Chapter 1
Global Optimization in Supply Chain Operations

Maksud Ibrahimov, Arvind Mohais, Zbigniew Michalewicz

Abstract
This chapter discusses some optimization issues from a business perspective in the context of the supply chain operations. We note that the term “global optimization” may have different meanings in different communities and we look at it from the business and classical optimization points of view. We present two real-world optimization problems which differ in scope and use them for our discussion on global optimization issues. The differences between these two problems, experimental results, the main challenges they present and the algorithms used are discussed. Here, we claim neither uniqueness nor superiority of the algorithms used, rather the main goal of this chapter is to emphasize the importance of the global optimization concept.

1.1 Introduction

Many aspects of commercial and industrial processes are subject to optimization, and this fact has been the source of many applications of modern heuristic methods. For example, the sequencing of jobs can be optimized in the classical job shop scheduling problem [2, 14, 4], there are many algorithms that address different kinds of planning and cutting problems [8, 7], many methods were developed to solve different types of transportation issues [11], as well as allocation and distribution problems [3, 17]. These are all examples of problems in which a globally optimal solution is sought for a specific and fairly narrow and focused problem. In real world business situations, however, it is often the case that the potential for optimizing operations and profits can be increased by considering broader subsets of the business, and by looking at how major processes affect each other, in addition to the traditional approach of seeking independent opportunities for improvement. For example, in the case of job shop scheduling, one might also look at the logis-
tics of the provision of raw materials for the shop, as well as the details of post-production inventory shipping, marketing and retailing. From a business perspective, this is global optimization.

Lawrence Davis, one of the experts in supply chain optimization, describes this situation as follows [9].

“...companies should be able to use solutions to optimize across their sourcing and procurement, production and distribution processes all at the same time. Instead, supply chain solutions tend to break those functions out into separate modules, each of which runs separately from the others. The answer you get when you first do production and then do distribution or any of these functions independently is not as good as the answer you get when you do them together.”

Consider an example in which a company produces and distributes certain types of goods. Suppose that the company’s factory is very efficient and, in a given month, produces an exceptionally large quantity of finished goods. From a local point of view this component of the supply-chain operation (i.e. the factory) performed excellently. However, if we look at the broader picture of global production, if the excess goods are not readily distributed and consumed, the company will need to use storage space which will cost extra money, thus making this locally good production run sub-optimal from a global point of view.

It is possible to take the idea of global optimization from a business perspective to increasingly greater heights by including into consideration more and more factors that affect a business’ profitability. However, practical limits are often encountered when the scope of the optimization problem begins to encroach on boundaries of another business entity’s operations. For example, it may be that the provision of raw materials to a job shop is a matter handled by a different business entity (i.e. the component) who may not be willing to divulge minute details of their operations.

In this chapter, we will look at characteristics of global optimization from a business perspective, and use two real-world business examples that we have solved to illustrate the distinction between a relatively simple situation that focuses on one aspect of a business’ operations, and another, in which several components are considered simultaneously. The business domains of each example will be presented in depth, along with details of the evolutionary algorithms that were used to solve the problems. The data for each example was collected from actual businesses, but has been altered systematically to protect the privacy of the entities involved. These examples will allow the reader to appreciate the use of nature-inspired techniques in solving the problems. Our broader goal, however, remains to highlight the differences between the two types of problems, and how those differences might influence the solutions to the problems using evolutionary algorithms.

When we make the distinction between local and global optimization from a business perspective, the idea of global optimization in the classical sense still holds in both cases. If we try to optimize over the entire supply chain, then in terms of the objective function being considered, we still seek a clas-
sical global optimum. On the other hand, if we restrict our attention to optimizing the operations of a single factory, then this is local optimization with respect to the broader picture of the supply chain; but nevertheless we also seek a classical global optimum for the objective function associated with the optimization of the factory. The distinction between local and global optimization from a business perspective lies in the differences of their respective search spaces.

The general problem of global optimization, in the classical sense, is to find $x$ so as to

$$\text{optimize} \ f(x) , \ x = (x_1, \ldots, x_n) \in F \subseteq S ,$$

where $S$ defines the search space and $F$ defines feasible region which is bounded by the set of $m$ constraints: $g_j(x), \forall j = 1, \ldots, m$.

That is, one is seeking to find an element in the search space $S$ that optimizes the objective function.

Usually, the search space $S$ is quite simple: for numerical optimization it is $S \subseteq \mathbb{R}^n$, and for combinatorial optimization problems, $S$ is discrete in nature, like a set of permutations, or a set of edges of a graph. An example of a combinatorial optimization problem that one might encounter in a commercial setting is finding a permutation of $n$ jobs to be executed on a machine that would minimize the amount of time spent re-configuring the machine between jobs (or from another perspective, maximizes the amount of time the machine is in use).

In silo\(^1\)-focused optimization, the search space $S$ is typically built from a single type of variable. For example, as above, $S$ may consist of all permutations of the set $\{1, 2, \ldots, n\}$. Business global optimization problems, on the other hand, typically have a more complex search space that consists of compositions of simpler search spaces. We will refer to these types as single component and multiple component search spaces respectively.

Some typical characteristics of a problem that suggest that there might be an opportunity for global optimization from a business perspective are the following:

- **Dependence or partial dependence among components.** Problem specific constraints and business rules tie single components together creating dependencies among them. Note that constraints here can be intracomponent constraints, which apply to a single component itself and intercomponent constraints, that actually make these dependencies. Intracomponent constraints are the classic type of constraints. Intercomponent constraints have a slightly different purpose in global optimization compared to the classic one.

- **Multiple objectives.** Having many single components in the problem, each component having its own objective or even several objectives, yields a

\(^1\) In supply chain business terminology, the word “silos” is used to denote a single component of the business.
multiple objective optimization problem. From a business perspective, global optimization problems are multi-objective problems, but they are distinguished by the fact that the objectives are related in a hierarchical manner with some objectives related to single silos, and other higher-level objectives being compositions of lower-level objectives.

- Need to understand complex what-if scenarios. Businesses and their competitors are rarely static. Instead, they are dynamic and the business environments are constantly changing. Thus, companies want to make some decisions and changes to their operations. There is a large need for businesses to foresee how a particular change or decision will impact their priorities, profit and other objectives. Complex multi-component systems with a large network of dependencies are a lot harder to understand and predict than a single component system.

Examples in this chapter concentrate mostly on the first point (i.e., Dependence or partial dependence among components). Although one of the problems has multiple objectives, we do not treat our problems as such, as the main purpose of this chapter is to underline the importance of the concept of global optimization from a business perspective. The third point (i.e., what-if scenarios) is not considered in this chapter.

It is important to understand that the term “global optimization”, as used here, is not a method to solve supply chain management problems but rather a concept of viewing multi-silo supply chain problems from a broader perspective.

In recent years, there has been an increased interest in solving supply chain management problems using evolutionary algorithms. David Naso et al. [13] look at the problem of coordination of just-in-time production and transportation in a network of partially independent facilities of ready-mixed concrete. They optimize the network of independent and distributed production centers serving a number of customers distributed across a certain geographical area. This problem, which has high complexity and strict time delivery constraints, is approached with a meta-heuristic based on a hybrid genetic algorithm with combined constructive heuristics.

Altiparmak et al. [1] proposes algorithms using mixed-integer, non-linear programming model for multi-objective optimization of a supply chain network based on the real world problem of a company that produces plastic products. They compare three approaches to find the set of Pareto-optimal solutions and discuss the pros and cons of each of them.


In Vergara et al. [15], an evolutionary algorithm was developed for dealing with the coordination of supply chain flows between different members of the chain. The authors of that paper recognized the importance of an overarching algorithm that optimizes the whole system. Their work looked at the flow of
materials in terms of supply and demand, but did not consider the intricacies of production within each business silo.

Even much earlier, researchers had been working on this type of problem as can be seen in the following references. Lee and Choi [5] apply genetic algorithms to solve a single machine scheduling problem with distinct dates and attempt to minimize all penalties. This method produces near optimal solutions, which they prove by comparison with an exact algorithm. Lee et al. [6] address the problem of inventory management of a refinery that imports several types of crude oil and proposes a mixed-integer linear programming model. Martin et al. [10] create a linear programming model to optimize flat glass production.

The rest of the chapter is organized as follows: the next two sections introduce two real-world optimization problems, their difficulties, algorithms that solve them, and the experiments conducted. Section 4 discusses the differences and similarities between these two problems. Section 5 concludes the chapter.

1.2 Optimization of the Real-world Glass Company Operations

In this section, we present a real world example of the glass item production optimization problem, its proposed solution, and the experimental results.

1.2.1 Problem Statement

Here, we consider a real world example of an optimization problem that focuses on a single silo of a business' operation. Assume that ‘Max Glass’ is a company that produces glass products. It uses machines called powerpacks to produce molten glass from raw materials, and other moulding machines to make moulds and uses the molten glass to make finished products. This process is illustrated in Figure 1.1.

The glass-producing factory uses several types of glasses to make a variety of finished products. Any powerpack may be used to prepare any type of molten glass. However, when changing between glass types on the same powerpack, a cleaning operation is needed. The extent of this cleaning procedure varies. Sometimes melting of new materials can start without any cleaning, but sometimes a careful and thorough procedure is required when switching between materials. This operation consumes time and money, and should be avoided as far as possible.

The moulding machines, also called production lines, are not all the same. Some products can be produced only on one machine, whereas others can be
produced on more than one. In the latter case, however, production rates may differ due to different efficiency rates. Production lines use special implements called tools to make moulds, and naturally different products require different tools. As before, time and money are consumed when there is a need to change tools.

Other business rules and constraints that characterize the problem are the following:

- Powerpacks and production lines can only operate at certain times of the day and at certain days of the week.
- Powerpacks operate by batch loads. This means that they produce a specific tonnage of molten glass in a batch, and that smaller amounts are not permitted. So, for example, assuming a batch size of 8 tonnes, if we need to have just 5 tonnes of material for a particular job, we would be forced to produce 8 tonnes and there would be 3 tonnes of wastage.
- No production run is perfect and some products can be defective. To take this issue into account, a small percentage of surplus is usually planned.
- Certain different products cannot be produced around the same time because they are too similar in appearance, and since all products end up on a common polishing line, different products that look very similar cannot be easily distinguished.

### 1.2.2 Customers’ Orders

For a given period of time, say one week, there are a number of customer orders that need to be produced. Suppose that for such a period of time, we have $n$ orders placed by $n$ customers $C_1, C_2, \ldots, C_n$. Each customer’s order consists of a set of ordered items, with associated quantities for each. Assuming that the $m$ possible products that can be produced are $I_1, I_2, \ldots, I_m$, table 1.1 shows what the list of orders might look like. In Table 1.1, each cell is called an order item (OI).
Given the above description of the application domain, the problem can now be stated as follows:

Given a set of order items \( \{OI_1, OI_2, \ldots, OI_k\} \) to be completed during a specific time frame, and a set of production lines \( \{P_1, P_2, \ldots, P_l\} \), allocate each order item to a particular production line in such a way as to have a feasible schedule in which as many order items are completed as possible, and the production costs are minimized.

### 1.2.3 Solution

Figure 1.2 shows a screen of a system that was developed to solve the sequencing problem described above. It shows three production lines called Machine #1, Machine #2 and Machine #3, and two power packs called Supplier #1 and Supplier #2 for melting glass. In this figure, each order item is represented by a rectangle and each glass type is represented by a different color. Some orders have been scheduled for the production and some are still unassigned. In the middle section of the figure is a table that shows the details of all the orders, and the bottom part of the figure shows further details of a single selected order, such as the production lines where can it be produced, the rate of production, the tools required and the quantity of finished products needed.

### 1.2.4 Representation

The following version of evolutionary algorithm was used for this problem.

**Solution Representation**

We represent our solution indirectly in the following way

\[
X = (x_1, \ldots, x_v)
\]  

(1.2)

where each \( x_i \) is a pair \((OI_k, P_l)\), \(OI_k \in OI, P_l \in P\). The pair \((OI_k, P_l)\) means that order item \(OI_k\) will be executed on machine \(P_l\).
Fig. 1.2 The application’s main screen with production lines (A), powerpacks (B), ordered items (D). Here, grayed areas represent hours when a particular machine cannot work (C). An ordered item’s selected properties are shown in the bottom box (E). (F) represents the load of a particular powerpack at particular time.

After the population of individuals has been initialized, the evolutionary algorithm enters the main loop where the population is evolved until a specific criterion is satisfied. Each individual is assigned a fitness value by an evaluation function which is designed so that its value is indicative of how well the individual solves the problem. The evaluation functions, in our work, makes use of penalty functions which essentially assign a value to an individual in proportion to how badly the individual violates some constraint.

During the main loop, individuals with higher fitness are probabilistically given more chances to reproduce, that is, to make modified copies of themselves to replace them if modified copies have better fitness values. The changes are made to an individual by means of operators, which are functions that take one or more individuals and combine, alter and manipulate their structures in a partially random way. After operators have been applied, some of the resulting new individuals are passed through repair functions, that attempt to randomly, but intelligently alter their structure so as to reduce the extent of their constraint violations.

Decoders
From the description of the solution structure, the order in which these jobs will be executed is not clear. This decision is the job of the decoders. It works in the following way. The ordering of tuple \( x \) is taken as a priority ordering. Starting with a blank state, i.e. with all machines marked as completely unoccupied, \( x_1 \) is given first priority to find a place on the machine.
Suppose that $x_1 = (OI_{k_1}, P_{l_1})$, then the decoder looks at the timetables of machine $P_{l_1}$ and finds the first suitable time slot in which to place ordered item $OI_{k_1}$. If there is no possible time slot in machine $P_{l_1}$, then other machines are considered. Next the decoder processes $x_2$, and so on until the last element.

**Operators**

- **Swap order item operator:** When this operator is applied to an individual, it randomly picks two elements, for example $x_a = (x_a.OI, x_a.P)$ and $x_b = (x_b.OI, x_b.P)$, and swaps the order items in each pair. So, after the operator is applied, these elements change to $x_a = (x_b.OI, x_a.P)$ and $x_b = (x_a.OI, x_b.P)$. Figure 1.3 illustrates this.

- **Swap machines operator:** This operator is very similar to the previous one with the only difference being that machines are swapped instead. Hence if the operator randomly chooses element $x_a = (x_a.OI, x_a.P)$ and $x_b = (x_b.OI, x_b.P)$, then after swapping, they become $x_a = (x_b.OI, x_a.P)$ and $x_b = (x_a.OI, x_b.P)$.

- **Swap pairs operator:** This is another variation of two previous swap operators. In this case, both the order item and the machine are swapped, so basically, we change the priorities of the selected pairs. Hence, if the operator picks elements $x_a = (x_a.OI, x_a.P)$ and $x_b = (x_b.OI, x_b.P)$, after execution of the operator, they become $x_a = (x_b.OI, x_b.P)$ and $x_b = (x_a.OI, x_a.P)$

- **Move pair left operator:** This operator picks a random element $x_i$ and moves it a random number of places to the left, thus randomly increasing its priority. Figure 1.4 illustrates this.

---

**Fig. 1.3** Swap order item operator. Vector $x$ before and after the operator.

**Fig. 1.4** Move order to the left operator.
• Mutate machine operator: This operator randomly chooses a pair \( x_i = (x_i.OI, x_i.P) \) and randomly changes its machine to another machine which is capable of producing its order item \( x_i.OI \).

• Prioritize late order operator: This operator searches for orders that are scheduled to be completed later than their due date according to current decoded individual. It randomly increases the priority for each such order by shifting them left.

• Move order to a less occupied machine operator: This operator is used to decrease the load of one machine by moving orders to others. It randomly chooses a pair \( x_i = (x_i.OI, x_i.P) \), and if there is a machine capable of producing \( x_i.OI \), which is less loaded than \( x_i.P \), it changes the assigned machine the new one.

• Group orders with the same tool operator: A group is a sequence of consecutive pairs that share a common value. There are two types of groups: tool groups and material groups. This operator randomly chooses a pair and then finds a tool group that this element belongs to by searching to the left and to the right. Then it tries to find another tool group that has the same tool, and places the two groups together. Figure 1.5 illustrates this.

```
C B A A A C C B B B A A A A A B
               ------------------------
C B A A A A A A A A C C B B B B
```

Fig. 1.5 Group orders operator.

• Group orders with the same material operator: This operator is similar to previous one, except that it uses glass type as the grouping criterion.

Penalties

The evolutionary algorithm uses the following penalties:

• Late penalty: This penalty is used to avoid late orders. The value returned by this penalty is the sum of the number of hours all the orders are late by.

• Early production penalty: Orders that are completed excessively early are also not preferable because of the need to store them somewhere until the delivery date. This penalty is not as severe as late penalty, but is nonetheless needed.

• Material change penalty: As discussed earlier some cleaning is needed when changing materials in the powerpacks. Therefore, it is preferable to have as few material changes as possible. This penalty penalizes individuals in proportion to the number of material changes in their phenotype.

• Tool changes penalty: This is similar to the material change penalty, except that individuals are penalized in proportion to the number of tool changes in their phenotype.
• **Surplus material penalty:** As previously mentioned, powerpacks produce molten glass in batches. If all of the material in a batch is not used, the leftover would be wasted which is not desirable. The penalty applied to individuals is in proportion to the total wastage that occurs in their phenotype.

**Evaluation Function**

The evaluation function of an individual $X$ is calculated in the following way:

$$F(X) = - \sum_i (R_i \times \text{Penalty}_i(X))$$ (1.3)

where $R_i$ is a $\text{Penalty}_i(X)$ coefficient. Our evolutionary algorithm will try to maximize this function $F(X)$.

### 1.2.5 Experiments and Results

The evolutionary algorithm as described above was implemented in Java. We used a Core 2 Duo T9300 machine with 2 GB of RAM running Windows Vista to test our solution. We used a dataset with 3 production lines, 2 power packs and 327 ordered items. The optimization runs took 5-6 minutes on average. The following parameters were applied:

- Population size = 2000
- Mutate machine operator probability = 0.5
- Swap probability = 0.5
- Move probability = 0.5
- Metal changes penalty coefficient = 100.0
- Tool changes penalty coefficient = 500.0
- Surplus metal penalty coefficient = 0.1
- Tournament selection of size 2
- A steady state population was used

Figure 1.6 shows the fitness over generations in a sample run and the best run. The delay penalty function over generations is shown in Figure 1.7.

We made 90 runs of our algorithm on the same dataset. This experiment shows stable results with median of $-4.31E+08$ and standard deviation of 699205.8988.

The software developed is currently in use at the company as an optimization tool and decision support system. The modeling of different situations and what-if scenarios becomes a rather easy task which can lead to better production strategy improvements.
1.3 Optimization of Operation of a Real-world Steel Company

In this section, we present a real world example of the steel sheet production optimization problem, then the proposed solution, and finally the experiments conducted on this problem.

1.3.1 Problem Statement

Suppose that a company called Global Steel has several plants across Australia and specializes in producing sheet steel. Sheet steel products are ba-
ically thin sheets of steel that have been coiled into rolls. There are many
different types of products that the company can manufacture, that differ in
the chemical composition, density and width. Furthermore, the rolls can be
configured in different diameters and cores (see Figure 1.8). When a plant
manufactures a roll of steel, the width of the roll cannot be changed. However,
customers typically place orders for smaller width, and so, large rolls must be
cut into pieces along their width to satisfy these orders. This requires orders
of similar configuration to be grouped together before cutting the sheets (see
Figure 1.9).

The constraints and business rules of this problem are listed below; many
of which are similar to the previous problem.

- Each plant’s daily work hours are limited.
- Each plant can produce a subset of all possible goods. Some products
can be produced at multiple plants, but the cost and rate of production
is different at different plants.
- Each plant is equipped with a particular number of ‘knives’, which are
used to cut large rolls into smaller ones. Each plant can only produce
sheets of fixed width and has special knives to cut to the width needed.
- Each plant has a fixed daily operating cost which is the cost of running
the plant, and does not depend on the actual products produced.
- After production, products are delivered to customers, and this incurs a
transportation cost. There are defined transportation costs between each
customer and plants. Some plants cannot ship to some customer due to
specific business rules.
- Due to shipping constraints, export orders must be produced and shipped
from a single plant; whereas for others, part of the order can be produced
at one plant and other parts at other plants.
- Orders are produced in batches by the plants, which will be explained in
detail later in this section.

In order to understand the problem better, let’s look at a simplified ex-
ample. Suppose we have 5 plants: \( P_1, P_2, P_3, P_4, P_5 \) and 7 customers \( C_1, C_2, \ldots, C_7 \). Let’s say that customer \( C_2 \) has an order for 5 tonnes of product \( I_1 \),
and 8 tonnes of \( I_3 \). Customer \( C_6 \) wants 15 tonnes of \( I_1 \), 3 tonnes of \( I_2 \), and
10 tonnes of $I_3$. $I_1$ can be produced only at $P_2$, $I_2$ can be produced at $P_1$ and $P_4$ and $I_3$ can be produced at $P_2$, $P_3$ and $P_5$ although with different costs of production. (See Table 1.2(a)). Table 1.2(b) shows the products that can be produced at each plant, as well as the tonnes per hour rate of production for each.

Suppose that we have to complete these orders in a time frame of 100 hours. Let’s look at one possible way of distributing this workload amongst the plants. As $I_1$ can only be produced at plant $P_2$ we are forced to assign all 100 tonnes of $I_1$ there. This then implies that the required 30 tonnes of $I_2$ must be produced at $P_4$, using up 20 hours of the available time. 280 tonnes of $I_4$ still need to be produced. $P_2$ could produce 200 tonnes of it, $P_3$ could produce 150 tonnes and $P_4$, because it only has 80 hours of available time left, could produce 120 tonnes. Thus the 280 required tonnes of $I_4$ can be split amongst these three plants in various ways.

<table>
<thead>
<tr>
<th>Customer</th>
<th>$I_1$</th>
<th>$I_2$</th>
<th>$I_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C_2$</td>
<td>50</td>
<td>180</td>
<td></td>
</tr>
<tr>
<td>$C_5$</td>
<td>30</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td><strong>20</strong></td>
<td><strong>3</strong></td>
<td><strong>18</strong></td>
</tr>
</tbody>
</table>

(a) Demand table: The quantity of each item ordered by each customer.

<table>
<thead>
<tr>
<th>Plant</th>
<th>$P_1$</th>
<th>$P_2$</th>
<th>$P_3$</th>
<th>$P_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$I_1$</td>
<td>1.0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$I_2$</td>
<td>1.0</td>
<td>1.5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$I_3$</td>
<td>2.0</td>
<td>1.5</td>
<td>1.5</td>
<td></td>
</tr>
</tbody>
</table>

(b) Production capacity table: Tonnes per hour production rates of each plant for different products.

Plants can produce rolls of sheet steel of a certain width only, which is generally much larger than the widths typically required by customers, and therefore, ordered items that have the same type, core, and diameter must be batched into a single roll. That is, a large roll is cut into pieces to satisfy these orders. This is illustrated in Figure 1.9(a).

This process of batching sometimes results in wastage. To illustrate, suppose that customers $C_1$ and $C_2$ need rolls of product $I_1$, of the same diameter and with the same type of core. $C_1$ needs 2 rolls of width 500 mm, and $C_2$ needs 4 rolls of width 600 mm. Let’s assume that the only plant that can produce $I_1$, must produce rolls of steel of width 2000 mm. In this situation, batching can be done as shown in Figure 1.9(b). The plant can produce two rolls of $I_1$ of width 2000mm. One can be cut into 4 pieces: 3 pieces of 600mm and 1 piece of 200mm. The 3 pieces of 600mm can be used for orders, but the remaining 200mm is wasted. It may be recycled, but a loss is incurred nevertheless. The second roll of 2000mm can be cut into widths of: 600mm, 500mm, 500mm, 400mm, the first 3 of which would go towards satisfying orders, and the last piece ends up as wastage. Some wastage is unavoidable, but clearly it is desirable to have as little wastage as possible in order to max-
imize production efficiency and hence profit. Intelligent grouping of orders can minimize wastage.

Plants use cutting tools called knives, physically arranged in parallel, to cut large rolls into smaller ones. Different plants are equipped with different numbers of knives, and this presents another constraint to consider when batching orders. A plant equipped with \( n \) knives can cut a roll into at most \( n+1 \) pieces. In the above example (Figure 1.9(b)) cutting a large roll into four small ones, would have required three cuts and hence three knives.

The goal of the problem is to find a set of assignments of orders to plants that optimizes the overall profit of the company. Profit is determined by the following formula:

\[
Profit = S - C_m - C_t
\]  

(1.4)

where \( S \) is selling price of the goods produced, \( C_m \) is the cost of manufacturing those goods, and \( C_t \) is the cost of transporting the goods to customers. Selling prices may vary for the same product depending on the market which it is being sold. The cost of manufacturing a particular order item is the sum of the cost per hour of producing that grade of sheet steel, and the fixed hourly cost of running the relevant plant for the time it takes to produce the required tonnage. Transportation costs vary with the destination address of the customer. Export orders naturally incur higher transportation costs. These last two components are the negative components of the profit; the bigger they are, lesser is the profit. The selling price is the positive component of profit; it is known before hand, and does not depend on which plant an order is allocated to. Therefore, the goal of the problem is to minimize the transportation and manufacturing costs.
1.3.2 Solution

The software created for this problem used an evolutionary algorithm to find a solution. It is shown in Figure 1.10. The software is fully configurable to cater to various selling prices, transportation and manufacturing costs. Given the projected orders for an upcoming month, it seeks an optimal set of assignments that maximizes the profit of the company. Being a decision support tool, the software also provides financial comparisons with human-proposed allocations. In practice, the software suggests decisions that have better financial bottom lines.

Fig. 1.10 Steel optimizer’s main screen.

Solution Representation

The representation used for the evolutionary algorithm consists of two parts. The first part is a string of quantities assigned to each capable plant for various order items. Figure 1.11 illustrates the structure of this representation hierarchically. It can be visualized as a string of $s$ records, one for each order being considered. In turn, each order record can be thought of as consisting of a number of smaller records, one for each item in that order. And finally, each order item record can be thought of as consisting of a number of allocation records, one for each plant that is capable of producing the item under consideration. Such a lowest-level record is called a plant quantity allocation (PQA). More formally, assuming that there are $s$ orders to be satisfied, a candidate solution is represented as $X = (x_1, \ldots, x_s)$, where for $1 \leq i \leq s$, $x_i = (y_{1i}, \ldots, y_{mi})$, where $m_i$ is the number of items in order $i$. And further, for $1 \leq j \leq m_i$, $y_{ji} = (z_{1ij}, \ldots, z_{k_{ij}})$, where $k_{ij}$ is the number of
plants capable of producing item $j$ of order $i$. Each $z_{ij}$ is a tuple $(O_{ij}, p, q)$; it is a plant quantity allocation in which plant $p$ is allocated $q$ rolls of item $j$ in order $i$.

**Fig. 1.11** First part of the representation of an individual.

The second part of the representation is a string of permutations of the plants in the system. Each permutation represents the preferred order of the plants for use in export orders. This can be formalized as $X_{pref} = \{x_{pref_1}, \ldots, x_{pref_s}\}$, each element $x_{pref_i}$ is a permutation of the set of plants associated with order $i$. Figure 1.12 illustrates the second part of the representation.

**Population Initialization**

At the start of the evolutionary algorithm, a population of randomly initialized individuals is created. First, each order item cell in Figure 1.11 is initialized by uniformly randomly divvying up the number of rolls of steel required for that order item into the PQA cells lower down in the hierarchy. Second, each order cell in Figure 1.12 is assigned a randomly generated permutation of the steel plants.

**Main Algorithm Loop**
After the population of individuals is initialized, the evolutionary algorithm enters a main loop in which the population is evolved until a specific criterion is satisfied. Each individual is assigned a fitness value by an evaluation function which is designed so that its value is indicative of how well the individual solves the problem. The evaluation functions in our work make use of penalty functions which essentially assign a value to an individual in proportion to how badly the individual violates some constraint.

During the main loop, individuals of higher fitness are probabilistically given more chances to reproduce, that is, to make modified copies of themselves to replace them if the modified copies have better fitness values. The changes made to an individual are accomplished by means of operators, which are functions that take one or more individuals and combine, alter and manipulate their structures in a partially random way. After operators are applied, some of the resulting new individuals are passed through repair functions, that attempt to randomly, but intelligently alter their structure so as to reduce the extent of their constraint violations.

### Penalties

Not every candidate solution that can be constructed as described above would represent a feasible solution. For example, if in a candidate solution, a plant is assigned more work that it could possibly produce in the time frame under consideration, then the solution is infeasible. Many techniques are known to handle different types of constraints and most of them are based on penalty functions. The approach used for this algorithm is also based on penalty functions. Penalties were used to deal with both hard and soft constraints. Penalty functions for hard constraint were given larger weights than those for soft constraints. The penalty functions developed were:

- **Quantity production penalty:** This penalty is used to avoid underproduction or overproduction of an ordered item. If an order item requires $N_1$ rolls of a given product, and the sum of the PQA quantities for this order item is $N_2$, then this penalty function returns a value that is proportional to $|N_1 - N_2|$. This is a soft constraint because it is not always possible to produce the exact amount demanded, as plant capabilities are limited and fixed.
- **Plant capacity violation penalty:** If the amount of work allocated to a plant exceeds its capacity for the time period, then this function penalizes the solution in proportion to the over-allocation. This is a hard constraint because a plant cannot produce more than its capacity.
- **Wastage penalty:** Wastage is possible as a result of the batching procedure that occurs during production. This function penalizes in proportion to the number of tonnes of wastage. This is also a soft constraint.
- **Export order single source penalty:** Orders for export customers should be produced at the same plant, and this function penalizes export orders that use multiple plants. This is a hard constraint.

### Evaluation Function
Each candidate solution is assigned a fitness value by an evaluation function that returns a value that is indicative of how well that candidate solves the problem. For this problem, the fitness function is defined as:

$$f(X) = \text{Profit}(X) - \sum_i (R_i \times \text{Penalty}_i(X))$$  \hspace{1cm} (1.5)$$

$X$ is a candidate solution, and $R_i$ is the coefficient of the penalty function $\text{Penalty}_i(X)$. $\text{Profit}(X)$ is calculated using three components: manufacturing cost, transportation cost and selling price, as described previously. Our evolutionary algorithm tries to maximize this function $f(X)$, i.e. maximize profit, and minimize constraint violations.

**Operators**

An evolutionary operator takes one or more candidate solutions and transforms it (or them), into a new candidate solution. This is typically done by using partially random manipulations of the representation. Operators are employed probabilistically. We used several operators to solve this problem. Some of them look like classic mutation and crossover, and some of them are modified versions of these two. The operators used are the following:

- **Quantity crossover:** This is a classic crossover-like operator. It takes two individuals, randomly chooses a crossover point at the boundary of an order item cell, and performs a crossover at this point.
- **Quantity mutation:** This operator is similar to classical mutation. It takes an individual, and probabilistically chooses several PQA cells and changes their quantity values by random amounts within a certain range. The changed amount is chosen randomly in the range $[-\alpha, \alpha]$, where

$$\alpha = \frac{\text{quantityOrdered} - \text{quantityAllocated}}{|\lambda(OI)|}$$  \hspace{1cm} (1.6)$$

where $\lambda(OI)$ is the quantity required for a particular order item.
- **PQA Shuffling:** This operator randomly chooses one or several ordered items and shuffles quantities assigned to plants within each ordered item. It preserves the sum of quantities within one ordered item before and after the shuffling. So, if we randomly chose ordered item $OI_k$ then $G(OI_k) = G'(OI_k)$, where $G$ is the sum of all quantities of ordered item $OI_k$ assigned to all capable plants before applying the shuffling operator and $G'$ is the sum after applying it.
- **Preferred plant operator:** This operator looks at the second part of the representation of an individual, i.e. the permutation of plants, and performs a random transposition on it, i.e. it randomly swaps two elements of the permutation.

**Repairers**

A repairer is an operator that tries to modify a candidate solution in such a way as to eliminate or reduce the number of constraints that it violates.
In our evolutionary algorithm we use a set of problem-specific probabilistic repairers. The main difference between these and the classical repairers is that they are not executed each time a solution gets evaluated, but instead are applied probabilistically, which allows for tuning for better performance. The repairers used are:

- **Quantity allocation repairer:** This repairer tries to fix errors in over- and under-allocation of a particular order item amongst capable plants. It works in the following way. We choose an order which has underproduction or overproduction and increase or decrease the quantity produced at each plant. The increment or decrement are not random. The algorithm decides on the changes to be made. This will increase the fitness of individual, but being a random change, some diversity will be maintained.

```
P1 P2 P3 P4 Total
Quantity before repair 6 13 16 9 44
Quantity after repair  5 11 15 8 38
```

For instance, suppose that for a particular order item, 36 rolls are needed, but according to an individual about to be repaired, the capable plants have been allocated 44 rolls. The second row shows what might be the result of applying the quantity allocation repairer. Some values increased, some decreased. The new solution is still not perfect, but overproduction has been reduced.

- **Overallocation repairer:** This repairer is different from the quantity allocation repairer. It checks the total workload allocated to a plant, to see if it exceeds that plant’s capacity for the time period. If it does, the repairer randomly chooses some order item that uses the plant in question, and reduces its quantity allocation by one. It repeats this procedure until the plant can produce its assigned amount in the given time period.

- **Export order source repairer:** This operator checks for export orders in the individual that use multiple plants, and performs repairs according to the preferred plant permutation. For any such order, it tries to assign the whole work to the first capable plant in the permutation list.

- **Final repairer:** Unlike previous repairers, this repairer is run only at the end of evolutionary algorithm to “polish” the solution. It uses non-probabilistic versions of previous algorithms, since at the end of the algorithm, we want to make solutions as good as possible without worrying about diversity of individuals.

**Optimization of Batching**

One of the challenges of this problem is grouping ordered items into batches. The problem is known in the literature as the *stock cutting problem*. Recall that the problem is the following. We have a set of ordered items all with the same product, core, and diameter, but with different widths. These must be allocated to a plant that can produce rolls of a fixed width which can later be cut into smaller rolls to satisfy the orders. For convenience,
let us refer to these large fixed-width rolls as max-rolls. We need to group the ordered items into batches in such a way that the number of max-rolls produced by the plant is minimized, thus minimizing the wastage.

For this problem, a greedy approach was taken. The algorithm looks at all order items that are assigned to a particular plant, groups those that are compatible (i.e. that share the same product, diameter and core) and forms batches in the following way. The order item with the largest width is assigned to the first max-roll. Some of the width of this max-roll may be unassigned. The next largest order item is considered next. If there is a max-roll in which it can fit, it will be placed there, otherwise a new max-roll will be created. The same procedure is repeated for all order items. The pseudo-code of this algorithm is shown below:

Algorithm 1 Batching algorithm

\[
\begin{align*}
\text{maxRolls} & \leftarrow \emptyset \\
\text{for } \text{group} \in \text{orderItemGroups} & \text{ do} \\
& \text{sortByWidth(group)} \\
& \text{for } \text{orderItem} \in \text{group} \text{ do} \\
& \text{if canFindAvailableSpaceInMaxRolls(item) then} \\
& \text{placeItemInAvailableMaxRoll(item)} \\
& \text{else} \\
& \text{newMaxRoll} \leftarrow \text{createNewMaxRoll()} \\
& \text{newMaxRoll.add(orderItem)} \\
& \text{maxRolls.add(newMaxRoll)} \\
& \text{end if} \\
& \text{end for} \\
& \text{end for} 
\end{align*}
\]

The procedure \(\text{canFindAvailableSpaceInMaxRolls(orderItem)}\) iterates through all available max-rolls and checks if the given \(\text{orderItem}\) can be placed in a max-roll. If it can find such a max-roll, it will return true, if no such max-roll is available, the procedure returns false. The procedure \(\text{placeItemInAvailableSpace(item)}\) places given \(\text{item}\) in the available roll that has been found by \(\text{canFindAvailableSpaceInOrders}\).

### 1.3.3 Experiments and Results

The evolutionary algorithm in this study was implemented in Java, and was tested on real world industrial data. The purpose of the software is to provide decision support to business managers who approve production allocation.

In this subsection we present the results of experimental runs of this algorithm with a view to gaining an understanding of how the algorithm solves the problem.
An experiment was performed on a set of real-world data for a particular month. Due to the stochastic nature of evolutionary algorithms, the experiment was executed multiple times so as to allow for statistically accurate results. The specific parameters used for the experiment are shown below. The penalty coefficients were tuned for the business entity under consideration, and would perhaps not be ideal if this algorithm were to be replicated for another industry or business environment. In such a case, it is likely that re-tuning of the coefficients would be necessary. As such, it would not be prudent to try to interpret the significance of these penalty coefficient values.

- Population size = 250
- Number of trials = 90
- Mutation probability = 0.3
- Overallocation repair probability = 0.1
- Underproduction penalty coefficient = 10 000 000
- Overproduction penalty coefficient = 100 000
- Wastage penalty coefficient = 1
- Plant capacity violation penalty coefficient = 1 000 000 000
- Export order multi source penalty coefficient = 10 000 000 000
- Transportation penalty coefficient = 1
- Tournament selection of size 2
- A steady state population is used

Figure 1.13 illustrates the evolution of the profit variable during the lifetime of a single run of the experiment. From time to time, there is a decrease in the profit value; this corresponds to instances where the algorithm is able to find a solution that dramatically reduces constraint violations, but at the expense of less profit being generated. However, based on the penalty values being generated, the solution is still considered to be superior to what was in hand before. However, the algorithm is eventually able to produce highly profitable solutions.

Figure 1.14 shows the decrease in the summed penalty function values over the lifetime of a single run of the experiment.

A 90 run execution shows median 1.18E+07 and 52110.52404 standard deviation of profit.

The assignments generated by our software system produce financial results that are 4% better than those produced manually by the company’s experienced team of planners.

1.4 Discussion

In the previous two sections we described two optimization problems, their challenges, proposed algorithms to solve them, and analyzed their solutions. Though the two problems are somewhat similar in the sense that they both
relate to supply chain operations, they have some critical differences from a business perspective. One involves silo-based optimization and the other involves global optimization from a business perspective. Exploring these differences was the main goal of this chapter. It was not our goal to pit the two approaches against each other for the case studies considered, instead we sought to look at these two problems and their solutions by evolutionary algorithm in sufficient detail so that the reader would be able to compare and contrast, at a more-than-theoretical level, the approaches required.
The objective at the glass company was to optimize the operations of a single factory. In this case study, the sequence in which the orders are placed on the machines affects efficiency and hence profitability. A good sequence would not involve much re-configuration and cleaning of the machine between orders. Hence, the goal of the optimizer was to place orders on machines in such a way as to maximize profitability at the factory, by trying to run the machines at maximum capacity so that as many open orders as possible could be satisfied. Besides that, it also aimed to minimize production costs by grouping orders in such a way that raw material changeovers between orders were minimized, and machine tool changes between orders were also minimized.

In the second case study, the objective was to optimize operations over a set of steel mills. In general, for any given order item, there is a choice of one or more factories at which the item can be produced. Therefore, a decision has to be made on how to split the order amongst the factories capable of producing it. The decision as to which one (or ones) to use, affects the company’s profitability. A customer’s delivery address may be closer to one factory than it is to another, thus transportation cost is affected. Also, some factories are more efficient than others, so producing the same item at one factory may cost more than the other factories. Thus, one goal of the optimizer, in this case study, was to simply allocate orders to factories in such a way that the cost of production and transportation over all plants was minimized. Another facet of this problem is that before production, each plant has to batch orders, so several ordered items can be produced in one batch. This results in some wastage, and so, another goal of the optimizer was to minimize wastage over all the plants by batching efficiently.

Comparing these two problems, we can see that in the glass company problem, we are only concerned about optimizing production in one local factory. The scope of the problem does not include any information about availability of raw materials at any given time, nor does the problem deal with the distribution of products to customers, or any logistic issues. (See Figure 1.15) In the steel company, there are two levels of optimization. First, we try to optimize the production and transportation costs by distributing orders among plants. Second, we also optimize batches at each plant; this optimization is limited to information of orders that have been assigned to that plant and does not take into account any production at other plants, transportation and other costs.

The procedure of packing orders into batches at the steel factory is similar to the first problem at the glass factory; both are unconcerned with optimization at other units, they both optimize in local search spaces. After local optimization is complete at every steel plant, information about batched orders is returned to the top level optimization. The top level optimization procedure uses it to help guide optimization at a global level, taking into account the big picture of the whole business system. (See Figure 1.4) The steel company is an example of global optimization from a business per-
spective, whereas the glass company is an example of **local optimization** from a business perspective.

![Diagram of local optimization](image1.png)

**Fig. 1.15** Local optimization. Production and distribution units are not involved in optimization.

![Diagram of global optimization](image2.png)

**Fig. 1.16** Global optimization (steel company).

The experiments performed in this chapter are based on software that was written to solve real-world business problems for actual industrial companies. Values have been transformed to protect private commercial data, but otherwise are representative of actual day-to-day operations at the respective companies. The experiments performed were not for the purpose of demonstrating that one solution approach is better than another, but rather for demonstrating, with appropriate rigor, that our case studies have been solved in stable ways, one using a silo-based optimization approach, and the other using a business-perspective global optimization approach. In the case of optimizing the glass factory, our results indeed show that the changeover penalties are reduced over the course of a run of the algorithm. The fact that the results produced were actually significantly superior to the solution produced by a human being, was actually confirmed by representatives of the company. In the case of optimizing the steel factories, financial results were an integral part of the software’s evaluation and show objectively (on numerous data sets) that the results produced by the software were vastly superior to those produced by human planners.

Using these two example problems as a basis, we can try to take a more general point of view, and identify some characteristics of global optimization from a business perspective:
Two or more levels of optimization. In the steel company problem we have two levels. One is optimization of batches at the plant level, the other one is optimization of the whole system using information from the plants.

Partial dependence of lower level components on higher level components. In our example, optimization of batches at each plant is only concerned with its own orders, and does not need any external information about freight rates, customers and other plants. It only needs to know about the total tonnages of each product that have been assigned to the plant.

Limited communication between components. In global optimization problems, different components have limited information exchange due to either business rules or problem specific constraints. In the steel problem, information about local optimization at each plant is only indirectly communicated between plants by means of the higher level optimizer adjusting total tonnage assignments in response to the fitness values.

Different search spaces structures. We can think about a global optimization problem as several local optimization problems plus one or more higher level problems that combine the lower level problems. Thus, the biggest difference between local optimization and global optimization is in the search space structure and the sequence of steps needed to evaluate a candidate solution.

Future Improvements
There are several ways in which the global optimization case study can be broadened for future research:

The steel mill case study could be expanded to consider raw materials' input, as well as secondary processing of steel products that occurs after the rolls of sheet steel leave the factories. It may even be expanded further to include consideration of sales at retail outlets.

Future work can consider per-factory optimization under the umbrella of global optimization in the business sense. In the case of our sheet steel company, this would mean looking at what happens after the production quantities have been allocated in bulk to the various factories. The exact sequencing of the production could then be optimized locally (i.e. within a single factory) by reducing changeovers between the execution of orders, by reducing the storage of finished products in inventory, by minimizing wastage in the form of small cuts of steel, and by sequencing in such a way that orders are completed before their due dates.

Finally, instead of only considering per-factory optimization after the higher-level global optimization is complete, it is possible to develop an optimizer that considers fitness values based on both global and per-factory feedback. Such an optimizer could use two-way feedback so that the lower-level optimization affects decisions made at the higher level. This was partially done with the batching procedure, but further work such as sequencing procedures can also be done.
1.5 Conclusions and Future Work

There exists a large body of research on the application of optimization techniques to the highly-focused aspects of real-world business problems. This research may be suitable for small businesses seeking to optimize their production efficiency and profit. But for bigger businesses, especially ones that are vertically integrated, optimization needs to be looked from a much broader point of view.

Optimization of any of those business’ silos would definitely lead to an improvement in their operations in terms of time and money, and separate efforts to optimize multiple silos would also yield benefits. But businesses should be more interested in the optimization of their whole system rather than optimization of single components. A larger opportunity exists for global optimization from a business perspective. By modeling the situation in such a way as to have limited communication among the parts of the model representing the various silos, we can achieve a level of optimization that is better than what is attainable by isolating and optimizing individual components.

In this chapter, we considered two real-world business situations. The first looked at an important but isolated aspect of the operation of a glass factory. The second considered a multi-factory sheet steel business, and illustrated how a broader perspective can be taken in such a way as to optimize the operations of all the factories at the same time, as well as some aspects of the operations of each individual factory. These case studies were presented to illustrate some of the concepts associated with “local” and “global” optimization from a business perspective, and not to pit the approaches against each other. In fact, global optimization uses local optimization components as building blocks that optimize each silo.

From an abstract point of view, business-perspective global optimization fits into the framework of traditional global optimization. What distinguishes it as a subclass from traditional business optimization scenarios is that its problems can be formulated as objective functions with more complex-structured search spaces. For example, whereas a traditional problem might have the set of permutation of a finite number of objects as its search space, a business-perspective global optimization problem might have as its search space the vector space of \( n \) elements, each of which is taken from the space of permutations of a finite number of objects.

We believe that this abstraction will be useful for studying global optimization problems by allowing the dependence or partial dependence among components to be investigated in a more controlled setting. It would also permit researchers to examine subtle properties of such systems. Our future work will be based on such conceptualizations.
1.6 Acknowledgements

We thank the anonymous reviewers for their very insightful comments. This work was partially funded by the ARC Discovery Grant DP0985723 and by grant N516 384734 from the Polish Ministry of Science and Higher Education (MNiSW).

References


