Learning from Nature: Using Genetic Algorithms for Inventory Optimisation

Thesis

In Fulfillment of the Requirements for the Degree of

'Bachelor of Science'

in Global Business Management

at the Faculty of Business and Economics of the University of Augsburg

Chair of Quantitative Methods in Business and Economics

Submitted to:	Prof. Dr. Michael Krapp
Supervisor:	Deniz Preil (M.Sc. mult.)
Student:	Leopold Pfeiffer
Matriculation No.:	[redacted]
Adress:	[redacted]
	[redacted]
E-mail:	[redacted]

Augsburg, June 2020

Contents

Abstract

A battery of approaches has been applied by researchers and practitioners in the field of inventory optimisation to find optimal inventory policies that can drive the success of businesses of various industries. One such approach is based on the use of genetic algorithms, a multi-purpose subclass of evolutionary algorithms that imitate the principles of evolution to solve combinatorial problems. In this thesis, we extensively explore the theoretical background of inventory optimisation as well as genetic algorithms before we develop a four-stage serial supply chain model and implement a genetic algorithm for base-stock level optimisation.

1 Introduction

Inventory typically makes up 20% to 60% of total assets on the balance sheet of manufacturing companies making well-calibrated management of this inventory indispensable for the success of these businesses (see Arnold et al., 2008, p.254). At its core, inventory management faces a trade-off between customer service, i.e. lead times that should be as short as possible, and inventory cost constituted of holding and order costs (see Schmidt and Wilhelm, 2000, p.3). For the former, inventory management serves as a hedge against uncertainty as it attempts to enable the firm to satisfy customer demand even under unforeseen circumstances, whereas for the latter, the goal is to keep costs as low as possible. While inventory management can be limited to the scope of a single business, research often puts a special focus on entire supply chains (SC), 'network[s] of organi[s]ations that are involved, through upstream and downstream linkages, in the different processes and activities that produce value in the form of products and services in the hands of the ultimate consumer.' (Christopher, 2005, p.17). It is this two-way relationship between the SC agents that is paramount for inventory control as product and information flows between agents crucially influence what an optimal inventory policy looks like. In an attempt to master this problem, a variety of inventory optimisation approaches have been developed, which range from exact to heuristic or approximate solutions. A widespread issue with exact solutions is that the underlying models are exceedingly expensive from a computational point of view and are therefore often not feasible in practice (see Clark, 1960, p.429). Furthermore, the stochastic nature of many real-world applications makes it very difficult to find exact solutions in the first place. Heuristic approaches, on the other hand, aim to get sufficiently close to the real optimum (or even reach it) while being more computationally efficient and hence more suitable in practice.

In this thesis, Genetic Algorithms (GA) as such a metaheuristic[1] are presented as a promising method for inventory optimisation. GA are a class of search algorithm that imitate the principles governing Darwinian evolution in genes by searching through populations of chromosomes (i.e. possible solutions), where in each iteration some members of the population die and others reproduce with variations, while progressively adapting to their environment (see Reeves and Rowe, 2002, p.3). More concretely, in this thesis

[1] Generally speaking, the difference between heuristics and metaheursitics is, that the former are usually problem-specific whereas the latter are applicable across a wide range of problems.

we will set up a four-actor serial supply chain consisting of retailer, distributor, manufacturer, and supplier, and a corresponding GA to find the optimal base-stock levels of all echelons in the sense that the total supply chain cost ($TSCC$) is minimised.
The remainder of this paper is structured as follows. We will first give a comprehensive overview of the existing scientific literature on the serial supply chain model as well as a brief outline of different applications of genetic algorithms. In Section 3, a serial supply chain model is derived and the mathematical optimisation problem is formulated. Following this, we take a closer look at the concept of genetic algorithms, their mathematical background and properties as well as their limitations. Section 5 combines the two previous sections and we implement a GA for our SC model. The performance of the algorithm is evaluated via a range of empirical tests in Section 6 before we conclude the thesis in the last section.

2 Literature Review

To explore the research efforts made in the fields of serial supply chains as well as genetic algorithms we will start by giving an overview of the relevant literature.

2.1 Literature on Serial Supply Chains

The concept of serial supply chains entered the inventory management arena in 1960 when Clark and Scarf introduced the multi-echelon inventory problem, which for the first time allowed for the integration of multiple actors (or echelons) in a supply chain (Clark and Scarf, 1960). This novel approach allowed the authors to explore many more details of the interdependent relationship between up- and downstream actors, such as the effect of inventory shortage at one installation when an order is placed from its respective downstream installation. Accordingly, they broke with the traditional approach of optimising the order quantities of each actor individually and defined a system-wide cost function that takes into account the distinct parameters on all levels. As a result, the previously required impractical recursive computation of optimal solutions on each level was made redundant, which drastically reduced the mathematical and computational complexity of the problem without sacrificing optimality (Clark and Scarf, 1960) and (Shang and Song, 2003, pp.618). What is more, Clark and Scarf (1960) showed that for a finite time horizon, the so-called base-stock policy is optimal (cost-minimal). A base-stock (or order-up-to) policy is governed by a policy vector holding one variable for each echelon denoting its respective base-stock level. Whenever the on-hand inventory of an installation falls below its base-stock level, the difference between target and actual inventory is ordered from the upstream member. The optimality of such a base-stock policy was extended to an infinite-time horizon by Federgruen and Zipkin (1984) who pointed out that the algorithm gains in simplicity in this scenario. Chung et al. (2001) contributed to the literature by proposing an exact algorithm for maximising the expected profit of the supply chain in a single-period stochastic-demand model and characterised the properties of the optimal solution. As the practicality of exact solutions remained limited by computational capacity and reliance on simplified assumptions, researchers put an increasing focus on the development of heuristic solutions that reached satisfactory results, while requiring only a fraction of the computational power. Gallego and Zipkin (1999) derived several heuristic solutions taking advantage of empir-

ical observations regarding the behaviour of the model in light of changing parameters, pointing out the pivotal role of the growth of holding costs, which tend to be higher for downstream members, with regards to the optimal base-stock level. Aiming to provide a heuristic that is both easily adoptable by practitioners as well as sufficiently effective, Shang and Song (2003) derived a simple heuristic for an N-echelon serial supply chain by minimising $2N$ single-stage newsboy-type[1] cost functions yielding upper and lower bounds which can subsequently be averaged to obtain a heuristic solution.

While the research referenced so far made important advances with respect to exact and heuristic solutions, simulation presents another crucial pillar for inventory optimisation. Clark (1960) first made the case for the use of simulations since analytic solutions - albeit available - might either be too complex to be practical or would only work properly for unrealistic scenarios due to excessively restrictive model assumptions. This call was followed by Towill et al. (1992), who used simulation techniques to empirically evaluate demand amplification in different supply chain strategies such as Just-In-Time (JIT) policy or elimination of the distributor. Lee and Billington (1993) provided a real-world example of a serial supply chain under central control in their development of a model for material management at Hewlett-Packard by heuristically determining the base-stock levels and evaluating the results using simulations. Glasserman and Tayur (1995) introduced a novel methodology called infinitesimal perturbation analysis (IPA), an estimation of the derivatives of inventory costs with respect to policy parameters. After an initial simple approximation of the base-stock levels which is evaluated via simulations, IPA is then used to steer the search in the direction of optimality. As part of the launch of a new product line at Caterpillar, Rao et al. (2000) developed a supply chain model that relied on IPA among other simulation-based optimisation techniques and demonstrated the applicability of the approach. Adding to the existing repository of inventory optimisation models, Petrovic et al. (1998) introduced a fuzzy modelling approach which accounted for both stochastic customer demand and stochastic external supply of raw material and an evaluative simulation was employed. Ettl et al. (2000) proposed a conjugate gradient search algorithm to find optimal base-stock levels constrained to a given service-level, which could also be used for base-stock evaluation.

Simulated Annealing (SA), a frequently used metaheuristic for finding global optima in large search spaces, was introduced as an optimisation method while the evaluation was conducted through a ranking and selection procedure, a statistical method for solving discrete stochastic optimization problems (Ahmed and Alkhamis, 2002). The authors stipulated that their iterative method combined with the ranking and selection procedure almost certainly converges to the global optimum. A similar path was taken by

[1] In the newsboy/newsvendor problem an actor needs to order a certain number of products on a given day without knowing beforehand how high the demand will be. Both underage as well as overage causes opportunity costs, which is known. More on this topic can be found in Porteus (2008)

Daniel and Rajendran (2005a) who also employed SA but used simulation to evaluate the base-stock levels and found that their approach yielded exceedingly good results if a simple heuristic solution is fed into the algorithm as a starting point. Chaharsooghi et al. (2008) resorted to a reinforcement learning (RL) model and compared it to a GA-based optimisation and simple 1-1 replenishment where agents order from the upstream whatever is ordered from their downstream. The RL model managed to surpass the simple 1-1 policy in all tests and the GA-based method in some cases.
Kimbrough et al. (2002) outlined a GA to determine an optimal order policy which was then evaluated by means of simulation. They showed that optimal solutions from simple situations are not generalisable to similar but more complex set-ups for which exact solutions are unknown and that in these complex scenarios their algorithm quickly outperformed the supposed optimal policies from the simple case. Daniel and Rajendran (2005b) picked up the application of a GA and extensively analysed its performance in comparison to a random search procedure as well as the optimal solution obtained by complete enumeration of the solution space. They showed that GA performed impressively, arriving at or very close to the optimal solution - a result that was shown to be robust even as the assumption of deterministic lead times is relaxed. The algorithm and SC model presented in the work of Daniel and Rajendran (2005b) will serve as the basis for the implementation and empirical testing in the remainder of this thesis.

2.2 Literature on Genetic Algorithms

The term Genetic Algorithm was first introduced by John Holland in 1975 in his book *Adaptation in Natural and Artificial Systems*, which paved the way to what is now a method that is widely used over a rich spectrum of fields (Holland et al., 1992). As metaheuristics, GA are not restricted to a specific domain but can be applied to all types of (usually) discrete and combinatorial problems (Bianchi et al., 2009). In the following section, we will present a brief selection of such applications from all kinds of fields to demonstrate the versatility of this method.
One application of the algorithm in biology was introduced by Parsons et al. (1995) as part of the well-known Human Genome Project. The authors' approach was aimed at ordering a set of DNA fragments to form a more consistent map for sequencing. A strand of DNA that is too long to be sequenced directly is replicated multiple times, then broken randomly into smaller parts, whose overlap is evaluated in pair-wise comparisons, which determines the fitness value used in the GA. The different permutations of the fragmented sequence represent the chromosomes in the GA, which aims to bring them into one correctly ordered continuous string.
Mori and Tseng (1997) use a genetic algorithm for a multi-mode resource-constrained

project scheduling problem, where activities have different modes, i.e. durations and resource requirements in which they can be scheduled. As finding an optimal solution in a finite number of steps in large and complex projects becomes exceedingly computationally hard, a genetic algorithm that incorporates domain-specific knowledge is developed and shown to outperform the stochastic scheduling method, a weighted random selection technique.

A long-standing and probably one of the best-known problems in operations research, the Travelling Salesman Problem (TSP), was addressed by Moon et al. (2002). They investigated the use of a genetic algorithm for a TSP with precedence constraints where topological sort (an ordering of vertices in a directed graph) is used in the encoding of the solution to account for the precedence relations. They found that their algorithm arrived at optimal solutions for small and medium-sized problems and outperformed other methods in larger scenarios.

Kramer (2017) provides insights into several more recent and arguably quite innovative applications of GA that further demonstrate the universal applicability of this class of algorithms. For instance, they use a GA for balancing ensembles of machine learning methods, a technique where the predictive results of several classifiers are combined to surmount weaknesses of a single classifier on its own using the idea of wisdom of the crowd. The GA, in this case, can help to find models that either achieve a minimum prediction error or a minimum runtime (or a combination in between the two extremes) depending on the user's preference (see Kramer, 2017, pp.75). Kramer furthermore presents an interesting application of GA to feature tuning in a machine learning model that aims to predict the wind level at a given wind turbine based on neighbouring turbines for short time horizons which is essential for setting up smart power grids. GA are used to determine the weights that should be given to the influence of each turbine in the model for it to yield the best results (see Kramer, 2017, pp.79).

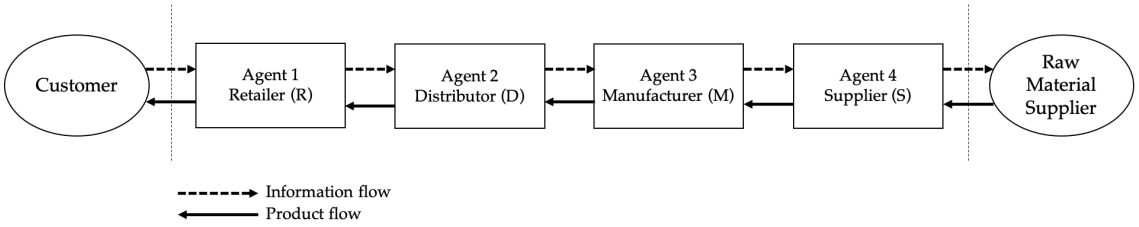
3 Description of the Serial Supply Chain Model

In the following part of the thesis, we will set up and describe the serial SC model that will be used for the rest of the thesis, specifically in our evaluation of the GA presented in Section 4. Firstly, a general SC framework is set up for which we specify several assumptions to make the framework operational. Secondly, we develop the mathematical formulation of the optimisation problem and give a verbal explanation of both the steps that the SC runs through during one time period as well as the simulation approach that we take to evaluate a given base-stock policy.

3.1 Model Assumptions

Our model draws many of its specifications from the one presented by Daniel and Rajendran (2005b), though we will make a few minor modifications, relaxing some assumptions to make the model more realistic. We consider a four-stage serial supply chain processing a single product with upstream information flow and downstream product flow.
Figure 3.1 depicts a schematic layout of the SC.

Figure 3.1: Serial supply chain with four agents, a customer and a raw material supplier. Figure based on Daniel and Rajendran (2005b).



Customer demand is placed at the retailer (agent 1) who fulfils the order directly (if enough on-hand inventory is available) or places an order to her upstream agent (distributor). The distributor behaves analogously, and so do the manufacturer and supplier. If the supplier finds herself in a position where she does not have sufficient inventory on hand, she places an order with the raw material supplier (RMS).

In the following, the term *downstream* of a given agent will denote the SC member one stage closer to the customer (thus agent 2 is downstream to agent 3), whereas *upstream* will indicate the contrary. According to the basic description above, interactions between the agents in the model as well as with the customer and the RMS take place either via information flow or product flow. The former is the order amount which reaches the upstream member after a certain time-lag (information lead time, ILT, e.g. order processing time). Likewise, product flow includes any products that are delivered from a member to her downstream member, again with a certain delay after the order reaches the agent (replenishment lead time, RLT, e.g. production or shipping time). This general framework we set up needs further specifications and a few assumptions to make it operational.

Assumptions regarding the entire SC.
The SC handles a single product. All agents of the SC follow a base-stock policy with periodic-review, where the period is a single time unit, i.e. in every period if their on-hand inventory is depleted, agents place a replenishment order with their upstream member. Any orders are placed directly and only with the subsequent upstream agent while shipments are delivered directly and exclusively to the immediate downstream member. The retailer is subjected to customer demand, which is stochastic but stationary. Specifically, demand is *i.i.d.* in a uniform distribution. All variables in the supply chain (i.e. base-stock levels, inventory, backlog, customer demand) take discrete integer values.

Assumptions regarding order and shipment processing, and inventory
There are no lot-sizes, in other words, the minimum order amount is one unit. An agent receives any orders from their downstream agent after an agent-specific stationary information lead time. Accordingly, any shipments from an agent to her downstream member arrive after the stationary replenishment lead time of the latter. Note that since information lead times are (or at least can be) nonzero, we also need to define it for the RMS who realises her demand after her ILT.

Assumptions regarding inventory
Any demand that cannot be fulfilled in the same period as it arrives at the agent is placed in the back-order queue, i.e. is backlogged, and a replenishment order is placed with the upstream member. All orders that have not been fulfilled but ordered are stored in the on-order inventory. The total inventory position held by an agent can therefore be calculated as on-hand plus on-order inventory minus back-order quantity. Generally, all agents have infinite storage capacity and the RMS has infinite supply.

Assumptions regarding costs

Two types of per-period costs occur in the SC, holding-cost and shortage-cost. The former is directly proportional to the units held in the on-hand inventory in a given period whereas the latter grows linearly to the amount that is in the backorder-queue. Note that there is no discount-policy for large orders. All costs are stationary and defined for each member locally.

3.2 Mathematical Formulation

To formulate the mathematical optimisation problem the notation must first be established. We follow the notation used by Daniel and Rajendran (2005b) and extend it where necessary (for instance w.r.t nonzero information lead time). The same goes for the mathematical formulation as well.

$TSCC$: total supply chain cost
i: installation index, i.e. agent index if $i \in \{1, ..., N\}$, customer if $i = 0$, RMS if $i = N+1$
N: number of agents in the supply chain
s_i: base-stock level at agent i
h_i: per period, per unit holding cost rate at agent i
b_i: per period, per unit shortage cost rate at agent i
ILT_i: information lead time at agent i if $i \in \{1, ..., N\}$ or at RMS if $i = N + 1$
RLT_i: replenishment lead time at agent i
$I_{i,t}$: on-hand inventory at agent i at the end of time t
$I^*_{i,t}$: on-hand inventory at agent i at the beginning of time t (after replenishment from agent $i + 1$ has arrived)
$OI_{i,t}$: on-order inventory at agent i at the end of time t
$OI^*_{i,t}$: on-order inventory at agent i at the beginning of time t
$B_{i,t}$: backlog at agent i at the end of time t
$D_{i-1,i,t-ILT_i}$: demand placed by $i - 1$ to upstream agent i at time $t - ILT_i$
$D^*_{i-1,i,t-ILT_i}$: dummy variable introduced to satisfy $D_{i-1,i,t-ILT_i} \leq 0$
$QS_{i+1,i,t-RLT_i}$: quantity shipped from agent $i + 1$ to i at $t - RLT_i$

In the SC model at hand, we consider the case where all agents are working together cooperatively to reach a common goal, namely the minimisation of the total supply chain cost ($TSCC$) over a prespecified time horizon of length T. This might seem implausible at first, however, one can easily imagine a case where a larger company incorporates an entire supply chain in its business, a strategy commonly referred to as vertical integration (see Gaughan, 2013, p.158)), thereby defining a shared objective of minimising

the total cost. At the end of the T periods, $TSCC$ is calculated as the sum of all costs accrued by all agents in all periods. Hence, we can define our objective function as follows:

$$\text{Minimise TSCC} = \min \sum_{t=1}^{T} \sum_{i=1}^{N} (h_i I_{i,t} + b_i B_{i,t}) \tag{3.1}$$

This optimisation is subject to some constraints. All the following verbal explanations are taken from the perspective of agent i in period t. Firstly, the shipping amount that arrives at the beginning of the current period shifts from on-order to on-hand inventory. Therefore, we can set the first two constraints as

$$OI^*_{i,t} = OI_{i,t-1} - QS_{i+1,i,t-RLT_i} \tag{3.2}$$

$$I^*_{i,t} = I_{i,t-1} + QS_{i+1,i,t-RLT_i} \tag{3.3}$$

The third constraint sets the end-of-period on-hand inventory equal the start-of-period on-hand inventory minus the amount that is shipped to the downstream in the current period, which can be calculated as the currently realised demand plus the change in backlog between the current and previous period.

$$I_{i,t} = I^*_{i,t} - (D_{i-1,i,t-ILT_i} + B_{i,t-1} - B_{i,t}) \tag{3.4}$$

The shipping amount to the downstream agent $i-1$ is now rather straightforward (since we already implicitly defined it in the last constraint) the difference between start-of-period and end-of-period on-hand inventory.

$$QS_{i,i-1,t} = I^*_{i,t} - I_{i,t} \tag{3.5}$$

Next, we schedule our current period order amount as the demand that arrives at $i+1$ in ILT_{i+1} periods, thus

$$D_{i,i+1,t} - D^*_{i,i+1,t} = s_i + B_{i,t} - I_{i,t} - OI^*_{i,t} \tag{3.6}$$

The end of period on-order inventory is given by its value from the start plus the quantity that was scheduled as demand at the upstream, i.e.

$$OI_{i,t} = OI^*_{i,t} + D_{i,i+1,t} \tag{3.7}$$

The last constraint determines the shipment the supplier receives from the RMS, which equals the demand that was placed from the supplier with her upstream ILT_{N+1} periods ago.

$$QS_{N+1,N,t} = D_{N,N+1,t-ILT_{N+1}} \tag{3.8}$$

What remains is a battery of non-negativity and initialisation constraints which are rather self-explanatory.

$$OI_{i,0} = 0 \tag{3.9}$$
$$B_{i,0} = 0 \tag{3.10}$$
$$I_{i,0} = s_i \tag{3.11}$$
$$QS_{i+i,i,\tau} = 0, \forall \tau \leq 0 \& \forall i \leq N \tag{3.12}$$
$$LB_i \leq s_i \leq UB_i \tag{3.13}$$
$$OI_{i,t}, OI^*_{i,t}, I_{i,t}, I^*_{i,t}, QS_{i+1,i,t}, D_{i+1,t}, D^*_{i+1,t}, B_{i,t} \geq 0, \forall i \leq N \& \forall t \leq T \tag{3.14}$$

The full model can be found in the appendix on page 53. We will refer back to this optimisation problem in Section 5, where we will develop the pseudocode corresponding to the implementation of the SC model.

4 Theory of Genetic Algorithms

In this section, we explore the concept of Genetic Algorithms, their mathematical properties, as well as a detailed overview of the steps involved in each iteration of a typical GA. We will further dive into the issue of encoding chromosomes and in the last part of this section, elaborate on their inherent limitations and discuss possible techniques to overcome some of them.

4.1 Optimisation Problem and Solution Representation

Genetic Algorithms, introduced by John Holland in 1975 in his book *Adaptation in Natural and Artificial Systems*, imitate the behaviour that is found in nature in the evolution of species (or more specifically genes) as it was proposed in the world-famous work of Charles Darwin, *On the Origin of Species* (Holland et al., 1992) and (Darwin and Bynum, 2009). GA are metaheuristic algorithms that can be used for optimisation problems whose analytical solution is often computationally tedious or - if the solution space is large enough - practically impossible to solve analytically. The archetypal example for this is the travelling salesman problem (TSP) in which one seeks to minimise the length of a roundtrip through a set of cities that an imaginary salesman has to take. As the set of cities that need to be visited grows larger, the total number of possible permutations of the tour grows exponentially making the problem exceedingly hard to solve (see Kramer, 2017, p.3).
For the sake of formality and reference, let us define a generic optimisation problem (for example (see Reeves and Rowe, 2002, pp.19)) with the objective function g that maps from the discrete search space (or phenotype space) V to $\mathbb{R}$, for which we want to find the decision variables $\mathbf{v}$ that give the minimum value of g.

$$g : V \to \mathbb{R} \tag{4.1}$$

$$\arg\min_{\mathbf{v} \in V} g \tag{4.2}$$

Note that it is possible to formulate certain GA implementations without the steps

that follow and simply use the phenotype representation explained so far, however, we would still like to explain them for the sake of completeness and since they are indeed fundamental for more advanced applications. Thus, we define the encoding function c which maps from the alphabet[1] A back to the original search space V.

$$c : A^l \to V \tag{4.3}$$

This function c provides us with a mapping from an encoded string of length l (depending on the dimensions of A and V) of each element in $\mathbf{v}$. All strings are then concatenated to give a (longer) string $\mathbf{x}$, which is the genotype representation of the phenotype $\mathbf{v}$. Furthermore, since not all strings that can be generated form A^l are valid solutions to the problem, we define the genotype space X as a subspace of A^l, i.e. $X \subseteq A^l$. Consequently, the optimisation problem translates to the following.

$$\min_{\mathbf{x} \in X} g(\mathbf{x}) \tag{4.4}$$

with

$$g(\mathbf{x}) = g(c(\mathbf{x})). \tag{4.5}$$

We can define another function f from X to $\mathbb{R}^+$, which is the fitness function that is used to assess the fitness value of a given chromosome $\mathbf{x}$.

$$f : X \to \mathbb{R}^+. \tag{4.6}$$

If a given GA implementation makes use of such encoding, all operations of the GA (e.g. crossover and mutation) are performed on the encoded (genotype) representation $\mathbf{x}$. To illustrate this, consider this arbitrary example. Assume that the search space is $V = \{0, 1, ..., 9\}$. Since one-digit integers can be represented by a bit-string of four bits, the GA performs its operations on exactly such bit-strings and we can therefore specify,

$$c : \{0, 1\}^4 \to \{0, 1, ..., 9\}, \tag{4.7}$$

[1]Let A be a finite set, then a finite sequence of elements from A is called a string over the alphabet A. For example $xxyxzzxyz$ is a string of length 9 over the alphabet $\{x, y, z\}$ (e.g. Keller and Trotter, 2017, p.17).

a function that converts four-bit binary numbers into their decimal representation. Assume now that the GA finds as a solution the binary $x = 0b1001$. Passed to the encoding function, this is converted to decimal such that $c(0b1001) = 9$, which is $\mathbf{v}$, the phenotype representation of our problem solution. In this example, it also becomes apparent why $X \subseteq A^l$, as the binary number $0b1011$ is clearly in $\{0,1\}^4$, whereas the corresponding decimal representation 11 is not in $\{0,1,...,9\}$.
As mentioned before, not every application requires this genotype-phenotype differentiation and the additional steps that come with it. In fact, we will not make this differentiation in our GA for inventory optimisation either as related research suggests that the algorithm performs excellently for the problem at hand even without additional encoding (Daniel and Rajendran, 2005b). Thus, it is fair to ask, why walk the extra mile of encoding at all? Early on, researchers and practitioners tended to follow the lead of John Holland, who strongly focused on binary encoding (see Mitchell, 1998, p.117). Holland et al. (1992) also gave a theoretical justification for the use of binary encoding based on his idea of schemata, certain patterns within candidate solutions which can be processed much more efficiently when using binary encoding. For a more detailed description of schemata, we refer to Reeves (1997), Reeves and Rowe (2002) and Mitchell (1998). In addition to the concept of schemata, encoding can be advantageous from a programming point of view (see Kramer, 2017, p.15). For example, an encoding function allows the user to add a certain sequence at the beginning or the end of an encoded solution which is understood as a command for certain behaviour by the computer. In other cases, encoding strategies such as tree encoding make room for open-ended exploration (see Mitchell, 1998, p.118). To conclude this section, let it be said that while it is not always necessary to make use of encoding, there are many applications where the right encoding strategy can have outstanding effects on the quality of the result. We once again refer to Mitchell (1998) and Reeves (1997), for an extensive elaboration on various encoding strategies.
We will now dive deeper into the concrete steps that are taken by GA in each iteration.

4.2 Iterative Process

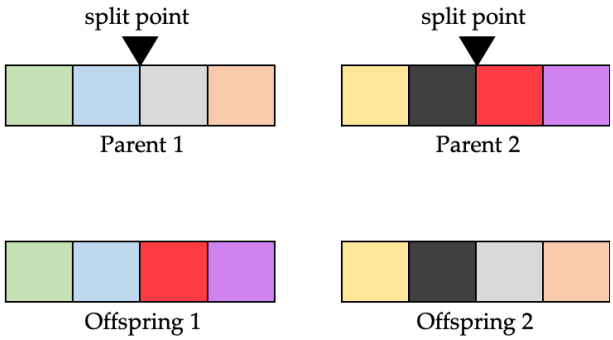
When a GA is used to solve an optimisation problem, a solution $\mathbf{x}$ is not considered on its own, instead, a population of candidate solutions is considered in each iteration in analogy to the biological scenario where at a given point in time several individuals coexist. The candidate solutions in the population are referred to as chromosomes and the elements within each chromosome are its genes or the values taken by the variables of a candidate solution (see Reeves, 1997, p.232). Even though there are numerous ways to set up a GA, an iteration of the algorithm generally runs through the following five-step

process. Assume we already have an initial population, that is a set of chromosomes with candidate solutions. This initial population is usually chosen at random from the search space (see Reeves and Rowe, 2002, p.29). The sequence of steps is as follows.

Crossover

The crossover operator combines the genes of different chromosomes according to a predefined strategy. A commonly used one is the so-called n-point crossover where two (parent) chromosomes are combined by splitting each of them at n points and swapping the fragments to create two offspring chromosomes (see Kramer, 2017, p.12) and (Reeves and Rowe, 2002, p.38). Figure 4.1 demonstrates a one-point crossover of two parent chromosomes.

Figure 4.1: One-point crossover of two parent chromosomes



Note that not all parent chromosomes must necessarily be crossed over, as we can set a crossover rate (CR) to have only some chromosomes undergo the operation. It is also possible to choose a CR which decreases with every generation Davis (1991) in Reeves and Rowe (2002). Furthermore, it is necessary to consider which pairs of chromosomes are crossed with each other. This mating selection can follow similar rules as described in the paragraph on selection (see Kramer, 2017, p.17).

Mutation

GA can only find an optimal (or good) solution if we let it progressively explore the search space. This is where mutation operators come in, whose purpose is to distort the current solution by a specified degree - the mutation rate (MR) - to allow for the exploration of new solutions (see Mitchell, 1998, p.129). These operators come with three main requirements that need to be taken into account for the GA to work frictionless (see Kramer, 2017, pp.13). Firstly, we need to make sure that all solutions within the solution space can be reached via mutation otherwise one runs the risk of hindering the algorithm from finding the optimum altogether. Secondly, mutation operators should be unbiased that is they should not cause the population to drift into a particular direction of the search space. An exception to this might be given in the case of constrained

solution spaces where a biased search might increase the speed with which the optimum is found. The last condition that should be met by operators is that of scalability, i.e. their strength should be freely adaptable to some extent. For instance, if mutation is based on a probability distribution, one could change the standard deviation or another measure of statistical dispersion to scale the operator. For bit-string representations, mutation usually works by flipping bits in the string with a certain probability (see Kramer, 2017, p.14). It is also suggested that there is need for a balance to be struck between the extent of crossover and the extent of mutation, however, this balance is very problem-specific and can therefore not be generalised (see Reeves and Rowe, 2002, p.45). We will deal with this problem by performing an extensive pilot study to determine a reasonable choice of CR and MR.

Genotype-Phenotype Mapping
This step is only necessary if the differentiation between genotype and phenotype is implemented, i.e. if encoding via an encoding function takes place, and evaluation can only be done based on the phenotype representation. The chromosomes are passed to the encoding function to transform the genotype representation back into phenotype form to allow an evaluation of the value of the objective function given the current candidate solution (see Kramer, 2017, p.15). Since our implementation does not make use of encoding, we can skip this step.

Evaluation
In the penultimate step of each iteration, all chromosomes are evaluated by plugging the candidate solutions into the fitness function to determine their respective fitness values. This is not always a straightforward process as their might exist multiple objectives which give rise to the question of how each objective should influence the fitness value. If GA are used for mathematical optimisation, this question does not pose itself as the fitness of a chromosome equals the result when passed to the objective function (see Mitchell, 1998, pp.7). However, since this is not always as easy as merely plugging the candidate solution into a function, one might rely on running a simulation on the candidate solution to evaluate its fitness. Whichever form the fitness evaluation takes, the number of times the fitness function is called presents a possible bottleneck to the performance of the algorithm since for example in the case of simulation-based evaluation running the simulation might require significant computational capacity each time a solution is evaluated. Therefore, GA are often compared using the number of calls of the fitness function until the optimal solution is reached where less calls signify better performance (see Kramer, 2017, p.16). In some cases, when fitness evaluation is particularly expensive, it could prove advantageous to perform only a partial evaluation (see Reeves and Rowe, 2002, p.45). To take the problem at hand as an example, instead

of evaluating the fitness of a candidate solution by simulating the SC model for T units of time, one could settle for $T/2$ periods instead and presume the result to maintain valid.

Selection
Lastly, a subset of all currently available chromosomes, that is parents and offspring, ought to be selected to be the new parent population in the next iteration to incorporate Darwin's principle of survival of the fittest. This selection generally aims to take the best current chromosomes into the next generation (whereas the not selected chromosomes die) according to a certain selection rule (see Kramer, 2017, pp.16). Most rules can either be classified as deterministic or probabilistic selectors. Examples for the former are the elitist rule, which selects the best solutions of the offspring as parents or the comma selector, which chooses the μ best solutions from λ offspring solutions. A well-known randomness-based selection rule is called fitness proportional or roulette-wheel selection, according to which the relative fitness of each chromosome in a population is first calculated using normalisation, which then positively influences the probability of being selected (see Reeves and Rowe, 2002, p.31). The main advantage of using fitness proportional selection lies in the fact that all chromosomes have a nonzero probability of being selected for the next generation, which opens the door for forgetting the best current solution to overcome local optima (see Kramer, 2017, p.17).

These five steps are repeated until a specified termination criterium is reached. Suitable criteria can be manifold. A common and very simple one, however, is termination after a predefined number of generations has been processed. The process could alternatively be terminated if the rate of convergence (the change in improvement per generation) falls under a predefined level, since this would signify that the optimum has been reached already (see Kramer, 2017, pp.17).
The entire iterative process is visualised in Algorithm 1 in the appendix which depicts the pseudocode for a generic GA. Note that the if condition in line 19 makes sure that genotype-phenotype mapping is only performed if the implementation makes use of this differentiation.

4.3 Caveats and Limitations

No algorithm comes without its downsides and even though GA are a broadly applicable and very potent tool, they too have their caveats. In the following section we explore several of these potential pitfalls and - where we can - discuss possible solutions.

4.3.1 Parameter Tuning

Wolpert et al. (1995) and Wolpert and Macready (1997), proposed a range of No-Free-Lunch Theorems (NFL) for search and optimisation in which they formulated the impossibility of finding a one-size-fits-all approach to optimisation. One of the NFL theorems they postulated concerns the aspect of parameter tuning in GA: no parameter setting delivers optimal results for all optimisation problems. It is also hardly possible to give general recommendations that work for all applications alike, however, Reeves and Rowe offer a few broad guidelines to consider (see Reeves and Rowe, 2002, pp.275). Due to this lack of general setting, practitioners implementing their own GA need to find the particular parameter settings that maximise the performance of their implementation. What results is an optimisation problem before the actual optimisation problem (see Kramer, 2017, p.22). Even for plain-vanilla GA this already includes multiple parameters (e.g. CR, MR, population size, ...) whose number could increase significantly for more complex implementations. During our implementation later we will see how extensive this pre-optimisation can get even when only three parameters need tuning. Unfortunately, there is not really a way around this need for parameter tuning. Expert or domain knowledge about the problem can certainly help with finding suitable parameters more quickly, but at the end of the day, proper optimisation is the only real solution to the problem. Nevertheless, certain strategies exist to make parameter tuning more efficient than merely brute forcing the optimal setting. For instance, Kramer (2017) suggests using a meta-GA that takes parameter settings as its chromosomes. This approach is known to yield decent parameter settings, albeit at quite some overhead due to fact that each call of the evaluation function of the meta-GA runs several iterations of the GA itself (see Kramer, 2017, p.23).

Alternatively, one could employ parameter control strategies which attempt to find beneficial parameters for the GA during the run itself, such as deterministic control strategies that define an exogenous rule to modify parameters on the go (see Eiben et al., 2003, p.138). For example, it is generally agreed to that a decrease in the MR over the course of several generations should be reduced to enable the GA to converge to the optimal solution. This could either be done using a linear or a multiplicative rule. Suppose the MR of the GA in iteration n is denoted by $MR(n)$ then a linear rule would decrease the rate by $MR(n+1) = MR(n) - \kappa$ while a multiplicative rule would compute $MR(n+1) = MR(n) * \kappa$ with the control constant κ (see Kramer, 2017, p.23).

The second category are so-called adaptive control strategies, a well-known example of which is the Rechenberg rule that aims to increase the MR of the system whenever possible by considering feedback it receives from the algorithm itself (see Eiben et al., 2003, p.138). Using the Rechenberg rule, the MR is increased exponentially whenever we observe that the algorithm produces superior offspring compared to its parents (con-

sidering the best chromosome from the population) in more than one in five cases, while undercutting this success rate would lead to a decrease in MR (see Kramer, 2017, p.24). For instance, let MR be the mutation rate in one generation and $RX \in (0, 1)$ the Rechenberg parameter and S the success rate of previous generations, then in the next generation MR is adapted such that,

$$MR' = \begin{cases} MR * RX & S < 0.2 \\ MR/RX & S \geq 0.2. \end{cases}$$

In the third and last category, self-adaptation, each individual in the population is stored together with their own MR that is subject to its own crossover and mutation and is inherited to offspring (see Eiben et al., 2003, p.138). Crossover and mutation of the MR follows one of the crossover rules we explored earlier whereas the mutation of the MR usually relies on specific operators (see Kramer, 2017, p.26). For example, the mutation of a chromosome specific MR is often subjected to a log-normal operator, i.e. $MR'_k = MR_k * exp(\tau * \mathcal{N}(0, 1))$ where k is the index of the chromosome and τ is the global mutation rate of mutation rates (see Kramer, 2017, p.26).

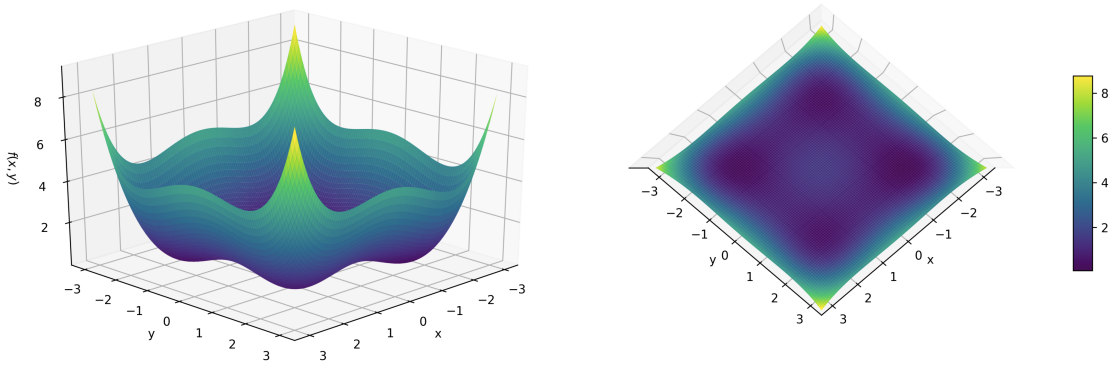
4.3.2 Multimodality

A common problem in optimisation concerns multimodality, that is functions with a hilly landscape with several local optima of different height. While one is generally interested in finding the global optimum of a given function, in practice, there are many cases where finding local optima might also be desirable since these solutions might have convenient practical implications (see Singh and Deb, 2006, pp.1305). This issue is closely related to the exploitation-exploration dilemma: how much should the neighbourhood of a suspected optimum be trusted (exploitation) opposed to exploring new areas of the solution space (exploration) (see Eiben et al., 2003, p.29). The problem from an algorithmic point of view is that the GA itself has no way of knowing whether the found optimum is local or global, which depending on the practical context of the problem, might be a crucial distinction (see Kramer, 2017, pp.31). To demonstrate the behaviour of GA when encountering a fitness function with a multimodal landscape, we implemented a generic GA (e.g. similar to Algorithm 1 in the appendix) and ran it to find the solutions $\mathbf{s}$ of the minimisation of the multimodal function

$$f(x, y) = \frac{1}{10}((x^4 - 5x^2 + y^4 - 5y^2) + \frac{1}{2}xy + \frac{3}{10}x + 15) \tag{4.8}$$

a visualisation of which is depicted in Figure 4.2

Figure 4.2: Topological plot of the multimodal function 4.8



Observe that the function has four local minima (s_1, s_2, s_3, s_4) where $s_i = (x_i, y_i)$ with the corresponding function values $(f(s_1), f(s_2), f(s_3), f(s_4))$ which were obtained analytically and are depicted in Table 4.1.

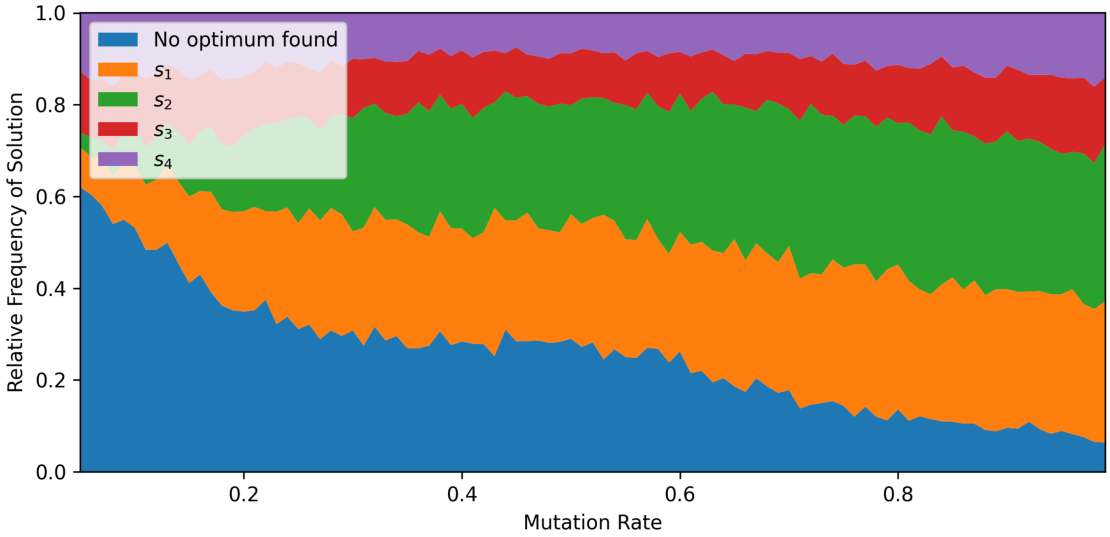
	x	y	$f(x,y)$
s_1	$3 * \sqrt{0.3}$	$-3 * \sqrt{0.3}$	≈ 0.1723
s_2	$-3 * \sqrt{0.3}$	$3 * \sqrt{0.3}$	≈ 0.0737
s_3	$-2\sqrt{2.3}$	$-2\sqrt{2.3}$	≈ 0.3275
s_4	$2\sqrt{2.3}$	$2\sqrt{2.3}$	≈ 0.4185

Table 4.1: Minima of the multimodal function 4.8

Since we are facing a minimisation problem s_2 is clearly the global minimum and hence our optimal solution. However, our simple GA could not consistently find the global optimum but instead sometimes converged to another local minimum or did not find any satisfactory solution at all. Figure 4.3 depicts the frequency with which each of the possible solutions (or none of them) was found by the algorithm after 25 generations for different mutation rates allowing for a deviation of 10% from the exact minimum value. We set the number of generations to 25 since in most runs the GA had converged by this point. If mutated, a gene x is altered (uniformly) at random in the interval $[0.8 * x, 1.2 * x]$. The mutation rate (i.e. the probability that a chromosome is mutated) is incremented from 0 to 1 in steps of 0.01. Each MR-setting was run 1000 times and the average frequencies were taken. Furthermore, we limited the search space to the interval $[-10; 10]$, so that the GA only looks for solutions in this interval. This is quite a strong simplification, especially, since in practice we might not be able to limit the search space so drastically but instead need to search a much larger interval. We shall promptly see how a larger search space compromises the solution quality significantly.

Looking at Figure 4.3 it is not unequivocal which MR we should choose for our optimisation. For example, we could opt for the setting in which the frequency of solution s_2

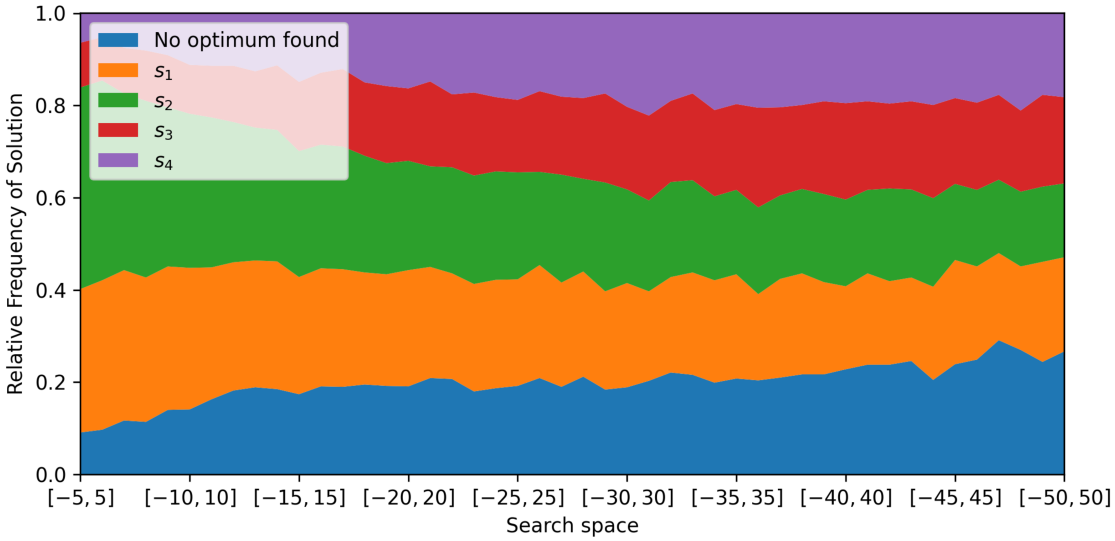
Figure 4.3: Relative frequency of local optima of function 4.8 found in 1000 runs of a simple GA with mutation rates between 0 and 1 with search in the interval $[-10, 10]$



is the highest (which is the case for $MR = 0.72$ where s_1 is found with a probability of 36.9%). However, one could also argue that $MR = 0.99$ (i.e. mutation on 99% of genes) is the best setting since it minimises the probability of not finding any of the four minima. For our analysis of the impact of a larger search space on the solution quality we choose the former setting ($MR = 0.72$) although our conclusions would remain unchanged for other settings. Figure 4.4 depicts the solution frequency after 25 generations for different symmetric search spaces (incremented by 1) when MR is fixed to 0.72 (again using 1000 runs per setting).

As we can clearly see, the solution quality is progressively inhibited by an increase in search space size such that the chance of finding no optimum at all is as high as 29.7% if we choose the interval $[-47; 47]$ and the probability of discovering the global optimum using this search space is diminished to 15.9%. Admittedly, the GA employed is certainly of a very simple kind and the performance could most definitely be increased by using more advanced methods, however, we must also note that the optimisation we face is not particularly hard either. In practice, one could face problems with many more local optima than in our sample function and the solutions could lay substantially further apart from each other, which would require a larger search space as well. Hence, it seems fair to conclude that GA do have a hard time when faced with complex multi-peak fitness landscapes and might not always be able to find sufficiently good solutions. To deal with this problem and to increase the chance of finding sufficient solutions nonetheless, several methods have been developed, two of which will now be presented. One group of strategies are so-called restarts, that is running the algorithm multiple

Figure 4.4: Relative frequency of local optima of function 4.8 found in 1000 runs of a simple GA with symmetric search spaces between $[-5; 5]$ and $[-50; 50]$ with $MR = 0.72$



times starting from different points each time (possibly with different parametrisation) and comparing the different optima that were found this way (see Kramer, 2017, pp.32). Global mutation rate control is such a strategy that increases the global MR if the same local optimum is found (with predefined accuracy) in more than p % of cases to boost exploration whereas it is decreased otherwise. The starting point for the next iteration in this case is the previously found local optimum and the parameter p is problem specific (see Kramer, 2017, pp.32).

An alternative approach to overcoming local optima is novelty search, the exploration of new parts of the solution space that might accommodate local or global optima. This requires a novelty condition that is satisfied if the newly generated candidate solution is sufficiently different from previous candidates (typically measured using distance measures such as Euclidian distance). Every time the GA creates new chromosomes this novelty condition is checked and only chromosomes that fulfil the condition are accepted (see Kramer, 2017, pp.35).

Beyond the two presented approaches there is a wide range of further methods that can be used to overcome stagnation in local optima, however, elaborating on those is beyond the scope of this paper. Instead, we refer to Chapter 4 of Kramer (2017), for a concise overview.

4.3.3 Convergence to the Global Optimum

The ultimate goal of any optimisation algorithm is to find the global optimum of the objective function. Hence, the question arises whether GA are in fact capable of doing

so.

The short answer to this question is yes, if a number of conditions are met for any given population (see Eiben et al., 2003, p.198). Firstly, a GA (or any evolutionary algorithm for that matter) ought to allow every individual to be selected as a parent (i.e. for crossover) with a nonzero probability. Furthermore, every individual must have a nonzero probability to be selected for survival. As a third condition, the authors state that the survival selection mechanism must be elitist. Note that in order to not contradict the second condition, this requires that the best available candidate solution must be selected for survival while still guaranteeing a nonzero survival probability for all other candidate solutions. The fourth and last condition requires that by means of mutation, every solution in the search space must be reachable.

The long answer to this question is a more sophisticated mathematical proof of the four stated conditions presented by Eiben et al. (1990). For our purposes, it suffices to keep the verbal conditions in mind when implementing a GA.

5 Implementing a GA for Base-Stock Level Optimisation

In this section and the following subsections, we will present the implementation of a Genetic Algorithm for inventory optimisation. In particular, the GA is implemented to optimise the base-stock levels (in a minimal-$TSCC$ sense) of the four agents in the model presented in Section 3. In the process, we will test several configurations of the algorithm to explore the impact on performance of some of the methods presented in the previous sections. We will now proceed by presenting the pseudocode of our models while elaborating on some of the relevant details more thoroughly[1].

5.1 Supply Chain Model Implementation

Algorithm 2 in the appendix gives the pseudocode for the serial supply chain model described in Section 3. After initialising some variables, the algorithm enters two nested for-loops the outer one iterating over the time periods from 1 to T while the inner one steps through the agents of the SC. Note that two additional dummy variables, *newbacklog* and *regship*, are introduced for ease of programming that are not listed in the mathematical formulation of the model and are assigned anew in every iteration. Further, in the case of the retailer ($i = 1$), demand from the customer $D_{0,1,t}$ is drawn from a uniform distribution $\mathcal{U}(a, b)$ for all t. In the case of the supplier ($i = 4$) placing an order with the upstream (i.e. RMS) automatically causes the scheduling of the shipment of the order amount ILT_{RMS} periods in the future since the RMS is not an active member of the model and therefore cannot schedule the shipment.
After completing the two for-loops, the holding and shortage costs of all periods on all stages are summed to yield the $TSCC$, which also corresponds to the fitness value of the base-stock level setting used. Hence, every time the GA calls the fitness function, a simulation on the above model is run and the resulting $TSCC$ is returned to the GA.

[1]The actual implementation was done in Python.

5.2 GA Implementation

Algorithm 3 (Appendix) presents the pseudocode used for the base version of the GA we use in our implementation.

5.2.1 Parameters

Firstly, a few parameters are introduced that are required in the different steps of the algorithm.
LB_i, UB_i: The lower and upper bounds on the base-stock levels of each agent used for the initialisation of the first parent population are calculated using the concept of minimum and maximum lead times. This is done in accordance with Daniel and Rajendran (2005b), however, since our model extends theirs by allowing information lead times to be nonzero, we adapt their procedure accordingly. The base-stock level lower bound of agent i is the product of the minimum demand at i per time unit and the minimum lead times w.r.t the agent. Since the demand is drawn at random from a uniform distribution $\mathcal{U}(a, b)$ the lower bound is thus $LB_i = a * (RLT_i + ILT_{i+1})$. This is so, since in the best-case scenario (that is when demand is as low as it gets and the upstream member has no shortage) replenishment of that demand takes at least $RLT_i + ILT_{i+1}$ time periods for agent i. Therefore, each agent must stock the minimum demand a for every time unit it takes to replenish. The upper bound corresponds to the maximum of the aforementioned measures and is computed as $UB_i = b * (\Sigma_{j=i}^{N} RLT_j + \Sigma_{j=i+1}^{N+1} ILT_j)$, since as a worst case scenario the lead times of all upstream actors accumulate if none of them has any on-hand inventory and customer demand is maximal.
MX: MX determines the strength of mutation, i.e. if a gene of value x is mutated the result is limited to the interval $[x * (1 - MX); x * (1 + MX)]$
MP: MP is the probability with which any given chromosome is mutated.
CR: CR gears the crossover procedure of the algorithm by setting the probability that a given chromosome is crossed with another chromosome.
n: The parameter defines the population size, i.e. the number of candidate solutions in one population.
max_gen: The parameter sets the number of generations the GA creates until it is terminated.

5.2.2 Iterative Steps

The general procedure of the algorithm corresponds to that of the generic GA earlier; after initialising the first parent population the steps crossover, mutation, evaluation, and selection are repeated until the last generation (max_gen) is reached. Note that no genotype-phenotype mapping is necessary since we do not differentiate between the two

and do not encode the solutions. The four different functions will be explained briefly while all details are depicted in the pseudocode.

Crossover occurs via a random operator that shuffles the order of the individuals in the current parent population to randomise the pairs of chromosomes which are to be crossed over. Subsequently, we iterate over the pairs of individuals, drawing a random uniform number between 0 and 1 in each iteration. If this number is smaller or equal to the crossover rate CR the pair is crossed and added to the intermediate population, otherwise the chromosomes are added to it as is.

The resulting intermediate population is passed as input to the mutation function where we iterate over all genes in all chromosomes of the intermediate population. In every iteration, we draw another random uniform number u between 0 and 1 which is compared to the mutation probability MP. If $u \leq MP$ the mutation is performed on the current gene in the sense that we calculate the new mutated gene as $newgene = gene*(1-MX)+gene*2*MX*u$. Note that this essentially limits the mutation to the interval from $gene*(1-MX)$ if $u = 0$ to $gene*(1-MX*(1+2*MP))$ if $u = MP$, which shows how MX is used to determine the strength of mutation. The resulting mutated genes are saved in new chromosomes which are placed in an updated intermediate generation. In the case where $u > MP$ the genes are not mutated and directly placed in new chromosomes which are also placed in the new intermediate generation. This procedure implies that there is a nonzero probability for each gene to be mutated, steered by the parameter MP. After completing mutation, the candidate solutions in the intermediate population are ready for *evaluation* and are passed to the fitness function, which in our case starts an SC simulation over $T = 1200$ periods using the candidate solution as base-stock levels. The $TSCC$ resulting from the simulation is passed to the input chromosome and saved alongside it. In the last step of each GA iteration, a new generation of the now evaluated chromosomes of intermediate and parent population are selected to survive as parents in the following iteration. Concretely, we employ an elitist selection operator that chooses the n best chromosomes from $(int\_pop \cup par\_pop)$. The elitist operator is adopted from Daniel and Rajendran (2005b). We would like to point out that this operator does not fully fulfil the third condition required for convergence as presented by Eiben et al. (2003) (see Section 4.3.3) since for the n chromosomes with the lowest fitness values, the probability of being selected for survival is zero. While still guaranteeing convergence, we can no longer be certain that the GA will eventually converge to the global optimum. Nevertheless, we will still make use of this operator to test the approach presented in the paper by Daniel and Rajendran (2005b). Algorithm 3 in the appendix presents the pseudocode corresponding to this algorithm.

As described earlier, there exists a range of configurations to alter the GA that could impact its performance. One such approach is using a different selection rule since elitist selection might suffer from being too confident in currently good solutions, forgoing

search in other directions where other local (or even global) optima might be located. To deal with this problem, the previously presented roulette wheel selection (or fitness proportionate selection) procedure can be employed, the pseudocode of which is depicted in Algorithm 4 in the appendix. In the original GA from Algorithm 3 we would simply substitute calling the *selection* function with calling *rouletteSelection* instead. In its first step, the function iterates over all chromosomes in one or more input populations and calculates their relative fitness $p_i = \frac{f_i}{\Sigma_{j=1}^{N} f_j}$ where i is the index of the current chromosome. Subsequently, the cumulative value of all p_i is computed as $CumP_i = \Sigma_{j=1}^{i} p_j$. This creates an imaginary roulette wheel where the probability of hitting a given field (i.e. chromosome) is proportionate to the relative fitness of this chromosome. We implement this by progressively filling a new population by spinning the roulette wheel and adding the resulting chromosome to the population until it contains n individuals. The possible advantage of this procedure compared to elitist selection is that all chromosomes have a nonzero probability of being selected for the next parent population allowing for a higher degree of exploration of the algorithm which could potentially overcome local optima. However, we sacrifice guaranteed convergence when employing roulette wheel selection, as it is also possible that we increasingly deviate from good solutions (see Reeves and Rowe, 2002, p.275).

The roulette crossover procedure presented above can also be used as an operator to determine the chromosome pairs to be crossed in the crossover step by running the *rouletteSelection* function on the parent population. Chromosomes with better relative fitness values are more likely to be added to the output population early on in the process, hence increasing their chance to be paired together. This way, we tend to cross chromosomes of similar fitness value with each other with the idea of taking advantage of good candidate solutions by making them even better. Again, we can simply call the *rouletteCrossover* function instead of the *crossover* function in the base version of the GA to implement this configuration. The corresponding function is presented in the appendix as Algorithm 5.

In the next section we combine some of the alternative GA configurations for crossover and selection with each other to test which configuration proves to be most suitable.

6 Empirical Testing

In the following section we will extensively explore the empirical results of four different SC settings, which are depicted in Table 6.1. Note that costs are increasing towards the downstream as a result of the value added to the product in each step. Conversely, lead times are lower at the downstream since we assume those members to be closer to each other (e.g. retailers are usually at close proximity to their customers) which reduces shipping time. Likewise, orders are assumed to take more processing time towards the upstream which also increases lead times towards the upstream. The extent of this decrease (increase) in costs (lead times) towards the upstream is varied in each of the four settings.

We use antithetic sampling in the generation of customer demand to reduce sample variance. Antithetic sampling describes the process of drawing random numbers from a probability distribution that are negatively correlated to each other (see Kroese et al., 2013, p.105). For a sample of $2n$ random numbers drawn from a uniform distribution between 0 and 1, this can be achieved by drawing the first n numbers $u_1, u_2, ..., u_n$ as we normally would, and then calculating the second half of the sample as the respective antithetic variable of the first half, i.e. $u_{n+1} = (1-u_1), u_{n+2} = (1-u_2), ..., u_{2n} = (1-u_n)$ as was explained by (see Daniel and Rajendran, 2005b, p.114).

6.1 Parameter Tuning

As was explained earlier, implementing a GA comes with the necessity of choosing appropriate parameters to allow the algorithm to converge to a sufficiently good solution. While this parameter tuning becomes increasingly difficult and computationally exhaustive with increasing complexity of the implementation, even the relatively straightforward situation that we are facing is already quite a feat.

There are three parameters that are to be tuned for the base-stock level optimisation GA. Firstly, the crossover rate CR ought to be set to determine the probability that a given individual in the population is crossed over. We further have two parameters that determine the mutation rate, MP and MX, the former determining the probability that a gene is mutated, the latter specifying the strength of mutation, i.e. the maximum percentage by which a gene may be altered.

	R	D	M	S	RMS
S1: Base setting. Homogenous costs and lead times					
HC	4	3	2	1	-
SC	8	6	4	2	-
RLT	2	3	4	5	-
ILT	1	2	3	4	5
Minimum LT	4	6	8	10	-
Maximum LT	28	24	18	10	-
S2: Homogenous costs, heterogenous lead times					
HC	4	3	2	1	-
SC	8	6	4	2	-
RLT	2	4	16	32	-
ILT	1	3	9	18	24
Minimum LT	5	13	34	56	-
Maximum LT	108	103	90	56	-
S3: Heterogenous costs, homogenous lead times					
HC	12	8	4	1	-
SC	24	12	6	3	-
RLT	2	3	4	5	-
ILT	1	2	3	4	5
Minimum LT	4	6	8	10	-
Maximum LT	28	24	18	10	-
S4: Heterogenous costs and lead times					
HC	12	8	4	1	-
SC	24	12	6	3	-
RLT	2	4	16	32	-
ILT	1	3	9	18	24
Minimum LT	5	13	34	56	-
Maximum LT	108	103	90	56	-

Table 6.1: SC settings used during empirical testing

There is no single right approach to parameter tuning as any approach requires balancing a trade-off between setting the parameters to broad, i.e. setting them globally, which would lead to unsatisfactory performance in certain settings, or tuning them specifically for each setting, which could lead to overfitting and furthermore requires an even larger computational effort when additional settings are considered. In this thesis we adopt an approach that falls into the first category for two reasons. Firstly, our computational capacities are limited making setting-specific parameter tuning infeasible. Secondly, Daniel and Rajendran (2005b) also proposed a global tuning approach which we will adopt in this thesis.

In Section 5.2 we presented the GA used for optimising base-stock levels including two possible modifications that could impact its performance - substituting random crossover with roulette wheel crossover and using roulette wheel selection instead of elitist selection. As a benchmark, we also implement a sequential random search process (RS)

that starts with a random solution x, alters it in every iteration by the search radius $SR \in (0, 1)$ (i.e. in the interval $[SR * x, (1 + SR) * x]$) and adapts the newly found solution if its fitness value is better than the one of the original solution. Since random search processes are not the subject of this thesis, we skip the underlying theory and refer to Zabinsky (2003) for more details. Table 6.2 gives an overview of the different algorithms we tested empirically. GA1 corresponds to the configuration used by Daniel and Rajendran (2005b).

GA1 (Rou-CR, Elite-Sel)	Roulette wheel crossover & elitist selection
GA2 (Ran-CR, Elite-Sel)	Random crossover & elitist selection
GA3 (Ran-CR, Rou-Sel)	Random crossover & roulette wheel selection
RS	Sequential random search process

Table 6.2: Algorithms used during empirical testing

Theoretically, each of the three GA versions would require a separate pilot study for parameter tuning. Unfortunately, this approach is exceedingly time consuming as we will see shortly and therefore cannot be implemented in full. Instead, complete parameter tuning is only performed for GA1 as it is the version that is used by Daniel and Rajendran (2005b). Subsequently we limit the parameter search space to values surrounding the optimal parameters found in the pilot study for GA1 as we expect that these should at least be close to the optimal parameter settings for the other GA as they generally work quite similarly. Since RS only requires a single parameter, namely the search radius, we perform a simple pilot study to determine its optimal setting.
Hence, in the full pilot study for parameter tuning of GA1, we run the algorithm 30 times using a given parameter setting for the SC model for 1200 units of time with random demand sampled uniformly between 20 and 60. We stop the process in each simulation after 30 generations which, while not allowing for full convergence all the time, gets reasonably close to full convergence and is still within the limits of computational capacity. An extract from the results are presented in Table 6.3. Computing the relative increase in $TSCC$ for a given parameter set is done following the formula presented by Daniel and Rajendran (2005b),

$$\begin{aligned} &\text{Relative increase in } TSCC \text{ for given } \{MX, MP, CR\} \\ &= \left(\frac{TSCC_{\{MX,MP,CR\}} - \min(TSCC_{\{MX,MP,CR\}})}{\min(TSCC_{\{MX,MP,CR\}})} \right) \end{aligned} \tag{6.1}$$

where $TSCC_{\{MX;MP;CR\}}$ is the $TSCC$ obtained by the GA for the parameter setting specified in the index. The relative increase in $TSCC$ encapsulates how much worse, i.e. larger, the $TSCC$ is compared to the best $TSCC$ obtained for the SC setting. This process is repeated for all four SC settings and the mean relative increase in $TSCC$ for all of them is calculated. Table 6.4 contains an extract from the calculation results.

	TSCC			
MX,MP,CR	**S1**	**S2**	**S3**	**S4**
(0.1, 0.5, 0.5)	567913	1642875	1256094	4959872
(0.1, 0.5, 0.9)	512568	1345068	1052503	3333734
(0.1, 0.6, 0.7)	520922	1289278	1263653	3849600
(0.1, 0.7, 0.5)	499256	1222408	1131661	3184654
(0.1, 0.7, 0.9)	462042	1021706	1053697	2683696
(0.1, 0.8, 0.7)	444218	1048602	1112011	2248103
(0.1, 0.9, 0.5)	463006	1161408	1033791	2808439
(0.1, 0.9, 0.9)	435050	990458	1008917	2009930
(0.1, 1, 0.7)	436568	1040205	994911	2182661
(0.2, 0.5, 0.5)	476550	1285279	1183605	3943175
(0.2, 0.5, 0.9)	462322	1114121	1056611	2125383
(0.2, 0.6, 0.7)	457689	1002562	1060912	2464444
(0.2, 0.7, 0.5)	452163	1056609	1048716	2275190
(0.2, 0.7, 0.8)	442472	910500	995490	1886256
(0.2, 0.7, 0.9)	437782	907936	1019316	1914395
(0.2, 0.8, 0.7)	439159	921876	1004406	1889733
(0.2, 0.9, 0.5)	443917	978177	1007533	2022855
(0.2, 0.9, 0.9)	438926	922883	1004157	1874205
(0.2, 1, 0.7)	447110	955757	1016035	1959865
(0.3, 0.5, 0.5)	482466	1091977	1109694	2606763
(0.3, 0.5, 0.9)	466236	972791	1082614	2213051
(0.3, 0.6, 0.7)	450871	996329	1037474	2068642
(0.3, 0.7, 0.5)	446119	1002746	1028943	2036874
(0.3, 0.7, 0.9)	441470	930305	1012003	2077265
(0.3, 0.8, 0.7)	446795	948614	1012961	1948510
(0.3, 0.9, 0.5)	457873	975779	1028895	2031041
(0.3, 0.9, 0.9)	447717	962998	1020655	1991111
(0.3, 1, 0.7)	458574	1045324	1042062	2073316

Table 6.3: Extract from the $TSCC$ values obtained during parameter tuning of GA1

This process is extremely laborious and time consuming as the GA is run many times, every run calling a simulation of the supply chain multiple times. During our calculations, we simulated all combinations of the following settings:

$$\begin{aligned} CR &\in \{0.5, 0.6, ..., 1\} \\ MP &\in \{0.5, 0.6, ..., 1\} \\ MX &\in \{0.1, 0.2, ..., 0.5\} \end{aligned} \tag{6.2}$$

This resulted in $6^2 * 5$ different parameter settings, each running the GA 30 times with 30 generations and a population size of 20. Accordingly, the fitness function, i.e. the SC simulation, is called $6^2 * 5 * 30 * 30 * 20 = 3240000$ times per Supply Chain Setting.

	RIiF				
MX,MP,CR	**S1**	**S2**	**S3**	**S4**	**Mean**
(0.1, 0.5, 0.5)	0.3071	0.8103	0.2686	1.6561	0.7605
(0.1, 0.5, 0.9)	0.1797	0.4822	0.063	0.7853	0.3775
(0.1, 0.6, 0.7)	0.1989	0.4207	0.2763	1.0615	0.4894
(0.1, 0.7, 0.5)	0.1491	0.347	0.1429	0.7054	0.3361
(0.1, 0.7, 0.9)	0.0634	0.1259	0.0642	0.4372	0.1727
(0.1, 0.8, 0.7)	0.0224	0.1555	0.1231	0.2039	0.1262
(0.1, 0.9, 0.5)	0.0656	0.2798	0.0441	0.504	0.2234
(0.1, 0.9, 0.9)	0.0013	0.0914	0.019	0.0764	0.047
(0.1, 1, 0.7)	0.0048	0.1462	0.0048	0.1689	0.0812
(0.2, 0.5, 0.5)	0.0968	0.4163	0.1954	1.1117	0.455
(0.2, 0.5, 0.9)	0.0641	0.2277	0.0671	0.1382	0.1243
(0.2, 0.6, 0.7)	0.0534	0.1048	0.0715	0.3198	0.1373
(0.2, 0.7, 0.5)	0.0407	0.1643	0.0592	0.2184	0.1206
(0.2, 0.7, 0.8)	0.0184	0.0033	0.0054	0.0101	0.0093
(0.2, 0.7, 0.9)	0.0076	0.0005	0.0295	0.0252	0.0157
(0.2, 0.8, 0.7)	0.0107	0.0158	0.0144	0.012	0.0133
(0.2, 0.9, 0.5)	0.0217	0.0779	0.0176	0.0833	0.0501
(0.2, 0.9, 0.9)	0.0102	0.017	0.0142	0.0037	0.0113
(0.2, 1, 0.7)	0.029	0.0532	0.0262	0.0495	0.0395
(0.3, 0.5, 0.5)	0.1104	0.2033	0.1208	0.396	0.2076
(0.3, 0.5, 0.9)	0.0731	0.072	0.0934	0.1851	0.1059
(0.3, 0.6, 0.7)	0.0377	0.0979	0.0478	0.1078	0.0728
(0.3, 0.7, 0.5)	0.0268	0.105	0.0392	0.0908	0.0654
(0.3, 0.7, 0.9)	0.0161	0.0251	0.0221	0.1124	0.0439
(0.3, 0.8, 0.7)	0.0283	0.0453	0.0231	0.0435	0.035
(0.3, 0.9, 0.5)	0.0538	0.0752	0.0392	0.0877	0.064
(0.3, 0.9, 0.9)	0.0304	0.0612	0.0308	0.0663	0.0472
(0.3, 1, 0.7)	0.0554	0.1519	0.0525	0.1103	0.0925

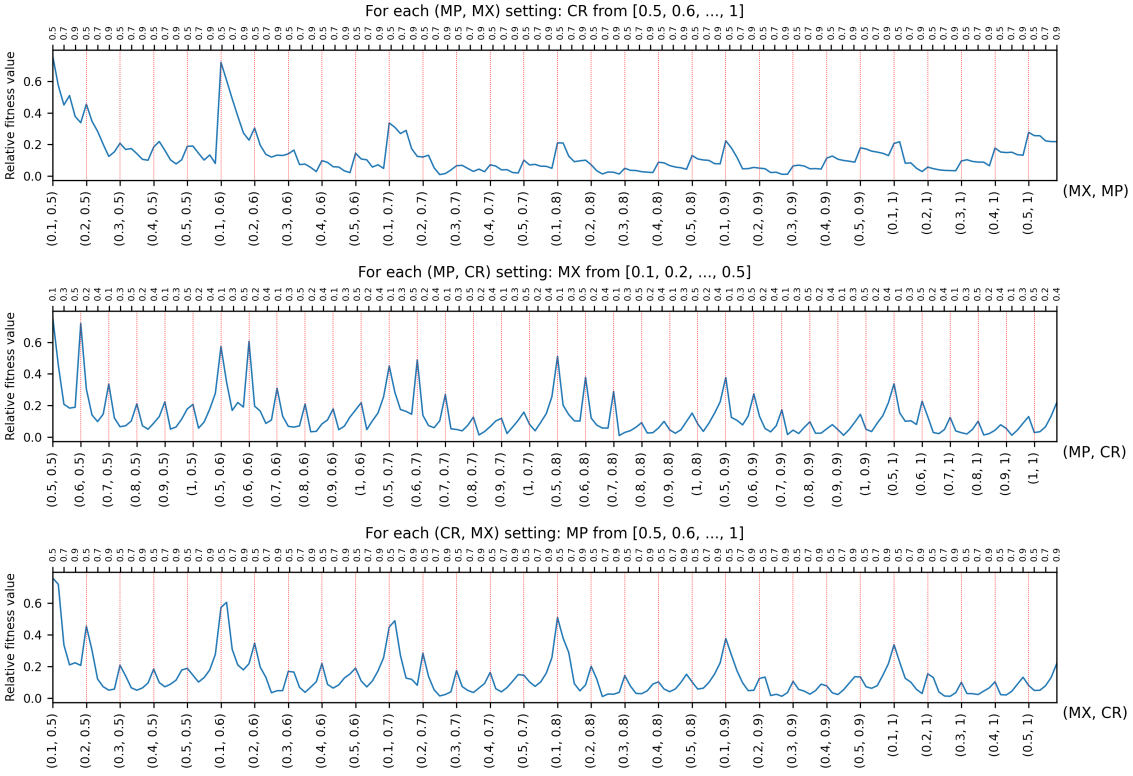
Table 6.4: Extract from the $RIiF$ values obtained during parameter tuning of GA1

Even if a single simulation of the SC takes only a split second (in our case ≈0.06s)[1], the parameter tuning process still requires several days of computing making parameter tuning extremely tedious. Before conducting further tests with the parameter choice $(MX = 0.2, MP = 0.7, CR = 0.8)$ for which the mean relative increase in $TSCC$ is minimal, we shall take a closer look at the results obtained by the pilot study and their implications. Starting by fixing two of the three available parameters and investigating the impact of a change in the third on the mean relative fitness value (as is depicted in Figure 6.1), we can make three interesting observations. Firstly, if MX and MP are fixed, the fitness value improves for higher $CR \in \{0.5, 0.6, ..., 1\}$, a trend that decreases

[1] All computations were performed on a cloud server running an Intel Xeon CPU E5-2650 v3 @ 2.30GHz with 12 cores and 48 GB RAM. Wherever seemed fit, we used multiprocessing to speed up the computation procedure.

as the settings for the mutation parameters increase in value. This effect can be observed when considering the mean (relative) fitness values in all four SC settings but is most pronounced in S1. We can therefore conclude, that crossing over more rather than fewer chromosomes is beneficial for the problem at hand and it is hence no surprise that the parameter setting determined by the pilot study sets $CR = 0.8$. Fixing MP and CR and iteratively incrementing $MX \in \{0.1, 0.2, ..., 0.5\}$, we find that for small value settings of MP and CR, the relative fitness improves as MX increases *ceteris paribus*. This effect gets weaker as values of MP and CR increase and is even reversed starting at medium sized MP and CR values from which point on higher MX settings lead to worse relative fitness all else being equal. Due to this directional change in the trend it is difficult to draw a universal conclusion which MX is best on its own for the given model, since the merit of any setting is strongly dependent on the other two parameters. Lastly, fixing CR and MX allows us to make a third observation which is analogous to the previous one: for low CR and MX, higher MP values improve relative fitness, the trend gets weaker as CR and MX grow until it reverses for medium sized parameter settings from which point on higher MP values worsen relative fitness. This analogous behaviour of MX and MP is not surprising, since both (although in different ways) determine the extent of mutation. To determine which MP setting is best, we therefore need to consider the remaining two parameters as well and cannot make a universal statement.

Figure 6.1: Sensitivity of RIiF to changes in parameters.



Using the best parameter setting for GA1 determined during the pilot study ($MX =$

$0.2, MP = 0.7, CR = 0.8$), we perform a reduced version of parameter tuning for the remaining three configurations of the algorithm as described earlier. Concretely, the parameter search space is limited to the neighbourhood of the tuned parameters of GA1, namely $MX \in \{0.1, 0.2, 0.3\}$, $MP \in \{0.6, 0.7, 0.8\}$, and $CR \in \{0.7, 0.8, 0.9\}$. We reduce the number of simulations to 20, and terminate the algorithm after 25 generations, limiting the required computation time to approximately $3^3 * 20 * 25 * 20 * 0.06 = 16200$ seconds (270 minutes) per SC setting per GA configuration[2]. To set the search radius SR for RS, we follow the same procedure as with the GA configurations, testing all search radii in $\{0.05, 0.1, 0.15, ..., 0.9, 0.95, 1\}$ in 20 simulations and 25 generations. The resulting optimal parameters for all algorithms are presented in Table 6.5.

GA1 (full parameter tuning)	$MX = 0.2, MP = 0.7, CR = 0.8$
GA2 (reduced parameter tuning)	$MX = 0.2, MP = 0.8, CR = 0.7$
GA3 (reduced parameter tuning)	$MX = 0.1, MP = 0.8, CR = 0.9$
RS (full parameter tuning)	$SR = 0.9$

Table 6.5: Optimal parameters according to the pilotstudy

6.2 Simulation Runs

All GA configurations and the RS are run 30 times with their respective optimal parameter settings for all four SC scenarios over 200 generations[3]. All other parameters remain as they were during the pilot study. The base-stock levels of the last generation corresponding to the best $TSCC^*$ as well as the mean $TSCC$ of all runs are presented in Table 6.6. The second row of each algorithm entry reports the corresponding standard deviation of all 30 simulations. The third row depicts the standard deviation divided by the best base-stock level or mean $TSCC$ respectively. The latter corresponds to the relative standard deviation of $TSCC$ obtained in the last generation. To evaluate the performance of the tested algorithms, we consider the mean $TSCC$ obtained in the last generation as opposed to the best $TSCC$, as the latter could be a product of lucky circumstances (e.g. below-average customer demand) rather than a sign of merit. This is especially important for GA3, which achieves slightly better $TSCC^*$ values than RS in the first three settings, however, its mean $TSCC$ is constantly significantly worse.
First of all, we would like to point out that it is possible that the results are slightly skewed in favour of GA1 as that is the only algorithm for which we tuned the parameters in full while the parameters of all other settings were only subjected to a limited pilot study. Although this is not ideal and GA2 as well as GA3 could end up performing

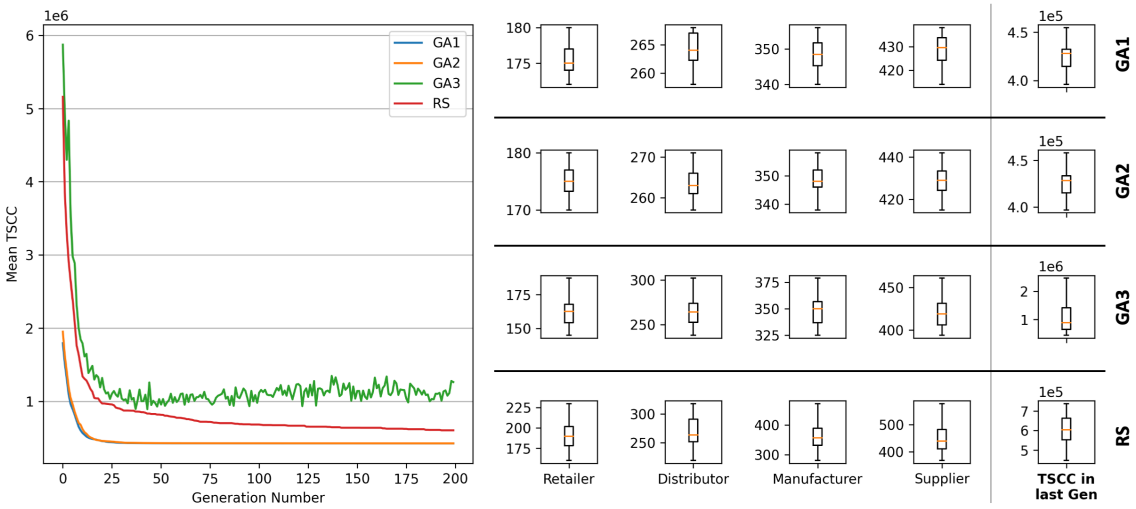
[2]Assuming that SC simulation takes ≈0.06s as was the case in our runs. The computation time connected to the GA itself is negligible.
[3]To allow reproduction of the results, we set a random seed of 123.

		R	D	M	S	TSCC*	Mean TSCC
S1	GA1	175	260	346	423	395929	425367
	σ	2.0114	2.1708	36.5392	22.7753	-	16047.1348
	$\sigma/\bullet$	0.0115	0.0083	0.1056	0.0538	-	0.0377
	GA2	173	261	345	430	397002	426031
		3.0977	3.7873	36.7617	35.2495	-	16125.5194
		0.0179	0.0145	0.1066	0.082	-	0.0379
	GA3	163	276	344	422	440190	1263470
		3.9967	4.2687	47.648	45.1241	-	928575.259
		0.0245	0.0155	0.1385	0.1069	-	0.7349
	RS	174	281	332	427	446010	604298
		5.8272	6.4688	55.7181	56.7155	-	75409.4059
		0.0335	0.023	0.1678	0.1328	-	0.1248
S2	GA1	213	544	1378	2268	765504	873095
		2.543	4.2984	44.5195	57.8346	-	59105.9794
		0.0119	0.0079	0.0323	0.0255	-	0.0677
	GA2	219	548	1392	2267	773346	876944
		6.5299	8.6137	26.5357	97.7979	-	58055.8089
		0.0298	0.0157	0.0191	0.0431	-	0.0662
	GA3	201	528	1453	2305	1019295	3589903
		19.2139	18.2623	121.3356	121.0262	-	2479471.37
		0.0956	0.0346	0.0835	0.0525	-	0.6907
	RS	216	503	1443	2297	1044619	1555885
		35.4987	32.0081	150.57	183.1285	-	281111.852
		0.1643	0.0636	0.1043	0.0797	-	0.1807
S3	GA1	174	259	332	423	916646	982283
		1.833	2.1817	14.3892	20.1653	-	36608.5925
		0.0105	0.0084	0.0433	0.0477	-	0.0373
	GA2	175	256	332	426	917904	983736
		2.8206	3.391	30.5782	27.1093	-	36298.2953
		0.0161	0.0132	0.0921	0.0636	-	0.0369
	GA3	189	241	336	429	1005380	2429768
		4.7151	4.4642	41.719	53.3317	-	1692080.97
		0.0249	0.0185	0.1242	0.1243	-	0.6964
	RS	165	262	334	519	1065616	1344630
		7.7423	8.6895	32.032	84.9229	-	175526.024
		0.0469	0.0332	0.0959	0.1636	-	0.1305
S4	GA1	220	535	1375	2276	1528952	1747023
		3.1489	3.7357	102.1239	50.6358	-	110076.631
		0.0143	0.007	0.0743	0.0222	-	0.063
	GA2	217	541	1369	2292	1530960	1753534
		6.4031	6.7314	129.9494	67.7522	-	107554.399
		0.0295	0.0124	0.0949	0.0296	-	0.0613
	GA3	205	544	1445	2434	2063341	6376449
		19.3248	19.5478	111.0942	139.8345	-	5067435.4
		0.0943	0.0359	0.0769	0.0575	-	0.7947
	RS	222	600	1338	2520	2060057	2995493
		44.2197	44.9901	113.2655	224.8436	-	491876.977
		0.1992	0.075	0.0847	0.0892	-	0.1642

Table 6.6: Simulation results. The first row of each algorithm depicts the value, the second row the standard deviation σ, and the third row the standard deviation divided by the value $\sigma/\bullet$.

better if tuned in full, we do not expect it to significantly influence the simulation results, since, as we pointed out before, the functionality of the three GA is generally very comparable and full parameter tuning would most likely yield the same parameter setting as the reduced study.

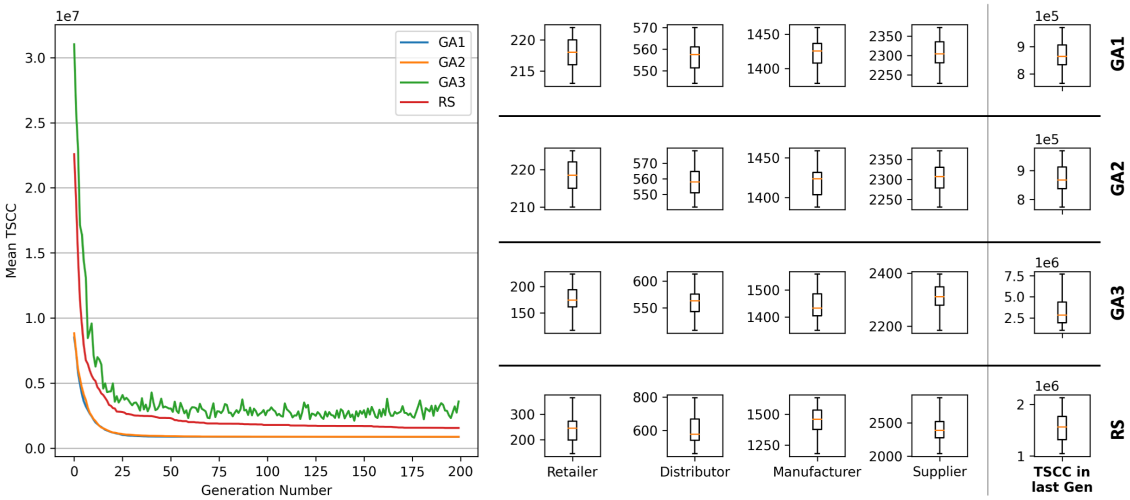
Figure 6.2: Plots for SC Setting S1. The lineplot on the left depicts the mean TSCC on 30 simulations per generation. The first four boxplots in each row correspond to the distribution of base-stock levels of each agent found by the algorithm in the last generation. The last boxplot in each row depicts the TSCC in the last generation.



Figures 6.2 to 6.5 show the main simulation results. The line plot on the left side of each graph visualises the mean TSCC in each generation of the respective algorithm. As they yield very similar results, the graphs of GA1 and GA2 can hardly be distinguished. For more detailed results of the two consider Table 6.6. The right side of the figures consists of several boxplots, where each row corresponds to an algorithm. The first four boxplots in each row show the base-stock levels per agent obtained in the last generation of each algorithm. Lastly, the boxplot furthest to the right in each row shows the distribution of obtained $TSCC$ values in the last generation. Note that in all boxplots, the horizontal orange line visualises the median, the lower and upper border of the box corresponds to the lower ($Q_{25\%}$) and upper quartile ($Q_{75\%}$) respectively, and the whiskers extend the two quartiles by one and a half times the interquartile range (IQR), i.e. they cover the interval $[Q_{25\%} - 1.5 * IQR, Q_{75\%} + 1.5 * IQR]$.

Let us first focus on what we can learn from the convergence of the algorithms before we move on to analyse the base-stock levels and the $TSCC$ in general. As the convergence properties turned out to be comparable in all four SC settings, all of the following observations are universal for all settings unless we state differently. Right away we can observe that GA1 and GA2 significantly outperformed GA3 and RS. What strikes the

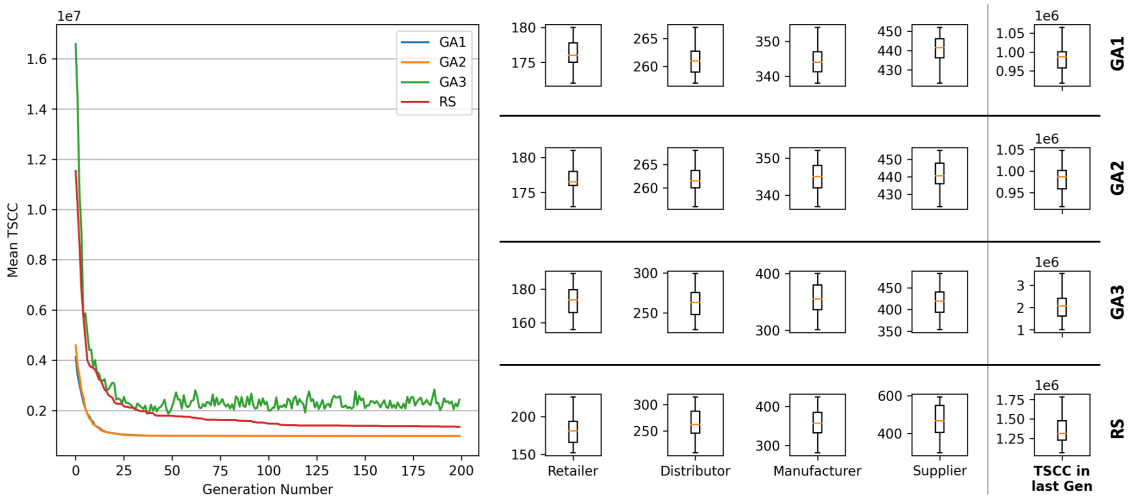
Figure 6.3: Plots for SC Setting S2.



reader immediately is the subpar performance of GA3, which could not even beat the random search process, which is due to the following. The configuration of GA3 makes use of roulette-wheel instead of elitist selection. While in theory, this should help with exploring the search space as all chromosomes have a nonzero selection probability, this obviously does not yield the desired results here. Rather than finding better solutions during exploration, roulette-wheel selection inhibits the convergence of the algorithm as there is no guarantee that the selection operator selects the best chromosomes for survival. We can therefore conclude that the more exploitative GA1 and GA2 do not tend to stagnate in local optima - or at least if they did (since we cannot evaluate whether a solution is a global optimum), those local optima are better than the ones obtained by the more explorative GA3. It seems to be the case that for selection, exploitation ought to be stressed over exploration. Mind you, we do not argue that this holds for all aspects of configuration since some degree of exploration is crucial in one form or another. We further observe, that despite its subpar performance, GA3 still achieves a similar initial rapid decrease in $TSCC$ as other configurations, which is due to the fact that in early populations, the difference in fitness values among individuals of a generation is still large. During the roulette wheel procedure, it is therefore highly likely that good chromosomes survive whereas weak ones die, leading to initial convergence. After 25 to 50 generations, this effect has worn off and the fitness values in one generation are much more homogenous, causing roulette wheel selection to resemble random selection, as most chromosomes are (approximately) equally likely to survive. This oscillatory behaviour in later generations can be observed in Figures 6.2 to 6.5 where GA3 is depicted as the green graph.

Next, let us take a closer look at the RS algorithm. Clearly, even for the vastly sim-

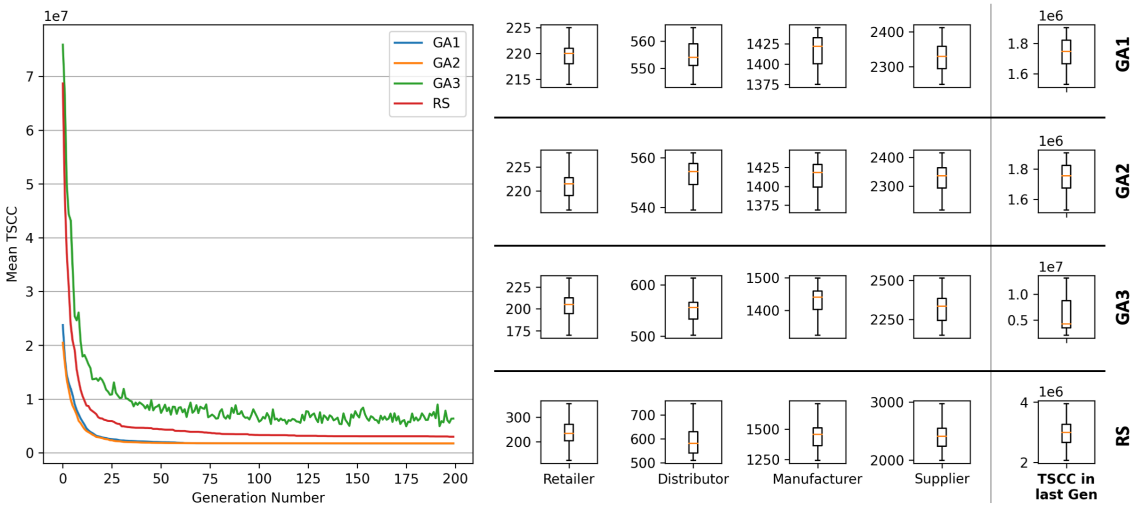
Figure 6.4: Plots for SC Setting S3.



pler algorithm, we can observe that it converges to a solution that in all four cases lies somewhere between the one obtained from GA1/GA2 and GA3. As the random search procedure only adopts a new solution if there is an improvement in the value of the objective function (i.e. lower $TSCC$), some degree of convergence is not surprising. We simply cannot guarantee that it converges to the global optimum as all solutions that would lead to improved fitness and therefore to a change in the candidate solution might lie outside the search radius at any iteration, causing RS to get stuck (see Zabinsky, 2003, p.16). The search radius of 0.9 leads RS to overcome initial bad solutions and quickly converge. After several iterations, the rate of convergence decreases as it becomes more difficult to find better solutions within the search radius. Were we to improve the performance of RS, it might be beneficial to adaptively vary the search radius and increase it if the rate of convergence falls below a specified threshold. However, since RS is meant as a simple benchmark to compare the performance of the GA we will refrain from delving deeper into this topic and conclude that GA1 and GA2 indeed yield significantly better results than RS. Nevertheless, it is important to remember that performance does not merely refer to a better fitness value but also the cost at which this increase in fitness comes, i.e. the required computation time. RS only considers a single candidate solution per generation as opposed to n (i.e. population size) candidate solutions in the GA case. Therefore, it requires only $1/n$ the number of calls of the evaluation function that a GA does in the same number of generations, thereby drastically reducing computation time. In light of the fact that GA1 and GA2 yield TSCC that are more than 25% lower than those obtained by RS in all SC settings, we are confident to state that the additional cost required for GA implementations is still worth it.

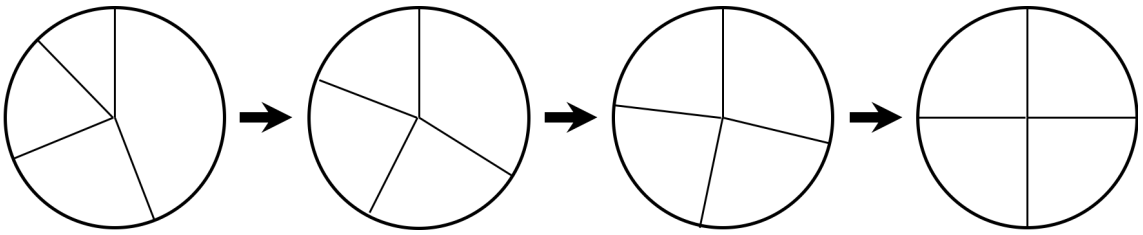
With regards to GA1 and GA2, we can instantly observe that their convergence and

Figure 6.5: Plots for SC Setting S4.



results were surprisingly similar. In large parts of the graphs it is impossible to distinguish between the two. The reason behind this apparent indifference in random and roulette wheel crossover in our application can be found when tracking the standard deviation of the size of the imaginary roulette wheel field of each candidate solution. In all generations, this standard deviation is very small implying that the imaginary fields are similarly sized and therefore almost equally likely to be selected for crossover. To be specific, the standard deviation in the first generation is approximately 0.02 which decreases quickly to only about 0.004 in the tenth generation. By the 200th generation, it has reduced to less than 0.00001. This phenomenon is conveyed in Figure 6.6 for a roulette wheel with four fields, that is a GA with a population size of four.

Figure 6.6: Convergence of roulette wheel field sizes to a uniform distribution.



While in the first roulette wheel, the different fields have different probabilities of being hit, the field sizes gradually converge until they produce a wheel with equally sized fields following a uniform random distribution. And since in our application the field sizes are relatively equal from the start, the roulette wheel crossover operator in GA1 essentially works like the random crossover operator in GA2. Of course, this is specific to our application and does not imply that roulette wheel crossover always acts in such a way.

It is furthermore striking that in none of the SC settings were there any significant improvements in TSCC after the 30^{th} generation, displaying the relatively fast convergence of GA1 and GA2. This implies that we could potentially reduce the simulation time by terminating the algorithm earlier, e.g. after 50 generations, cutting the required time by 75%. Certainly, this could decrease the solution quality by a small margin, yet in light of the substantial time saving, it could still be a worthwhile trade-off.
Moving on to the second part of our analysis, we can observe that both an increase in costs as well as an increase in lead times causes a surge in $TSCC$. However, the effect on the base-stock policy is heterogenous. Consider the base SC setting S1 where costs and lead times are relatively low and homogenous between agents. A more pronounced increase in lead times while keeping costs constant (S2) causes a sharp rise in base-stock levels at every echelon. Doing the opposite (i.e. a sharper increase in costs and keeping lead times homogenous as in S3) hardly affects the base-stock levels at all. This is indeed a very intriguing result as it would suggest that the policy is much more sensitive to changes in lead times than to changes in costs. After running further simulations, we can confirm that the effect on base-stock levels from an increase in costs was in fact negligible - but only as long as the increase did not (or at least not significantly) effect the cost structure itself, that is the costs at each agent relative to other agents. Not surprisingly, if we increase the holding cost disproportionately at one agent, the optimal policy suggests a lower base-stock level. However, it is still interesting that the policy remains unaffected by proportionate changes in costs. These results are independent of the algorithm employed. We can also observe, that the relative standard deviation of $TSCC$ (i.e. standard deviation of $TSCC$ in last generation divided by mean $TSCC$) of an algorithm is more or less the same for two SC settings that use the same lead time structure, thus for the pair S1/S3 and the pair S2/S4. For example, the relative standard deviation for GA1 in S1 is 0.0377 whereas for S3 it is 0.0373. Again, this relation holds for all tested algorithms. Looking more closely on how the policy in S1 evolves to the policy in S2 when lead times are longer, we find the following curious relationship. The increase in base-stock level per actor is approximately proportionate to the increase in minimum lead time of the same actor. For example, in S1, the manufacturer has a minimum LT of 8 and in S2 a minimum LT of 34 while the base-stock level (using GA1) in the former is 346 and in the latter 1378, thus $\frac{34}{8} \approx \frac{1378}{346}$. Note that this approximate relationship holds for the retailer, distributor, and supplier as well $\left(\frac{5}{4} \approx \frac{213}{175}, \frac{13}{6} \approx \frac{544}{260}, \frac{56}{10} \approx \frac{2268}{423}\right)$. We were able to confirm this in multiple preliminary tests where the GA was run on different LT settings and found the relation to hold in all cases. Most likely, this is connected with the change of the lower and upper bound (LB, UB) of the base-stock levels that accompanies the change in lead times as described in Section 5.2.1. A full-grown study into this finding and further empirical tests are out of the scope of this thesis, however, if this conjecture were to be universally

applicable, it would provide researchers and practitioners with a quick way of estimating the base-stock levels if a base scenario with known policy is available and only its lead times are altered.

From the boxplots on the right side of Figures 6.2 to 6.5 as well as from the standard deviations reported in Table 6.6, we can see that the standard deviation of GA3 far exceeds the one of GA1 and GA2, which should not be surprising in light of the graph of the algorithm and the described fluctuations that occur due to roulette wheel selection. With regards to the standard deviation relative to the base-stock levels that led to minimal $TSCC$s, we can say that in most cases the measure is moderate for GA configurations, almost always being under 10% and often even under 5%. For RS, the measure fluctuates slightly more, reaching values of up to 19.92% in S4 for the base-stock level of the retailer. The four boxplots to the left of each row in Figures 6.2 to 6.5 graphically undeline these findings and visualise the distribution of last-generation base-stock levels quite well. As stated before, the rate of convergence of the GA is reduced to almost zero after the 50^{th} generation, making it questionable whether it would be worth increasing the maximum generation in order to reduce the variance found in the base-stock levels. Presumably, in many cases the additionally required computation time is too costly to justify such an extension.

What can we conclude from the results and the analysis of the simulations? First and foremost, GA configurations based on elitist selection clearly outperform roulette wheel selection, the latter of which does not converge to a meaningful solution. The higher degree of exploration that comes with roulette wheel selection fails to provide satisfactory results. At the same time, it does not seem to make a difference whether roulette wheel crossover instead of random crossover is implemented, as in the application at hand the two are essentially equivalent. Furthermore, the rapid convergence of all considered algorithms offers the possibility of reducing the maximum number of generations, which would lead to a significant decrease in computation time, albeit at the small cost of a slightly larger $TSCC$. Last but not least, we can conclude that properly configured GA successfully and consistently outperform our benchmark, a random search process, which, while being less computationally greedy, could not provide a solution that can compete with the one obtained by the two best-performing GA configurations, GA1 and GA2.

7 Conclusion

In the sections leading up to this conclusion we explored a variety of aspects in the realm of inventory optimisation and genetic algorithms. We can draw conclusions from three different areas.

Firstly, with regards to the serial supply chain models, we explored the process involved in setting up such a model both mathematically as well as an algorithmic version that can be implemented programmatically and subsequently be used in simulation as a means to evaluate a given base-stock level. We would like to note that our model, while trying to get as close to a real-life situation as feasible in this thesis, for example, by allowing for both information as well as replenishment lead time, still makes a few assumptions that could inhibit its real-life applicability. To name just one assumption that could be relaxed, one could allow for stochastic lead times to simulate aspects such as unexpected delays during delivery.

Secondly, we delved deep into genetic algorithms, their theoretical background, possible applications and the implementation of such an algorithm. The relative ease with which one can set up a GA to solve combinatorial problems makes it an attractive method in many fields. Yet, the vast range of possible configurations that often come without general guidelines as to which method is best for a given application can complicate the implementation of a well-calibrated and specialised GA.

Thirdly, during the implementation of a GA for base-stock level optimisation, both merits and pitfalls related to GA became apparent. The arguably largest and most impactful problem turned out to be the tremendous computational effort required for full-fledged parameter tuning. Unfortunately, there is no good way around this without sacrificing some of the quality of the solution. The aforementioned problem of choosing the appropriate configuration (e.g. crossover operators etc.) poses an additional difficulty during implementation. Despite these challenges, two of the tested GA configurations proved to beat the benchmark, a random search process, by a significant margin, demonstrating that GA provide a promising method for base-stock level optimisation.

List of Tables

List of Figures

List of Algorithms

Bibliography

Ahmed, Mohamed A and Alkhamis, Talal M: Simulation-based optimization using simulated annealing with ranking and selection. In: *Computers & Operations Research*, Volume 29(4):pp. 387–402, 2002.

Arnold, JR Tony, Chapman, Stephen N and Clive, L: Introduction to materials management. In: *Prentice Hall*, 2008.

Bianchi, Leonora, Dorigo, Marco, Gambardella, Luca Maria and Gutjahr, Walter J: A survey on metaheuristics for stochastic combinatorial optimization. In: *Natural Computing*, Volume 8(2):pp. 239–287, 2009.

Chaharsooghi, S Kamal, Heydari, Jafar and Zegordi, S Hessameddin: A reinforcement learning model for supply chain ordering management: An application to the beer game. In: *Decision Support Systems*, Volume 45(4):pp. 949–959, 2008.

Christopher, Martin: Logistics & supply chain management: Creating value-adding networks (financial times series). 2005.

Chung, Chia-Shin, Flynn, James and Stalinski, Piotr: A single-period inventory placement problem for a serial supply chain. In: *Naval Research Logistics (NRL)*, Volume 48(6):pp. 506–517, 2001.

Clark, Andrew J: The use of simulation to evaluate a multiechelon, dynamic inventory model. In: *Naval Research Logistics Quarterly*, Volume 7(4):pp. 429–445, 1960.

Clark, Andrew J and Scarf, Herbert: Optimal policies for a multi-echelon inventory problem. In: *Management science*, Volume 6(4):pp. 475–490, 1960.

Daniel, J Sudhir Ryan and Rajendran, Chandrasekharan: Determination of base-stock levels in a serial supply chain: a simulation-based simulated annealing heuristic. In: *International Journal of Logistics Systems and Management*, Volume 1(2-3):pp. 149–186, 2005a.

Daniel, J Sudhir Ryan and Rajendran, Chandrasekharan: A simulation-based genetic algorithm for inventory optimization in a serial supply chain. In: *International Transactions in Operational Research*, Volume 12(1):pp. 101–127, 2005b.

Darwin, Charles and Bynum, William F: *The origin of species by means of natural selection: or, the preservation of favored races in the struggle for life*. AL Burt New York, 2009.

Davis, Lawrence: *Handbook of genetic algorithms*. CumInCAD, 1991.

Eiben, Agoston E, Aarts, Emile HL and Van Hee, Kees M: Global convergence of genetic algorithms: A markov chain analysis. In: *International Conference on Parallel Problem Solving from Nature*. Springer, 1990, pp. 3–12.

Eiben, Agoston E, Smith, James E et al.: *Introduction to evolutionary computing*. Springer, 2003.

Ettl, Markus, Feigin, Gerald E, Lin, Grace Y and Yao, David D: A supply network model with base-stock control and service requirements. In: *Operations Research*, Volume 48(2):pp. 216–232, 2000.

Federgruen, Awi and Zipkin, Paul: Computational issues in an infinite-horizon, multiechelon inventory model. In: *Operations Research*, Volume 32(4):pp. 818–836, 1984.

Gallego, Guillermo and Zipkin, Paul: Stock positioning and performance estimation in serial production-transportation systems. In: *Manufacturing & Service Operations Management*, Volume 1(1):pp. 77–88, 1999.

Gaughan, Patrick A: *Maximizing corporate value through mergers and acquisitions: A strategic growth guide*. John Wiley & Sons, 2013.

Glasserman, Paul and Tayur, Sridhar: Sensitivity analysis for base-stock levels in multiechelon production-inventory systems. In: *Management Science*, Volume 41(2):pp. 263–281, 1995.

Holland, John Henry et al.: *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

Keller, Mitchel T and Trotter, William T: *Applied Combinatorics*. 2017.

Kimbrough, Steven O, Wu, Dong-Jun and Zhong, Fang: Computers play the beer game: can artificial agents manage supply chains? In: *Decision support systems*, Volume 33(3):pp. 323–333, 2002.

Kramer, Oliver: *Genetic algorithm essentials*, Volume 679. Springer, 2017.

Kroese, Dirk P, Taimre, Thomas and Botev, Zdravko I: *Handbook of monte carlo methods*, Volume 706. John Wiley & Sons, 2013.

Lee, Hau L and Billington, Corey: Material management in decentralized supply chains. In: *Operations research*, Volume 41(5):pp. 835–847, 1993.

Mitchell, Melanie: *An introduction to genetic algorithms*. MIT press, 1998.

Moon, Chiung, Kim, Jongsoo, Choi, Gyunghyun and Seo, Yoonho: An efficient genetic algorithm for the traveling salesman problem with precedence constraints. In: *European Journal of Operational Research*, Volume 140(3):pp. 606–617, 2002.

Mori, Masao and Tseng, Ching Chih: A genetic algorithm for multi-mode resource constrained project scheduling problem. In: *European Journal of Operational Research*, Volume 100(1):pp. 134–141, 1997.

Parsons, Rebecca J, Forrest, Stephanie and Burks, Christian: Genetic algorithms, operators, and dna fragment assembly. In: *Machine Learning*, Volume 21(1-2):pp. 11–33, 1995.

Petrovic, Dobrila, Roy, Rajat and Petrovic, Radivoj: Modelling and simulation of a supply chain in an uncertain environment. In: *European journal of operational research*, Volume 109(2):pp. 299–309, 1998.

Porteus, Evan L: The newsvendor problem. In: *Building Intuition*, Springer, pp. 115–134. 2008.

Rao, Uday, Scheller-Wolf, Alan and Tayur, Sridhar: Development of a rapid-response supply chain at caterpillar. In: *Operations Research*, Volume 48(2):pp. 189–204, 2000.

Reeves, Colin and Rowe, Jonathan E: *Genetic algorithms: principles and perspectives: a guide to GA theory*, Volume 20. Springer Science & Business Media, 2002.

Reeves, Colin R: Genetic algorithms for the operations researcher. In: *INFORMS journal on computing*, Volume 9(3):pp. 231–250, 1997.

Schmidt, Günter and Wilhelm, Wilbert E: Strategic, tactical and operational decisions in multi-national logistics networks: a review and discussion of modelling issues. In: *International Journal of Production Research*, Volume 38(7):pp. 1501–1523, 2000.

Shang, Kevin H and Song, Jing-Sheng: Newsvendor bounds and heuristic for optimal policies in serial supply chains. In: *Management Science*, Volume 49(5):pp. 618–638, 2003.

Singh, Gulshan and Deb, Kalyanmoy: Comparison of multi-modal optimization algorithms based on evolutionary algorithms. In: *Proceedings of the 8th annual conference on Genetic and evolutionary computation*. 2006, pp. 1305–1312.

Towill, Denis R, Naim, Mohamed M and Wikner, Joakim: Industrial dynamics simulation models in the design of supply chains. In: *International Journal of Physical Distribution & Logistics Management*, 1992.

Wolpert, David H and Macready, William G: No free lunch theorems for optimization. In: *IEEE transactions on evolutionary computation*, Volume 1(1):pp. 67–82, 1997.

Wolpert, David H, Macready, William G et al.: No free lunch theorems for search. Technical Report, Technical Report SFI-TR-95-02-010, Santa Fe Institute, 1995.

Zabinsky, Zelda B: Pure random search and pure adaptive search. In: *Stochastic Adaptive Search for Global Optimization*, Springer, pp. 25–54. 2003.

Appendices

Mathematical Model for TSCC-Minimisation in a Serial Supply Chain

$TSCC$: total supply chain cost

i: installation index, i.e. agent index if $i \in \{1, ..., N\}$, customer if $i = 0$, RMS if $i = N+1$

N: number of agents in the supply chain

s_i: base-stock level at agent i

h_i: per period, per unit holding cost rate at agent i

b_i: per period, per unit shortage cost rate at agent i

ILT_i: information lead time at agent i if $i \in \{1, ?, N\}$ or at RMS if $i = N + 1$

RLT_i: replenishment lead time at agent i

$I_{i,t}$: on-hand inventory at agent i at the end of time t

$I^*_{i,t}$: on-hand inventory at agent i at the beginning of time t (after replenishment from agent $i + 1$ has arrived)

$OI_{i,t}$: on-order inventory at agent i at the end of time t

$OI^*_{i,t}$: on-order inventory at agent i at the beginning of time t

$B_{i,t}$: backlog at agent i at the end of time t

$D_{i-1,i,t-ILT_i}$: demand placed by $i - 1$ to upstream agent i at time $t - ILT_i$

$D^*_{i-1,i,t-ILT_i}$: dummy variable introduced to satisfy $D_{i1,i,t-ILT_i} \leq 0$

$QS_{i+1,i,t-RLT_i}$: quantity shipped from agent $i + 1$ to i at $t - RLT_i$

$$\text{Minimise TSCC} = \min \sum_{t=1}^{T} \sum_{i=1}^{N} (h_i I_{i,t} + b_i B_{i,t}) \tag{.1}$$

$$OI^*_{i,t} = OI_{i,t-1} - QS_{i+1,i,t-RLT_i} \tag{.2}$$

$$I^*_{i,t} = I_{i,t-1} + QS_{i+1,i,t-RLT_i} \tag{.3}$$

$$I_{i,t} = I^*_{i,t} - (D_{i-1,i,t-ILT_i} + B_{i,t-1} - B_{i,t}) \tag{.4}$$

$$QS_{i,i-1,t} = I^*_{i,t} - I_{i,t} \tag{.5}$$

$$D_{i,i+1,t} - D^*_{i,i+1,t} = s_i + B_{i,t} - I_{i,t} - OI^*_{i,t} \tag{.6}$$

$$OI_{i,t} = OI^*_{i,t} + D_{i,i+1,t} \tag{.7}$$

$$QS_{N+1,N,t} = D_{N,N+1,t-ILT_{N+1}} \tag{.8}$$

$$OI_{i,0} = 0 \tag{.9}$$

$$B_{i,0} = 0 \tag{.10}$$

$$I_{i,0} = s_i \tag{.11}$$

$$QS_{i+i,i,\tau} = 0, \forall \tau \leq 0 \& \forall i \leq N \tag{.12}$$

$$LB_i \leq s_i \leq UB_i \tag{.13}$$

$$OI_{i,t}, OI^*_{i,t}, I_{i,t}, I^*_{i,t}, QS_{i+1,i,t}, D_{i+1,t}, D^*_{i+1,t}, B_{i,t} \geq 0, \forall i \leq N \& \forall t \leq T \tag{.14}$$

Algorithm 1: Generic Genetic Algorithm

MR : mutation Rate
CR : crossover Rate
n : population size
max_gen: number of chromosomes to be created
par_pop : parent population
int_pop : intermediate population

```
1  Function crossover(pop_in, CR):
2  |  pop_out := crossover chromosomes in pop_in with probability CR;
3  |  return pop_out;
4  Function mutation(pop_in, MR):
5  |  pop_out := mutate chromosomes in pop_in with probability MR;
6  |  return pop_out;
7  Function evaluation(pop_in):
8  |  call fitnes function on each chromosomes in pop_in and save result to each;
9  Function genoPhenoMapping(pop_in):
10 |  pop_out := encode genotype chromosomes in pop_in to phenotype;
11 |  return pop_out;
12 Function selection(pop_in1, pop_in2):
13 |  pop_out := select chromosomes from pop_in1 and pop_in2 according to
   |   selection rule;
14 |  return pop_out;
15 Initialise par_pop of n random chromosomes;
16 for i := 1 to max_gen by 1 do
17 |  int_pop := crossover(par_pop, CR);
18 |  int_pop := mutation(int_pop, MR);
19 |  if genotype is not phenotype then
20 |  |  evaluation(genoPhenoMapping(int_pop));
21 |  else
22 |  |  evaluation(int_pop);
23 |  end
24 |  par_pop :=selection(int_pop, par_pop)
25 end
26 Best chromosome in par_pop contains the optimisation problem solution;
```

Algorithm 2: Serial Supply Chain Model

$OI_{i,0} = 0 \, \forall i$;
$B_{i,0} = 0 \, \forall i$;
$I_{i,0} = s_i \, \forall i$;
$QS_{i+1,i,\tau} = 0 \, \forall \tau \leq 0 \, \& \, \forall i \leq N$;
for $t := 1$ **to** T **by** 1 **do**
    Carry over backlog to next period $B_{i,t} := B_{i,t-1} \, \forall i$;
    for $i := 1$ **to** N **by** 1 **do**
        $I^*_{i,t} = I_{i,t-1} + QS_{i+1,i,t-RLT_i}$;
        $OI^*_{i,t} = OI_{i,t-1} - QS_{i+1,i,t-RLT_i}$;
        if $B_{i,t} > 0$ **then**
            Set $newbacklog := max(B_{i,t} - I^*_{i,t}, 0)$ and add backlog shipment to total shipment $QS_{i,i-1,t}$ += $B_{i,t} - newbacklog$;
            Update inventory $I^*_{i,t} := max(I^*_{i,t} - B_{i,t}, 0)$ and backlog $B_{i,t} := newbacklog$;
        end
        if *i is* 1 **then**
            Demand $D_{0,1,t-ILT_1}$ from customer is received;
        else
            Demand $D_{i-1,i,t-ILT_i}$ from the downstream agent is received;
        end
        Add regular shipment $regship = min(D_{i-1,i,t-ILT_i}, I^*_{i,t})$ to total shipment to downstream $QS_{i,i-1,t}$ += $regship$;
        Update inventory $I_{i,t} = I^*_{i,t} - regship$ and backlog $B_{i,t}$ += $D_{i-1,i,t-ILT_i} - regship$;
        if *i is not* 4 **then**
            Set order quantity $D_{i,i+1,t} := max(s_i + B_{i,t} - I_{i,t} - OI^*_{i,t}, 0)$;
        else
            Set order quantity $D_{4,5,t} := max(s_4 + B_{4,t} - I_{4,t} - OI^*_{4,t}, 0)$ to RMS and future shipment $QS_{5,4,t+ILT_{RMS}} := D_{4,5,t}$;
        end
        Update inventory $OI_{i,t} := OI^*_{i,t} + D_{i,i+1,t}$;
        Calculate period holding cost $h_i I_{i,t}$ and period shortage cost $b_i B_{i,t}$;
    end
end
Calculate and return $TSCC = \Sigma_{t=1}^{T} \Sigma_{i=1}^{N} (h_i I_{i,t} + b_i B_{i,t})$;

Algorithm 3: Genetic Algorithm for Base-Stock Level Optimisation

```
Function crossover(pop_in, CR):
    /* Performs random crossover                                    */
    pop_pool := shuffle chromosomes in pop_in randomly;
    u := random number from ~ U(0,1);
    for i := 1 to ⌈n/2⌉ by 1 do
        if u ≤ CR then
            cross pop_pool[i*2] and pop_pool[i*2+1] at random point and add
             result to pop_out;
        else
            add pop_pool[i*2] and pop_pool[i*2+1] uncrossed to pop_out;
        end
    end
    return pop_out;
Function mutation(pop_in, MR):
    /* Performs mutation on intermediate population                 */
    for chrom in pop_in do
        for gene in chrom do
            u := random number from ~ U(0,1);
            if u ≤ MP then
                newgene := gene*(1-MX) + gene*2*MX*u;
            else
                newgene := gene;
            end
            Add newgene to newchromosome;
        end
        Add newchromosome to pop_out;
    end
    return pop_out;
Function evaluation(pop_in):
    Run SC model as fitness function on each chromosome in pop_in and add
     result as attribute to each;
Function selection(pop_in1, pop_in2):
    /* Perform elitist selection                                    */
    sorted_pool := sort pop_in1 ∪ pop_in2 by decreasing fitness value;
    pop_out := first n chromosomes from sorted_pool;
    return pop_out;

Initialise par_pop of n random chromosomes within LB and UB;
Evaluate chromosomes in par_pop;
for i := 1 to max_gen by 1 do
    int_pop := crossover(par_pop, CR);
    int_pop := mutation(int_pop, MR);
    evaluation(int_pop);
    par_pop :=selection(int_pop, par_pop)
end
Best chromosome in par_pop contains the optimisation problem solution;
```

Algorithm 4: Roulette Wheel Selection Operator

```
1  Function rouletteSelection(pop_in1, optional: pop_in2):
       /* Perform roulette wheel selection                              */
2      for i in pop_in1 ∪ pop_in2 do
3          Calculate relative fitness value of chromosome p_i = f_i / Σ_{j=1}^{N} f_j;
4          Calculate CumP_i = Σ_{j=1}^{i} p_j;
5      end
6      while pop_out contains less than n elements do
7          u := random number from ~ U(0,1);
8          Select first chromosome k from pop_in for which cP_k ≥ u;
9          if k not yet in pop_out then
10             Add k to pop_out
11         end
12     end
13     return pop_out;
```

Algorithm 5: Roulette Wheel Crossover Operator

```
1  Function rouletteCrossover(pop_in, CR):
       /* Performs crossover using roulette wheel selection to pair
          chromosomes                                                   */
2      pop_pool := rouletteSelection(pop_in);
3      u := random number from ~ U(0,1);
4      for i := 1 to ⌈n/2⌉ by 1 do
5          if u ≤ CR then
6              cross pop_pool[i*2] and pop_pool[i*2+1] at random point and add
                result to pop_out;
7          else
8              add pop_pool[i*2] and pop_pool[i*2+1] uncrossed to pop_out;
9          end
10     end
11     return pop_out;
```