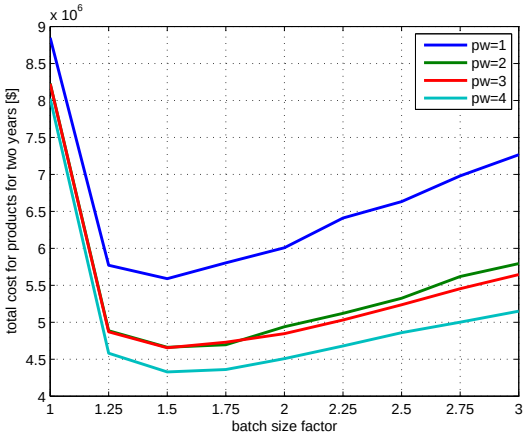

Fig. 9. Cost depends on batch size factor.

7. Conclusion & Managerial Implications

The case study's demand was non-stationary and required a dynamic schedule breaking away from

the main-stream "cycle methods". These demand variations of multiple products motivated agile manufacturing. To find an agile production schedule for the single manufacturing line, we have developed an iterative algorithm that guides parallel genetic algorithms. This approach was necessary to alleviate the effects of the "curse" of dimensionality that affected the solution quality. Here, the dimensions were the production windows and products. We have devised the factorial and fractional generation methods, which refine production windows systematically. The case study's product volumes revealed a Pareto distribution pattern. Therefore important products (e.g. based on aggregated demand) are used first to find schedules. The iterative algorithm increases the number of products incrementally, which are fed into the genetic algorithm We experienced that the pGA's configuration is essential for the solution quality despite of the iterative algorithm's control. The analysis has shown that the factorial approach finds the most cost efficient schedules, in comparison to the fractional approach. The results confirm intuition about complexity, such as higher prime number refined schedules led to higher inventory and production costs. We have also observed that significant cost savings are possible with the fractional approach by increasing the number of production windows initially. However, the saving opportunities through refined production windows decrease exponentially. Another interesting insight for operations managers is that utilizing machines' capacity (or maximizing the produced batch during a time window) does not lead to a cost reduction as shown in section 6. Overall it can be concluded that the number of products and time slots greatly affect the quality of schedules. The case study supports that agile manufacturing, i.e. adapting production according to demand, is a cost efficient approach.

The study gives rise to a number of avenues for further research. Following is a discussion of three possible avenues. The first avenue is in regards to stochastic demand. In practice the assumption that all demand for each week is known with certainty cannot uphold. Thus it is necessary to investigate the stochastic parts of the demand. Future research can use the formulations and algorithms presented here as basis for gaining insight into production planning time horizons. This can be achieved by using additional forecasting methods. The introduced case study can be used by defining fractions of the time series' demand as uncertain. The second avenue is to improve the model. This should cover the topology of the plant Section 5 introduced the production process with several stages. This can be mapped using simulation models. Furthermore the perishable nature of the products and limited storage should be taken into account. A third possible avenue is to look at different algorithmic approaches. Instead of the parallel genetic algorithm other meta heuristics such as SA and TS can be used to find production schedules. The iterative algorithm lends itself to accommodate other heuristics as long as they accept initial solutions. This algorithm in conjunction with the factorial and fractional method is general enough to be applied to other problem domains, which "struggle" with complexity. Using Mixed Integer Programming might be another option although the number of decision variables will constitute a challenge to practitioners. As an alternative MIP could be embedded in the iterative algorithm to solve parts of the problem exactly. An interesting approach might be to identify the demand profile as an "analog" signal and the production schedule as a discretization of this signal. That means the Fast-Fourier-Transformation algorithm can be applied. Future work should also include the development of a software (preferably with GUI) that can be readily deployed in production environments.

The dynamic nature of demand encourages agile manufacturing in many industries. Agile lot sizing challenges are therefore an interesting area for further research. The concepts and methods developed in this paper opens up the way for such analyses.

Appendix

Table 4 lists the required input data to run the iterative generation algorithm.

Table 5 summarizes the settings discussed in section 3.3.

References

Alander, J., 1992. On optimal population size of genetic algorithms. In: CompEuro 1992 Proceedings Computer Systems

Table 4. Data required to run the iterative generation algorithm.

input	unit	description	case study settings	see section
time slot t	[week,day,etc.]	duration the demand is specified in	week	2.2, 5.2
demand	[unit/t]	for each product i and time slot j	weekly data over two years	5.2, 5.3
depletion scheme	-	periodic or continuously	periodic	2.1, 4.1
holding cost	[$/unit/t]	cost of stocking a unit of product j for period t	10 products (dollar per unit per week)	2.2, 5.3
setup cost	[$/unit/$t$]	cost of changing over to product j	10 products (dollar per unit per week)	2.2, 5.3
shortage cost	[$/unit/t]	lost sales for product j during t	10 products (dollar per unit per week)	2.2, 5.3
batch size	[unit/t]	for each product j per t	same for all products, twice of the average demand	2.2, 4.1, 6
window refinement	-	desired number of production windows per t	1 to 20	6
design	-	factorial or fractional	both	4.2, 4.3, 6

Table 5. Settings for the Genetic Algorithm and tested alternatives.

symb.	parameter	chosen setting	tested alternatives
-	opening procedure	uniform random procedure	-
-	fitness scaling function	rank	proportional, top
s	population size	200	100
-	selection function	tournament	stochastic uniform, roulette
-	migration direction	forward	both
m_f	migration fraction	30%	10%, 20%, 40%
e_c	elite count	5%	10%, 30%
-	crossover function	scattered	single point, two points
c_f	crossover fraction	80%	70%, 90%
-	mutation function	normal distribution	-
-	time limit	600 seconds	20s, 3600s
-	generations without improvement	50	-
-	tolerance function	1.00E-06	-
-	max. generation iterations	200	100
-	fitness lower limit	0	-

and Software Engineering. IEEE Comput. Soc. Press, pp. 65–70.

Almeder, C., Mönch, L., Dec. 2010. Metaheuristics for scheduling jobs with incompatible families on parallel batching machines. Journal of the Operational Research Society 62 (12), 2083–2096.

Bradley, J. R., Conway, R. W., 2003. Managing cyclic inventories. Production & Operations Management 12 (4), 464 – 479.

Chan, H. K., Chung, S. H., Lim, M. K., 2013. Recent research trend of economic-lot scheduling problems. Journal of Manufacturing Technology Management 24 (3), 465 – 482.

Chang, P.-T., Yao, M.-J., Huang, S.-F., Chen, C.-T., 2006. A genetic algorithm for solving a fuzzy economic lot-size scheduling problem. International Journal of Production Economics 102 (2), 265 – 288.

Christopher, M., 2011. Logistics and supply chain management. Pearson Education, Harlow.

Dobson, G., 1987. The economic lot-scheduling problem: Achieving feasibility using time-varying lot sizes. Operations Research 35 (5), 764.

Erlenkotter, D., 1990. Ford whitman harris and the economic order quantity model. Operations Research 38 (6), 937 – 946.

Federgruen, A., Katalan, Z., 1996. The stochastic economic lot scheduling problem: Cyclical base-stock policies with idle times. Management Science 42 (6), 783 – 796.

Federgruen, A., Katalan, Z., 1999. The impact of adding a make-to-order item to a make-to-stock production system. Management Science 45 (7), 980 – 994.

Gaafar, L., Nov. 2006. Applying genetic algorithms to dynamic lot sizing with batch ordering. Computers & Industrial Engineering 51 (3), 433–444.

Gallego, G., 1990. Scheduling the production of several items with random demands in a single facility. Management Science 36 (12), 1579 – 1592.

Gallego, G., 1994. When is a base stock policy optimal in recovering disrupted cyclic schedules? Naval Research Logistics (NRL) 41 (3), 317–333.

Garn, W., Aitken, J., Apr. 2015. Splitting hybrid Make-To-Order and Make-To-Stock demand profiles. arXiv preprint arXiv:1504.03594, 15.

Gascon, A., Leachman, R., Lefranois, P., 1994. Multi-item, single-machine scheduling problem with stochastic demands: a comparison of heuristics. International Journal of Production Research 32 (3), 583.

Goldberg, D. E., Holland, J. H., 1988. Genetic algorithms and machine learning. Machine learning 3 (2), 95–99.

Harris, F. W., 1913. How many parts to make at once. Factory, The Magazine of Management 10, 135–136, 151.

Harris, R., Sollis, R., 2005. Applied Time Series Modelling and Forecasting, paperback Edition. John Wiley and Son.

Harrison, A., Hoek, R. I. v., 2011. Logistics management and strategy: competing through the supply chain. Pearson/Financial Times Prentice Hall, Harlow, England; New York, NY.

Holland, J., 1975. Adaptation in natural and artificial systems. University of Michigan Press.

Hsu, W.-L., 1983. On the general feasibility test of scheduling lot sizes for several products on one machine. Management Science 29 (1), 93.

Iverson, K. E., 1962. A Programming Language. John Wiley & Sons.

Khouja, M., Michalewicz, Z., Wilmot, M., Nov. 1998. The use of genetic algorithms to solve the economic lot size scheduling problem. European Journal of Operational Research 110 (3), 509–524.

Moon, I., Giri, B. C., Choi, K., 2002. Economic lot scheduling problem with imperfect production processes and setup times. The Journal of the Operational Research Society 53 (6), pp. 620–629.

Osman, H., Demirli, K., 2012. Economic lot and delivery scheduling problem for multi-stage supply chains. International Journal of Production Economics 136 (2), 275 – 286.

Pasandideh, S. H. R., Niaki, S. T. A., Nia, A. R., Mar. 2011. A genetic algorithm for vendor managed inventory control system of multi-product multi-constraint economic order quantity model. Expert Systems with Applications 38 (3), 2708–2716.

Raza, A. S., Akgunduz, A., Aug. 2008. A comparative study of heuristic algorithms on Economic Lot Scheduling Problem. Computers & Industrial Engineering 55 (1), 94–109.

Roeva, O., Fidanova, S., Paprzycki, M., Sept. 2013. Influence of the population size on the genetic algorithm performance in case of cultivation process modelling. Computer Science and Information Systems (FedCSIS), 2013 Federated Conference on, 371–376.

Rogers, J., 1958. A computational approach to the economic lot scheduling problem. Management Science 4 (3), 264 – 291.

Silver, E. A., Pyke, D. F., Peterson, R., 1998. Inventory Management and Production Planning and Scheduling, 3rd Edition. Wiley, New York.

Sox, C. R., Jackson, P. L., Bowman, A., Muckstadt, J. A., 1999. A review of the stochastic lot scheduling problem. International Journal of Production Economics 62 (3), 181 – 200.

Squire, B., Cousins, P. D., Lawson, B., Brown, S., Jul. 2009. The effect of supplier manufacturing capabilities on buyer responsiveness: The role of collaboration. International Journal of Operations & Production Management 29 (8), 766–788.

Taylor, D. H., Fearne, A., Sep. 2006. Towards a framework for improvement in the management of demand in agri-food supply chains. Supply Chain Management: An International Journal 11 (5), 379–384.

Torabi, S., Fatemi Ghomi, S., Karimi, B., Aug. 2006. A hybrid genetic algorithm for the finite horizon economic lot and delivery scheduling in supply chains. European Journal of Operational Research 173 (1), 173–189.

Wagner, H. M., 2004. Comments on "dynamic version of the economic lot size model". Management Science 50, 1775 – 1777.

Winands, E., Adan, I., van Houtum, G., 2011. The stochastic

economic lot scheduling problem: A survey. European Journal of Operational Research 210 (1), 1 – 9.

Yokoyama, M., Lewis, H. W., 2003. Optimization of the stochastic dynamic production cycling problem by a genetic algorithm. Computers & Operations Research 30 (12), 1831 – 1849.

Zipkin, P. H., 1986. Models for design and control of stochastic, multi-item batch production systems. Operations Research 34 (1), 91 – 104.

Zipkin, P. H., 1991. Computing optimal lot sizes in the economic lot scheduling problem. Operations Research 39 (1), 56.