



**Országos Tudományos Diákköri
Konferencia
2024**



**Risk-Based Maintenance Optimization of Production Tools in a
Dynamic Environment**

Készítette: Horváth Judit

Témavezető: Dr. Ruppert Tamás, Darányi András

Pannon Egyetem, Mérnöki Kar, Rendszermérnöki Intézeti Tanszék

A pályamunka lezárásának dátuma: 2024.12.19.

Contents

1	Introduction	2
2	Literature Review	4
2.1	Flexible Manufacturing Systems	4
2.2	Maintenance Strategies	5
2.3	Reliability Models	7
2.4	Genetic Algorithms	11
3	Developed algorithm	14
4	Case study and results	19
4.1	Description of the case study and parameters	19
4.2	Results	21
5	Conclusion	24
6	Appendix	27

Abstract

Flexible manufacturing systems are systems where machines are equipped with multiple tools, allowing them to produce various products by switching tools. This setup has many advantages, but one downside is that it is more challenging to perform proper maintenance on such a system because the usage and condition of the tools are highly variable due to the flexible scheduling.

Risk-based maintenance (RBM) is a maintenance approach that prioritizes maintenance activities based on risk, applying the appropriate maintenance strategy in every situation, whether it be preventive, reactive, or another approach. Traditional approaches to risk-based maintenance of complex systems, such as fault tree analysis, would be difficult to apply to FMS because modeling the entire system would require rebuilding the fault tree with every new production schedule.

Instead of the traditional approach, I focus on the maintenance of individual tools, as they are unique and simple, and I can perform reliability analysis based on their usage data. By combining the probability data obtained this way with the costs of maintenance and production downtime, I can create a risk-based objective function, which can form the basis of a population-based algorithm. Such an algorithm can provide a sufficiently good solution for determining when and what type of maintenance should be performed on a given tool.

In the case study, I create a model of a flexible manufacturing system in Python, generating usage data for the tools and assigning costs to maintenance events and production downtime. This allows me to obtain probability and severity data for the risk-based objective function, which will serve as the fitness function of a genetic algorithm.

The expected result is a maintenance model and algorithm applicable to flexible manufacturing systems, optimizing the scheduling of maintenance for individual tools.

Keywords: risk-based maintenance, genetic algorithms, flexible manufacturing systems, maintenance optimization

Chapter 1

Introduction

Maintaining production efficiency while managing costs is key to success in today's competitive manufacturing environment [1]. Flexible manufacturing systems (FMS), which allow production lines to be quickly reconfigured by equipping machines with multiple interchangeable tools [2], offer manufacturers a unique advantage in producing a variety of products on demand. However, this flexibility comes with a number of challenges, particularly in terms of maintenance. Unlike traditional systems, where maintenance schedules are more predictable, the changing usage patterns of FMS make it difficult to apply maintenance strategies correctly. As a result, there is a growing need for adaptive maintenance solutions that take into account both the condition of individual tools and the operational demands placed on them.

A promising approach to this problem is risk-based maintenance (RBM) [3]. RBM prioritizes maintenance actions based on risk, assessing both the probability of failure and the potential consequences to determine the most effective strategy, whether preventive, reactive, or otherwise. In an FMS, where the condition of each tool can change depending on its usage pattern, RBM can provide the flexibility required to dynamically adapt maintenance schedules. Traditional methods of implementing RBM, such as fault tree analysis [4], are limited in this context because they typically require the creation of complex system models. In an FMS, where production schedules and tool usage are in constant change, such methods become impractical because they would require frequent remodelling and recalculation.

This paper explores an alternative RBM approach by focusing on individual tools within an FMS. By analyzing the usage data of each tool, I can perform reliability assessments and determine the probability of failure based on historical usage patterns. Integrating these probability data with the costs associated with both maintenance activities and potential production downtime enables the creation of a risk-based objective

function. This function serves as the basis for a genetic algorithm (GA) designed to optimize maintenance decisions. Using a GA, my objective is to identify an effective maintenance schedule that minimizes downtime costs while preventing critical failures.

The cost aspect of this problem is particularly important as improper maintenance can lead to significant production losses. An optimized RBM strategy can support decisions on whether to perform extensive maintenance (at a higher cost) or take a short-term survival approach, balancing immediate costs with potential future risks. The results of this paper will provide insight into how an RBM strategy can improve the cost-effectiveness of FMS and reduce the risk of unplanned downtime.

For demonstration purposes, I construct a hypothetical case study in Python, inspired by real industry practices and examples from the literature. In this case study, I create a model of an FMS environment, generating tool usage data and associating specific costs with different types of maintenance events and downtime. The resulting data enables the calculation of a risk-based objective function, which acts as a fitness function within the GA. Although the study focuses on tool maintenance in manufacturing, the principles of the model are versatile and can potentially be applied to other areas, such as skills maintenance in workforce planning, where similar challenges exist in managing the reliability and cost-effectiveness of resources.

In summary, this research aims to present a maintenance optimization model and algorithm for FMS that specifically target the dynamic and risk-sensitive nature of tool maintenance. By implementing a GA with a risk-based objective, I propose a solution that not only improves maintenance efficiency but also provides a generalizable framework for managing RBM in other contexts.

Chapter 2

Literature Review

In this review of the literature, key concepts and methodologies that underpin RBM optimization in a dynamic production environment are explored. The review begins with an examination of FMS, which provide the adaptable system required for efficient and responsive production setups. Following this, I explore various maintenance strategies and approaches which are important in ensuring proper functioning in complex systems. Reliability analysis is then discussed, as it serves as the basis for assessing and predicting system performance over time. Finally, I examine the application of GA in optimization, highlighting their effectiveness in tackling complex, non-linear problems like maintenance scheduling. Together, these topics provide a comprehensive foundation for understanding and developing a risk-based approach to maintenance optimization, supporting the objectives of this project.

2.1 Flexible Manufacturing Systems

FMS are systems that can adapt to changing conditions, designed to easily adapt to the production of different qualities and quantities of products. This flexibility can mean different things. One category of flexible manufacturing systems is the flexibility of routing [2], which means that a product can be produced by multiple machines, so that the failure of one machine does not compromise the stability of production. The other category is machine flexibility [2], where one machine can produce multiple products using interchangeable tooling. In this way, such a system can easily adapt to changing customer needs. The latter category has been taken as a basis. In addition to flexibility, the main advantage of flexible manufacturing systems is a high degree of automation and fewer downtime compared to conventional systems [1]. However, the disadvantage is that FMS is more expensive to setup and more difficult to maintain properly [3].

2.2 Maintenance Strategies

To optimize maintenance procedures, we need to determine the most appropriate form of maintenance for the system or the tool. To do this, examine the main types of maintenance (Figure 2.1).

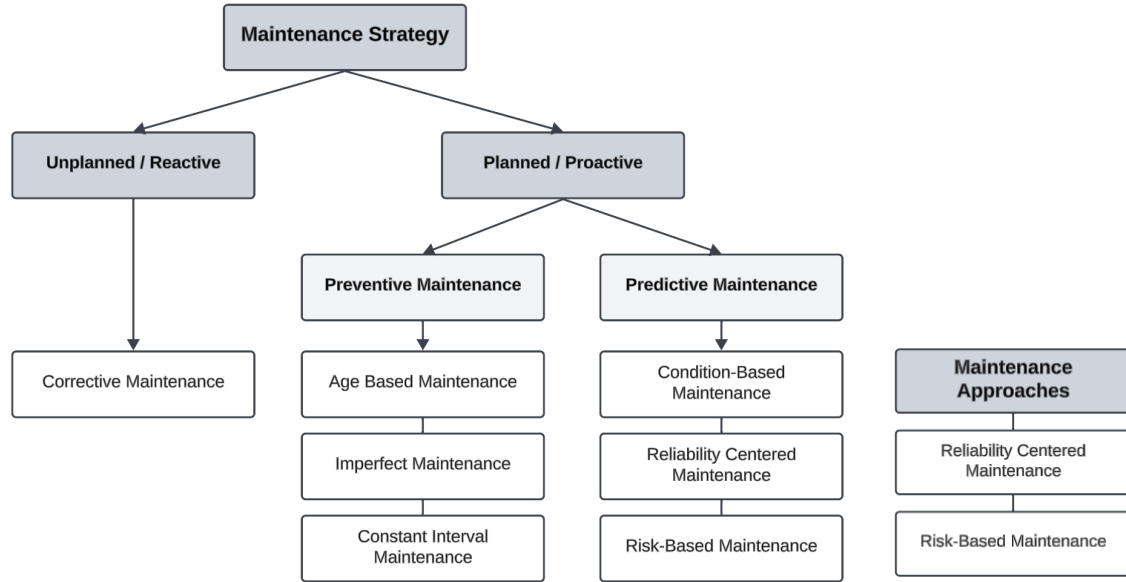


Figure 2.1: Maintenance strategies and approaches

Maintenance can be divided into two groups. **Reactive** forms of maintenance are not planned in advance, they react to the occurrence of a failure, while **proactive** forms of maintenance are planned in advance to prevent a failure from occurring [5] [6] [4].

Corrective maintenance [5] [6] (CM) reacts to the occurrence of an error; it waits until it appears. With reactive maintenance, maintenance is never performed unnecessarily, but in general, when a fault occurs, it becomes more costly to maintain than if it had been prevented. Furthermore, the consequences of an unexpected occurrence must be taken into account: the unplanned downtime of the machine or tool and the possible unavailability of the items needed to repair it.

With **preventive maintenance** [5] [6] (PM) strategy maintenance is scheduled based on a set of criteria to reduce the likelihood of failure. This can be divided into three groups: constant interval maintenance, age-based maintenance, and imperfect maintenance.

With the **constant interval maintenance** [5] [6] strategy, maintenance is performed at intervals, regardless of the state of the machine. If the machine breaks down, maintenance will be performed, but this will not change the maintenance schedule. The length of the intervals in this strategy aims to match the failure intervals of the machine, but if the interval is too short, maintenance is performed unnecessarily often, and if it is too

long, the chances of an unexpected failure occurring are increased.

With **age based maintenance** [5] [6], maintenance is only performed after a certain time has elapsed between two maintenance sessions, or we can say that the machine reached a certain age. If the machine breaks down in the meantime, the scheduled maintenance is delayed the same time after the repair. In this way, age-based maintenance events will result in fewer maintenance events overall than contingent interval maintenance, which may result in lower maintenance and downtime costs.

Compared to the previous two strategies, **imperfect maintenance** [5] [6] does not assume that the system is in its state of the art after maintenance, but takes into account the uncertainty of the current state of the equipment when scheduling future activities.

Predictive maintenance [5] [6] (PdM) often uses sensors and data processing to identify trends in machine condition, predict when a failure will occur and schedule maintenance [6]. The main difference with preventive maintenance is that the former produces a fixed schedule, while the latter produces a flexible one. It can be divided into two groups, condition-based maintenance and intermediate reliability-based maintenance.

Condition based maintenance [5] [6] (CBM) schedules maintenance based on sensors and data. When the preset values and the measured data are in good agreement, maintenance is performed. The advantage is that we can know for sure from the data that a change in the machine's condition has occurred that will lead to a fault, but the disadvantage is that on the one hand we have to install sensors and process the data, which makes condition-based maintenance expensive to maintain, and on the other hand it usually requires a fairly large degradation in condition to cross a threshold. This can potentially make maintenance more expensive than if it had been done earlier.

Reliability-centered maintenance [5] [6] [4] (RCM) uses reliability analysis of the system and does not aim to bring the system to a perfect state, but only to increase its reliability. This feature plays a role in increasing cost effectiveness. RCM recognizes that different machine parts or tools do not degrade in the same way under the same conditions. This maintenance approach seeks an optimal combination of other maintenance strategies to minimize equipment life cycle costs. The first step is performed using methods such as failure mode and effect analysis (FMEA), failure mode, effects, and criticality analysis (FMECA), or fault tree analysis (FTA) .

Risk-based maintenance [4] [7] is a maintenance approach that applies the right maintenance strategy in every situation by prioritizing maintenance activities based on risk. Risk (R) is a value obtained by multiplying the expected probability (or frequency) of an undesired event (p) by its severity (s).

$$R = p \cdot s \quad (2.1)$$

For maintenance, the event under consideration may be a failure, its severity (s) may also refer to the cost of maintenance or the cost of time outages. But it can also allow us to observe the risk of maintaining the machine too often and thus select the right maintenance approach; we can prioritize maintenance activities according to the situation.

If the expected frequency (p) is a probabilistic value based on the machine reliability analysis, then the RBM approach can use the FMEA, FMCEA and FTA methods discussed earlier, as can the RCM, the difference between the two approaches being that RBM takes into account severity and prioritizes based on risk, while RCM focuses on increasing reliability.

2.3 Reliability Models

During the operation of a system, unexpected events can occur, such as the failure of a machine or a component. These events are random, so their distribution over time can be described by probabilistic models in reliability analysis.

Reliability (R) or survival probability is the probability that the system will work as planned over a given time interval. As a probability, its value can be between 0 and 1. **Failure probability** (F) is the probability that the system will fail over a given time interval. As a probability, its value can also be between 0 and 1. The sum of reliability and failure probability is always equal to 1.

$$R(t) = 1 - F(t) \quad (2.2)$$

$$F(t) = 1 - R(t) \quad (2.3)$$

From the **density function** we can read the failure frequency of a given system, machine, component or device at a given point in time. The failure probability function shows us the cumulative frequency of failures too, in addition, we can write it as the integral of the density function:

$$F(t) = \int f(t) dt \quad (2.4)$$

It also means that the density function is the derivative of the failure probability function:

$$f(t) = F'(t) \quad (2.5)$$

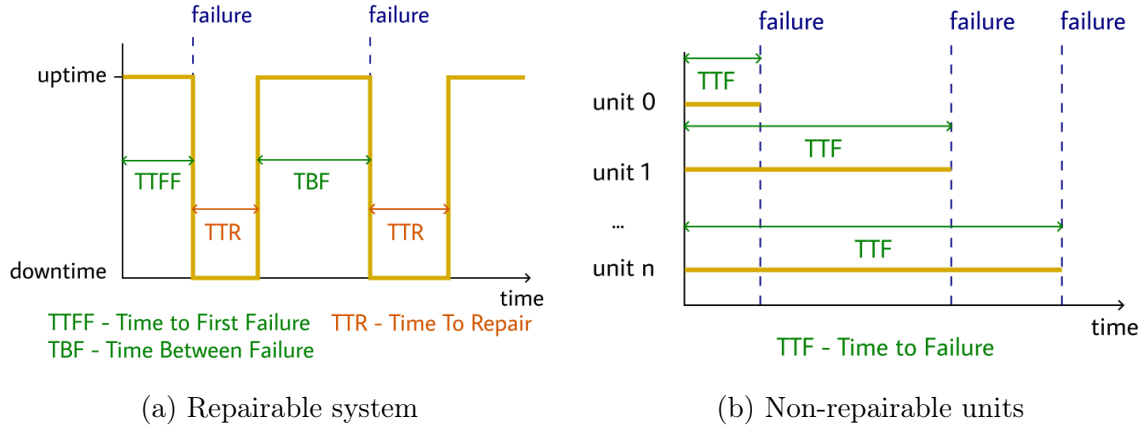


Figure 2.2: Scheme of a repairable system and non-repairable units

Parameters used to characterize reliability data [8]:

- **MTTFF** (Mean Time To First Failure) - The average uptime from the start of operations to the first failure (in repairable systems)
- **MTTR** (Mean Time To Repair) - The average time from the failure to the end of the maintenance.
- **MTTF** (Mean Time To Failure) - The average uptime from start of operation to failure (for non-repairable units)
- **MTBF** (Mean Time Between Failures) - The average uptime between two failures (in repairable systems)
- **Failure Rate** (λ) - The failures (at the in time t) divided by the sum of units are still working

$$MTBF = \text{mean of } TBFs = \frac{\text{uptime}}{\text{all failures}} \quad (2.6)$$

The difference in the nature of the failures between repairable systems and non-repairable units is shown in Figure 2.2.

In general, there are three main possible failures for any system: 1. Early Failures - Hidden failures that reveal themselves at an early stage; 2. Random Failures; 3. Wear Out Failures - Failures caused by aging. The probability of failure due to latent failures

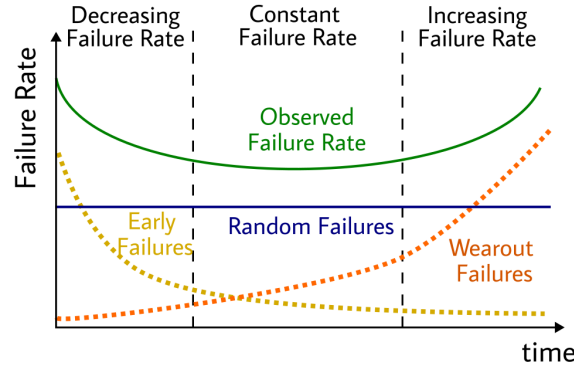


Figure 2.3: Bathtub curve

decreases over time and the probability of failure due to aging increases. Therefore, the failure rate function of engineering systems takes on a characteristic tub-shaped curve over the entire lifetime of the system, but this curve is not typical for all systems. On the Figure 2.3 can see how the failure rate varies over the life of a system.

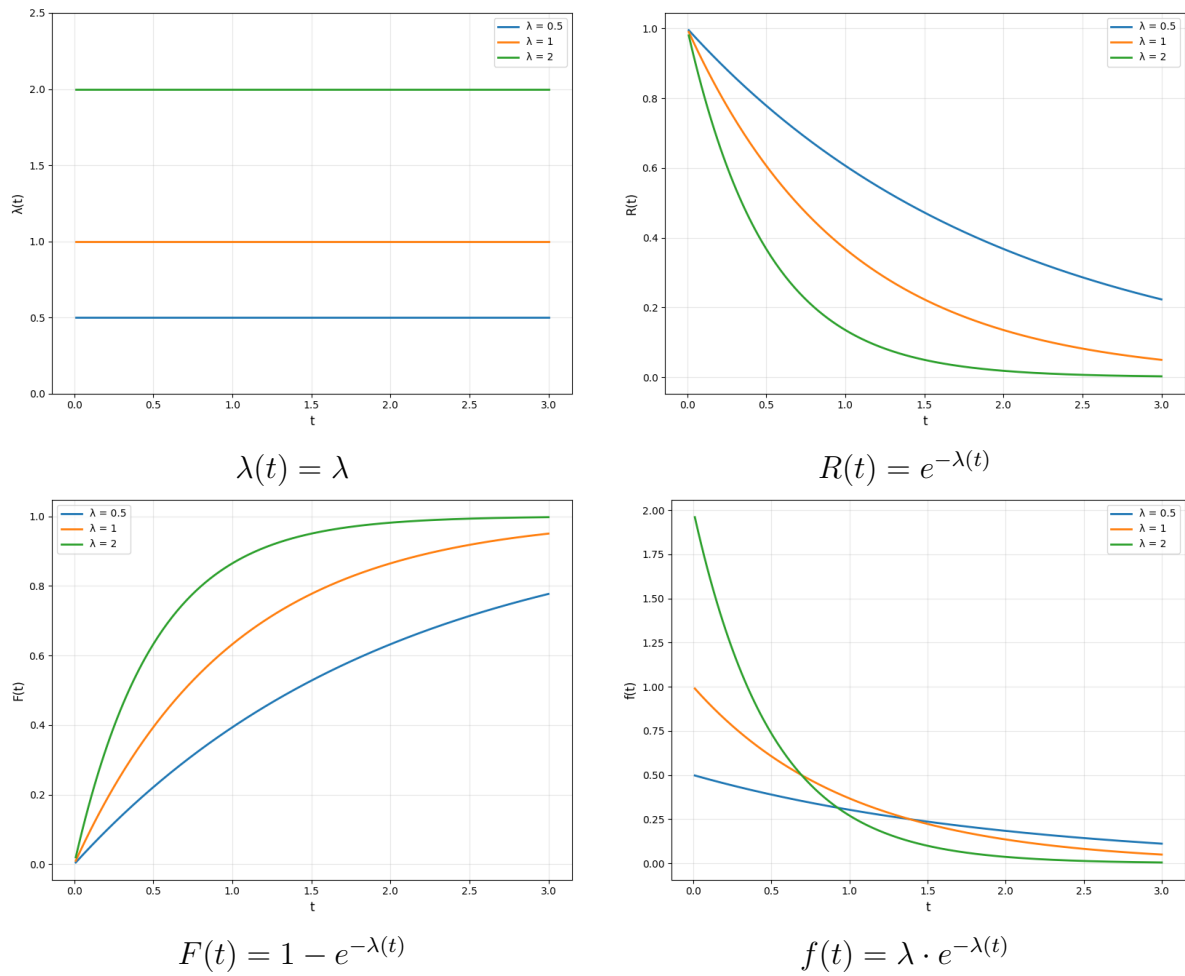


Table 2.1: Exponential distribution

But it is not only the failure rate (and its variation) that describes the reliability of a

system, but also the other three functions discussed earlier, the reliability function, the failure probability function and the density function. In the case where the failure rate is constant, the time to failure is a probability variable with an exponential distribution. Table 2.1 shows the basic characteristics of the exponential distribution of probability variables, including formulas and plots of the failure rate, failure probability, reliability and density function. This is typical for electrical systems, or (looking back at the bathtub curve) for the mid-life of systems.

If the system is new or in a wear out phase, its failure rate may not be constant. In this case, we can use the Weibull distribution. Table 2.2 shows the characteristic functions of the Weibull distribution, including the formulas for the failure rate, failure probability, reliability and density functions and their graphs.

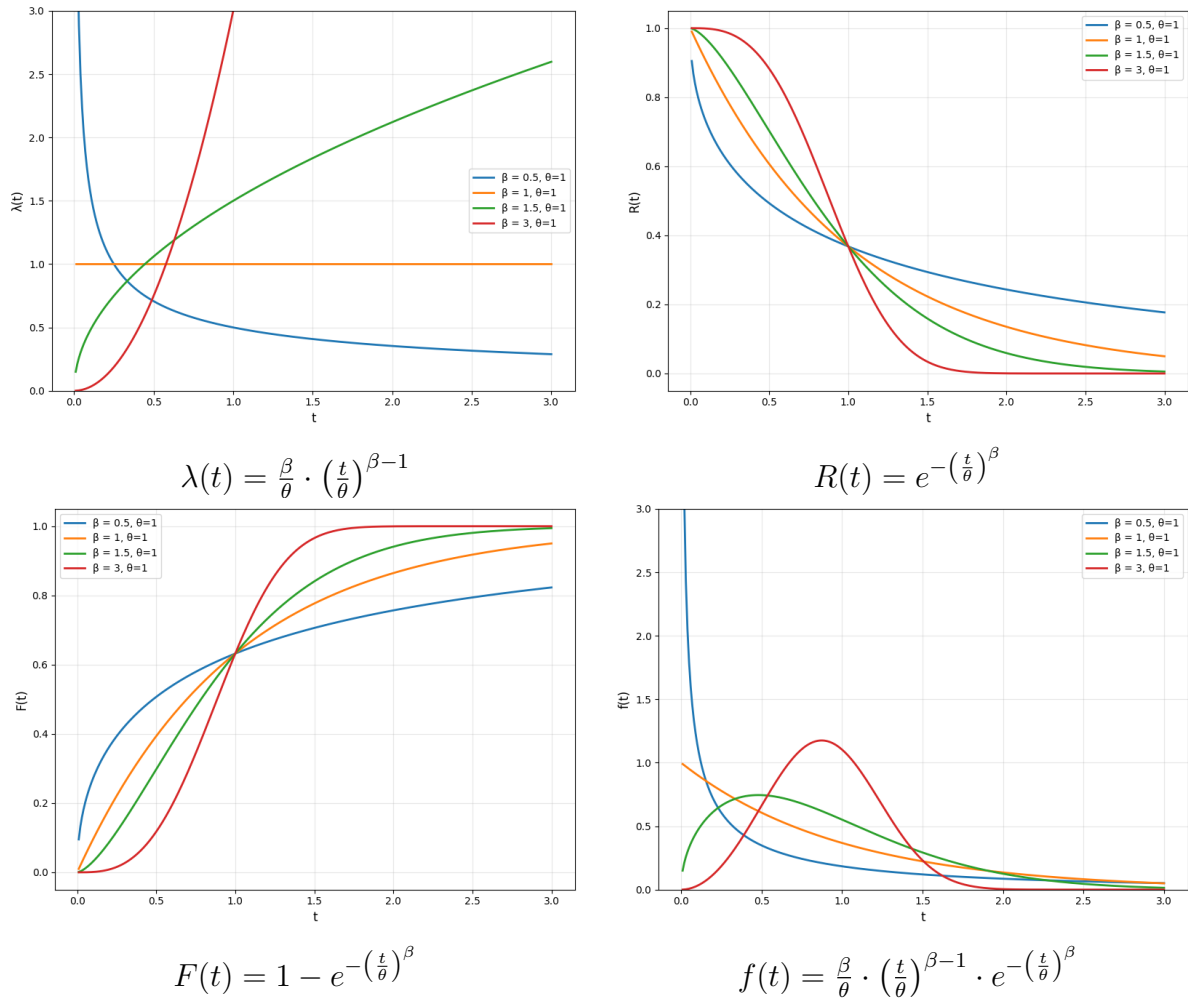


Table 2.2: Weibull distribution

In the equations in Table 2.2 θ is scale parameter, and β is shape parameter. If $\beta < 1$, this means that the frequency of failures decreases over time. If $\beta = 1$, then the distribution is identical in all respects to the exponential distribution, with the observed failure rate being shaped only by random failures. If $\beta > 1$, then the observed period falls

within the machine wear period, with the frequency of failures increasing with time. In the three-parameter version of the Weibull distribution, it is possible to shift the entire curve on the t -axis, to fit the data to the time shift.

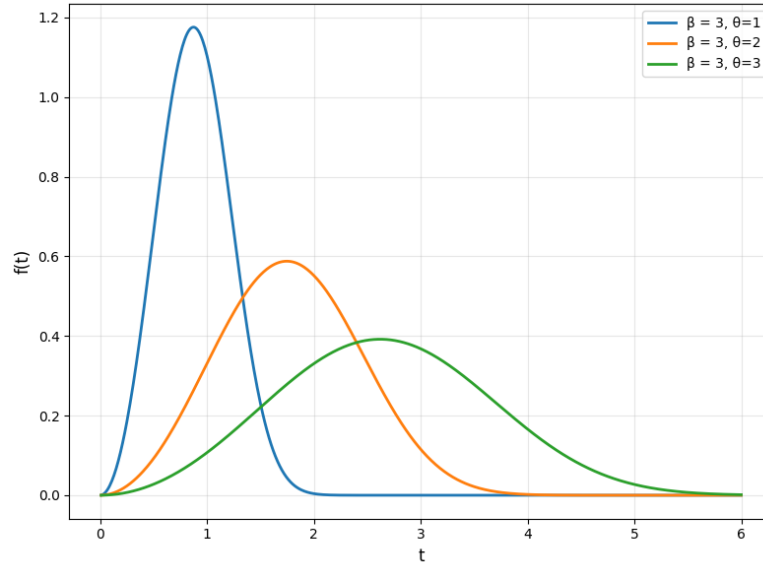


Figure 2.4: Impact of the scale parameter on the probability density function

The scale parameter determines how much the functions are stretched on the t -axis, in other words, the time scale on which the process takes place. For higher values of θ the function changes more slowly as a function of time, while lower values of θ result in a faster progression of the function, as shown in Figure 2.4.

2.4 Genetic Algorithms

There are many different types of optimization algorithms, each of them useful in different cases. GA is a population-based optimization algorithm, and within it, an evolutionary algorithm [9]. GA can be useful when a problem is too complex to find and evaluate all possible solutions individually, as it would take too much time. Instead, we model biological evolution to obtain a near-optimal solution with a relatively small search cost.

Genetic algorithms can be very diverse, but they all have the following steps in common [9] [10]:

1. find an appropriate genome representation
2. generate a population
3. evaluate the fitness of the individuals
4. select the better ones

5. crossover solutions to get new ones
6. mutate some of them
7. repeat the last four steps until the stop condition is met

Term	Explanation
Population	A set of solutions
Generation	A set of solutions in one iteration
Individual	A single solution
Genotype	Encoded solution
Phenotype	Decoded solution
Genome	The whole encoded solution
Gene	One value in the genome
Fitness value	A value shows how good is the solution

Table 2.3: Explanation of Genetic Algorithm terms

To make it easier to understand, I have summarized the terms used in GA and their explanations in Table 2.3.

Take a classic example, the knapsack problem. I have objects, each with value and weight. My task is to select them and pack them in a bag in such a way that the selected objects together have the highest possible value but do not exceed the weight limit of the bag. A possible solution is an individual in the population. The phenotype of an individual is the list of things it has in its bag, but in order to perform operations on individuals, we need to encode it in some way. This coded form becomes the genotype of the individual.

There are many ways to encode our solution, the most common are the binary vector, a vector of real or integer numbers, and permutation [10]. There are other ways to encode the possible solutions to suit the problem, the key is to choose a genetic representation that allows you to retrieve the phenotype at the end of the process and to perform all the necessary operations on the individual. In our case, a binary vector might be appropriate. Each object corresponds to a bit in the vector, and depending on whether the object is in the bag or not, the bit is set to one or zero. Then we generate individuals (in our case random bit vectors), which will form also the initial generation.

To find out how good the solutions are, we need to determine their fitness value [10]. In our case, this could be the sum of the value of the items if we are below the weight limit and zero if we have exceeded it. Here the higher the fitness value of an individual, the better the individual, the closer it is to the optimum. This is not a general rule either, it is possible to write a fitness function where the lower the value the better the solution.

The higher the fitness value of an individual, the better it is, and the closer it is to

optimal. This is not a general rule, as it is also possible to define a fitness function where a smaller value represents a better solution.

We then select the parents. There are several ways of doing this [11]: some are to select only the best individuals (truncation selection), and others are to select a few individuals at random from the whole population and select the best individual in this small group (tournament selection). The latter is obviously more complicated than the former, but we have to be careful that if we always select only the best individuals, the diversity of the population may be lost and the algorithm will run into premature convergence [9]. This means that individuals will be too similar before they are close enough to the optimal solution, but it can also be said that the algorithm is stuck in a local optimum [10].

The selected parents are then crossed to create offspring. In our case, we can do this by cutting the bit vector of the parents at a random position and then swapping them. The way of crossover depends very much on the genotype: a bit vector should be crossed differently than a permutation, but there can be different crossover methods for the same encoding form [12].

In order to maintain diversity, some of the newly created individuals are mutated, and how this is done depends mainly on genome representation. In our case, we can do this by setting a random position of a bit to its opposite. We keep creating more and more generations until the stopping condition is met. This can be reaching a maximum number of generations, reaching a maximum run time, approaching a predefined value, or even not changing the best individual during a given number of generations.

To further refine the algorithm, elitism can be applied. In this case, the best individual(s) are always passed on to the next generation, ensuring that the best individual in generation $n + 1$ is in no case worse than the best individual in generation n . In fine-tuning the algorithm, it is important to properly adjust parameters such as population size, mutation rate, and other rates that are related to a particular method, such as migration rate.

In summary, genetic algorithms are great optimization techniques inspired by natural evolution. Through the processes of selection, crossover, and mutation, these algorithms iteratively improve candidate solutions, converging toward optimal or near-optimal results for a wide range of complex problems. Despite their reliance on stochastic operations, genetic algorithms are capable of exploring diverse solution spaces and escaping local optima, making them suitable for problems where traditional methods may fall.

Chapter 3

Developed algorithm

The objective of the developed algorithm is to create an optimal maintenance schedule that determines when (in what hour) and to what degree (perfect or imperfect maintenance) the selected tool should be maintained in order to minimize the risk, taking into consideration the risk of under-maintenance and over-maintenance.

Figure 3.1 shows the stages in the development of the algorithm, showing how each component contributes to the final result.

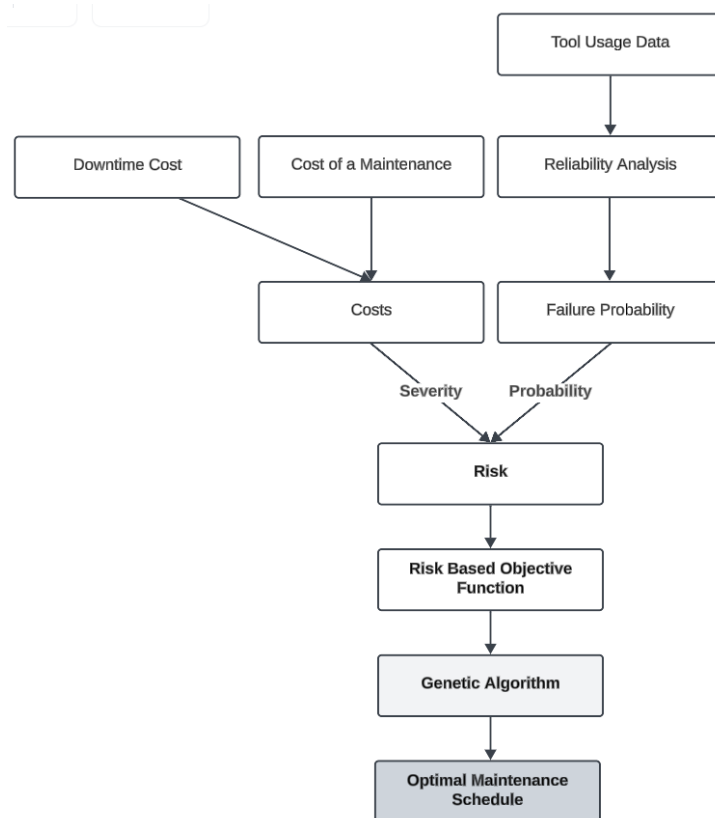


Figure 3.1: Overview of the process

To achieve this objective, I use three types of maintenance models: reactive maintenance, and within preventive maintenance, perfect and imperfect maintenance.

Reactive maintenance is performed following a failure. Once this maintenance has been performed, the reliability of the device after the maintenance will be the same as before the failure (as bad as old - ABAO). We can use the simple formula of the failure rate in Weibull-distribution:

$$\lambda(t) = \frac{\beta}{\theta} \cdot \left(\frac{t}{\theta}\right)^{\beta-1} \quad (3.1)$$

Perfect maintenance is performed on a preventive principle. When this maintenance is carried out, reliability is restored to as good as new (AGAN). We can use Renewal Process (RP) model [13]:

$$\lambda_t = \lambda(t - T_{N_t}) \quad (3.2)$$

Imperfect maintenance is also done preventively, but when this maintenance is done, reliability only increases to a certain extent, somewhere between ABAO and AGAN. We can use the Arithmetic Reduction of the Intensity (ARI) model [13]:

$$\lambda_t = \lambda(t) - \rho \sum_{j=0}^{N_t-1} (1 - \rho)^j \lambda(T_{N_t-j}) \quad (3.3)$$

In the Equation 3.2 and 3.3 t is time, T is a list of the maintenance times, N_t is index of the last maintenance before t in T , so T_{N_t} is time of the last maintenance before t . In the Equation 3.3 ρ is rate between 0 and 1

It is important to note that the above two models assume that $T_0 = 0$, so the first maintenance is performed at time zero, and the models do not take into account the time of maintenance, or assume that maintenance is performed immediately after a failure.

To get an optimal risk I combined the RP and the ARI models:

$$\lambda_t = \lambda(t - \hat{T}_0) - \rho \sum_{j=0}^{\hat{N}_t-1} (1 - \rho)^j \lambda(\hat{T}_{\hat{N}_t-j} - \hat{T}_0) \quad (3.4)$$

In the Equation 3.4 $\hat{T}$ is list of the maintenance times starting with the last perfect maintenance before t , $\hat{N}_t$ is index of the last maintenance before t in $\hat{T}$, so $\hat{T}_0$ is last perfect maintenance before t in T and first maintenance in $\hat{T}$, and $\hat{T}_{\hat{N}_t}$ is last maintenance

before t in $\hat{T}$.

What I actually do here is that, as in the RP model, I make the perfect maintenance the initial time, and subtract the elapsed time between the actual initial time and the last perfect maintenance from t , so that the failure rate is reset to zero after a perfect maintenance.

The models discussed earlier provide a formula for calculating the failure rate, from which we can obtain the reliability as follows:

$$R(t) = e^{-\lambda(t)} \quad (3.5)$$

A comparison of the failure rates and reliabilities plotted by the different models is shown in Figure 3.2.

To account for the risk of over- and under-maintenance, I need to consider the severity and probability of these two events occurring.

Over-maintenance occurs when, afraid that the equipment will fail, we maintain it too often, which leads to high costs. These costs in the algorithm are made up of two parts: the cost of maintenance and the cost of loss of production. The former is determined by the cost of the form of maintenance performed (imperfect is lower and perfect is higher), and the latter by the product the tool would be producing at the time of maintenance. Since the planned maintenance will most certainly be carried out, every maintenance event is multiplied by 1.

Under-maintenance occurs when, at the expense of the cost of maintenance, the tool is allowed to deteriorate, risking sudden loss of production and more costly maintenance. Here again, the cost of maintenance and the cost of loss of production must be taken into account, but since the failure event is not certain, the cost must be multiplied by the probability of the tool failing at a given time.

$$\text{Downtime cost per hour} = \text{price} \cdot \frac{\text{quantity}}{\lceil \text{production time} \cdot \text{quantity} \rceil} \quad (3.6)$$

To get severity to a given hour:

$$s = \text{Downtime cost per hour} \cdot \text{maintenance duration} + \text{maintenance cost} \quad (3.7)$$

To get probability to a given hour in case of scheduled maintenance:

$$p = 1 \quad (3.8)$$

To get probability to a given hour in case of production:

$$p = F(t) = 1 - R(t) = 1 - e^{-\lambda(t)} \quad (3.9)$$

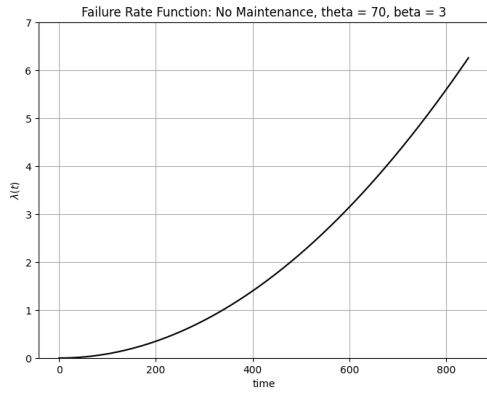
To get the total risk of the maintenance schedule, we need to calculate the risk for every hour and add them together.

$$\text{Total Risk} = \sum p \cdot s \quad (3.10)$$

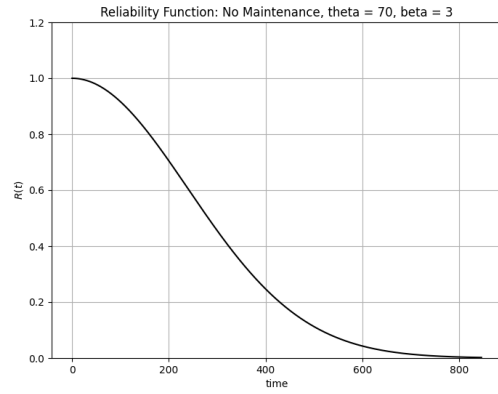
It is important to note that the algorithm assumes that all the tools needed to make a product must be available, that there are no spare tools (that means if a tool fails, the product will not be made), and that the tools are in constant use, which means the FMS always produces the products that require the observed tool.

In the GA the individual of the population is a solution for the scheduling problem for one tool. The genome of the solution looks like $[2, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 2, \dots]$ where 2 is coding perfect maintenance, 1 is coding imperfect maintenance and 0 means no scheduled maintenance, but there is could be reactive maintenance if tool fails.

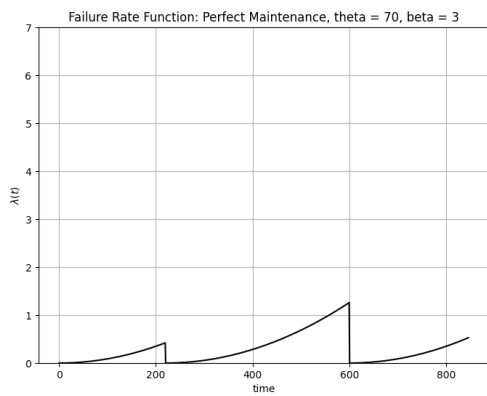
It is important to note that we always start the genome list with 2, because the model assumes that $T_0 = 0$, so the first maintenance is performed at time 0. Even the mutation function could not modify the first element of the genome. The fitness value of the genome is the risk of the solution, so the fitness function calculates the total risk. That means that the lower the fitness value, the better the solution. In this way, the GA gives us a near-optimal solution, a maintenance scheduling for a single tool.



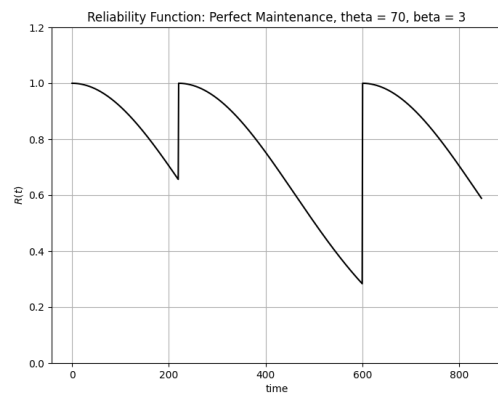
(a) Failure rate: no maintenance



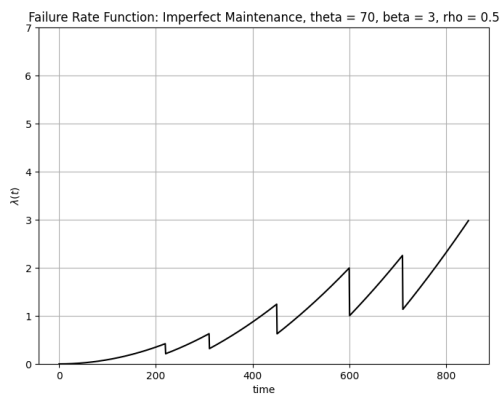
(b) Reliability: no maintenance



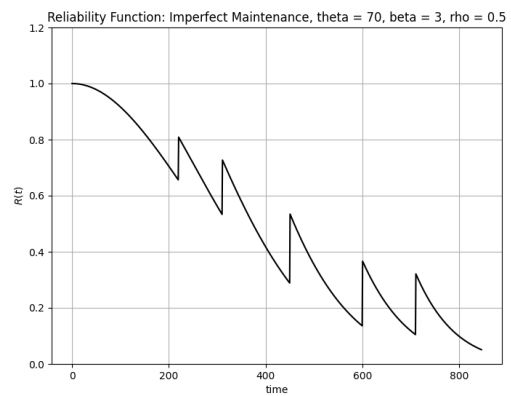
(c) Failure rate: perfect maintenance



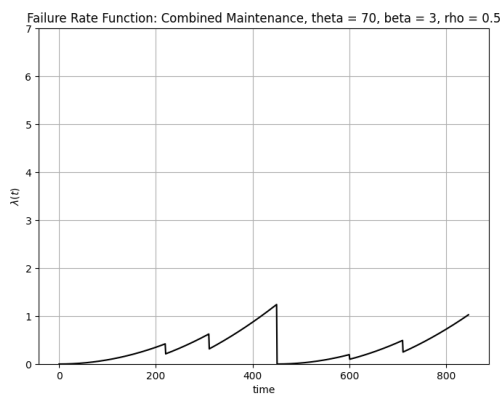
(d) Reliability: perfect maintenance



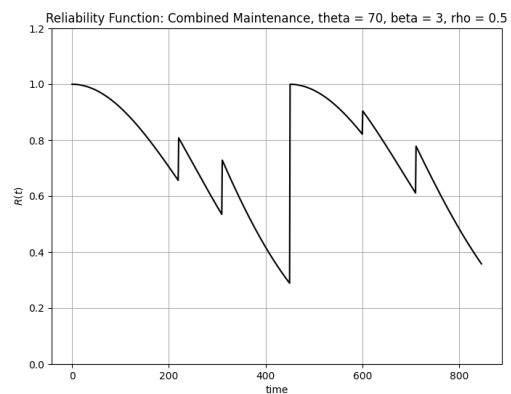
(e) Failure rate: imperfect maintenance



(f) Reliability: imperfect maintenance



(g) Failure rate: combined maintenance



(h) Reliability: combined maintenance

Figure 3.2: Failure rate and reliability with different models

Chapter 4

Case study and results

In this case study, I created a simplified model of an FMS in a Python environment and tested the algorithm on a selected device of the model. The data required for this are mainly based on the literature and arbitrary data, and the algorithm has not yet been tested on real industrial data.

4.1 Description of the case study and parameters

To model the FMS, I first created classes as shown in Figure 4.1, then created objects. The **product** object is the products to be produced, the **tool** objects are the tools needed for production, and the **cycle** objects represent the production cycles in which a given product is produced in a given quantity.

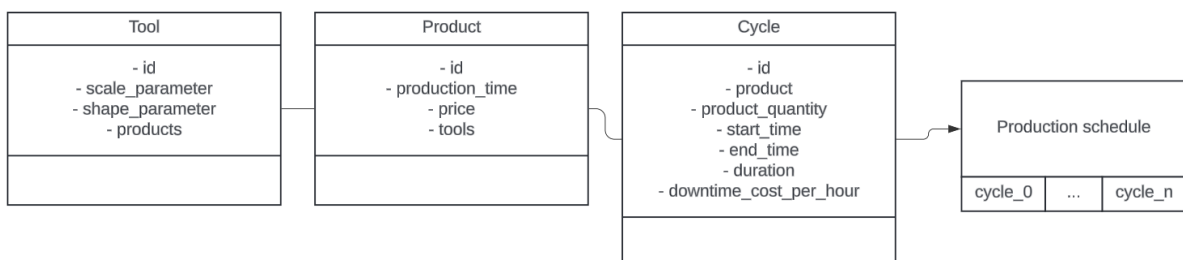


Figure 4.1: Class diagram of the created classes

After defining the price and the duration of production of the example products, I assigned them the tools needed to produce them. Then, I assigned to the cycle objects the product and the quantity to be produced. Then I created a production schedule, a list of production cycles in order, and in this way I was able to define the start and end times of the cycles, which plays an important role in determining the downtime cost. Based

on the example duration of the cycles, the maintenance schedule in this case study has a nearly 1000 hour length, 987 hours, to be exact.

It is also important to note that the tool objects have shape- and scale parameter data, which I manually entered during the case study, but would be obtained from the reliability analysis of the previous production data when working with real data. The functions and parameters used for the GA are described in Table 4.1.

Parameter	Description	Value
Population Size	Number of individuals in each generation	100
Mutation Rate	Probability of mutation occurring per gene	0.01
Tournament Size	Number of individuals in each tournament selection	6
Elitism	Number of top individuals retained each generation	1
Stagnation Limit	Generations without improvement to stop criterion	3000

Table 4.1: GA Parameters

I generate the individuals so that the first gene of the individual is always two, means perfect maintenance, for the reasons explained in the previous chapter, and the other genes are zeros. I do it this way because a gene represents an hour, and maintenance events on such a small scale are not densely spaced, so it takes the algorithm a long time to reduce the number of maintenance events to a near-optimal number, it takes a much shorter time for them to appear in the right place and at the right rate by mutations. I also tried shifting the default ratio of 0.33-0.33-0.33 up to 0.9-0.05-0.05, but starting with zero turned out to be the best.

I created the fitness function as described in the Developed Algorithm chapter, where the function goes through all the genes of the individual and depending on whether the individual proposes maintenance for the given hour and if so, what kind of maintenance; it calculates the risk of the proposal and the total risk of the solution becomes the fitness value of the individual. I use tournament selection in the selection function because after trying truncation and roulette wheel selection, this proved to be the most efficient. In this way, I randomly select 6 individuals from the population and the best of them becomes the parent. This allows the algorithm to converge relatively quickly while maintaining the diversity of individuals.

Because the lengths of the individuals can vary depending on the time span for which we want to create a maintenance schedule, I have discarded the multipoint crossover and instead use the uniform crossover, since it does not cut and match the two parents at a given number of points but decides for each gene which parent the value should come from. This creates two offspring. In the same position, one of them gets the gene from parent 1, and the other gets from parent 2. In the mutation function, I use a version of

bitflip mutation, which can work with three-valued digits and randomly chooses between the other two values. I do this for a gene when a random number falls below the mutation rate. The stopping condition of a GA is that if the best individual is unchanged for a given number of generations, then the algorithm stops. The stopping parameter is the stagnation limit. For this to work, I need to apply elitism, that means I move the best element of a given number of generations to the next generation. This ensures that the next generation does not have worse individuals than the current one.

4.2 Results

The example data I ran the algorithm on are shown in the appendix. The length and cost of maintenance are shown in Table 6.1, the shape and scale parameters for the tools are shown in Table 6.2, the price, production time and tools needed to make each product are shown in Table 6.3, and the products produced during the production cycles and their quantities are shown in Table 6.4.

I ran the GA 20 times and compared the best individuals. The results of the 20 runs are shown in Table 6.5 in the appendix. The fitness value of the best individual from 20 runs is 11 076 302 cost units, and Figure 4.2 shows the changes in reliability of the selected tool using the example production data and the maintenance schedule of the best individual.

In order to make sure that the combined model does indeed provide a better solution than either perfect or imperfect maintenance, I ran the algorithm using the RP and ARI models in the fitness function instead of the combined one. The results of the runs are shown in Figure 4.4 and 4.3. For both solutions, their fitness values are higher than the worst value obtained in the 20 runs with the combined model, means the combined model can indeed produce a better schedule than the RP and ARI models separately. The numerically summarized results are shown in Table 4.2.

Individual description	Fitness value
Combined model: best individual of 20 running	11 076 302 cost units
Combined model: worst individual of 20 running	11 929 496 cost units
RP model: just perfect repairs	12 633 704 cost units
ARI model: just imperfect repairs	14 761 373 cost units

Table 4.2: Comparison of the RP, the ARI and the combined model

The great advantage of the combined model is that it takes into account the production schedule, therefore the varying severity, and adapts the maintenance plan to the tool

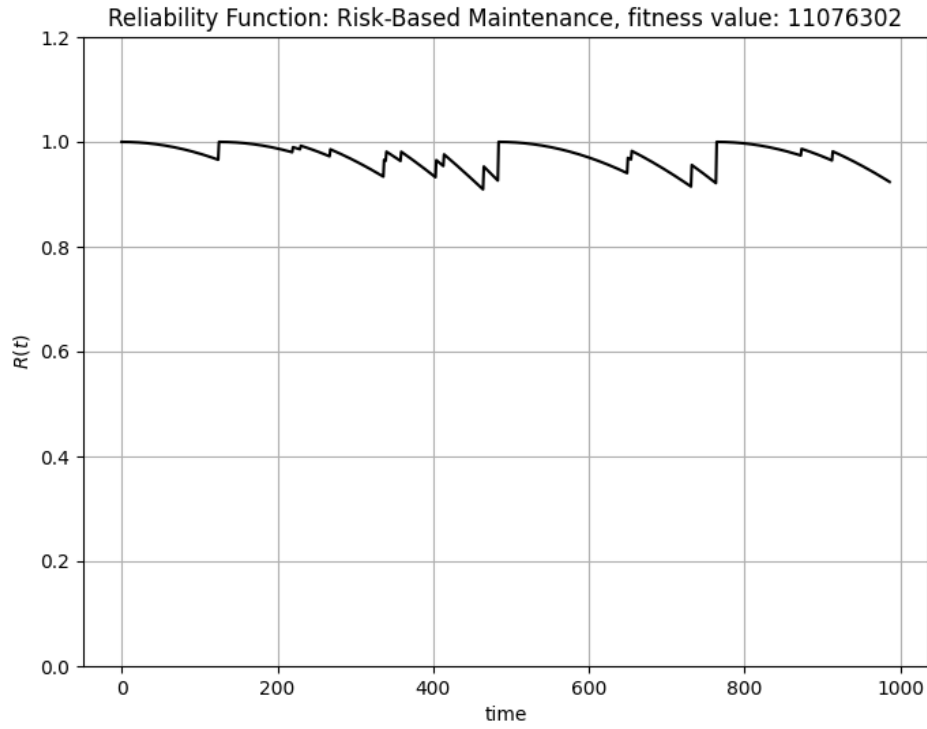


Figure 4.2: Reliability of tool 0 with example production data and maintenance schedule of the best individual

priorities resulting from the production needs. This flexibility is reflected in the fact that, if the downtime cost caused by perfect maintenance would be too high, it allows to postpone it to a cycle with a lower product cost, resulting in a lower downtime cost. This can be observed in Figure 4.2. For example, around the 800th hour, the model suggests first an imperfect and then a perfect maintenance. This is probably because cycle 5 ends at hour 769 and cycle 6 starts at hour 770. This is important because the unit price of the product produced in cycle 5 is higher than the unit price of the product produced in cycle 6, so the downtime cost is higher in cycle 5 than in cycle 6. Thus, perfect maintenance in cycle 6 has a lower cost.

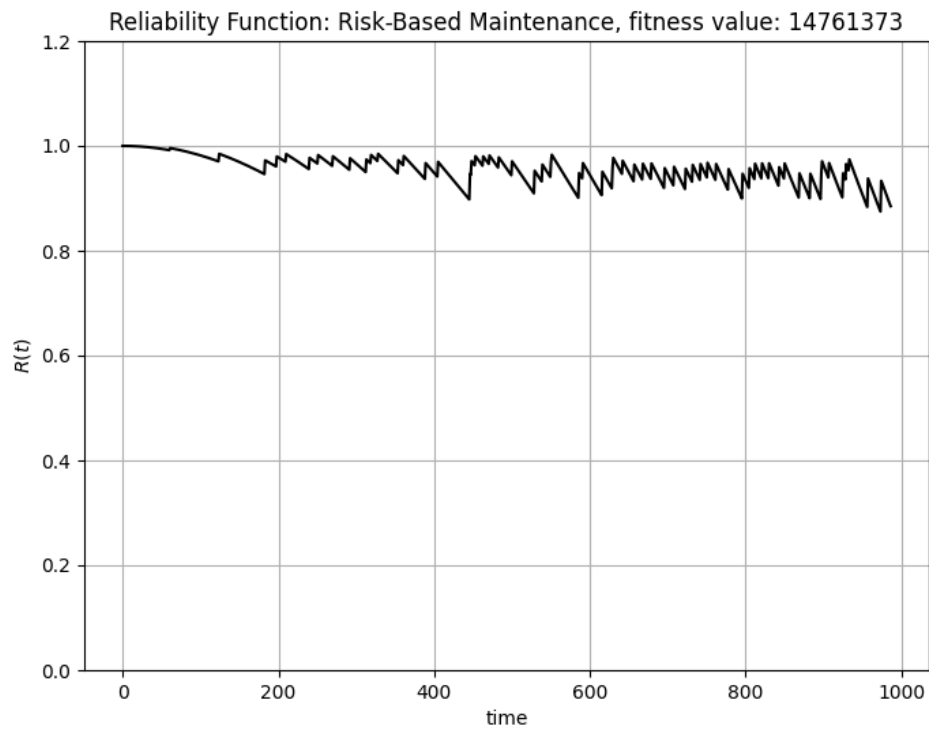


Figure 4.3: Reliability of tool 0 if we use the ARI model

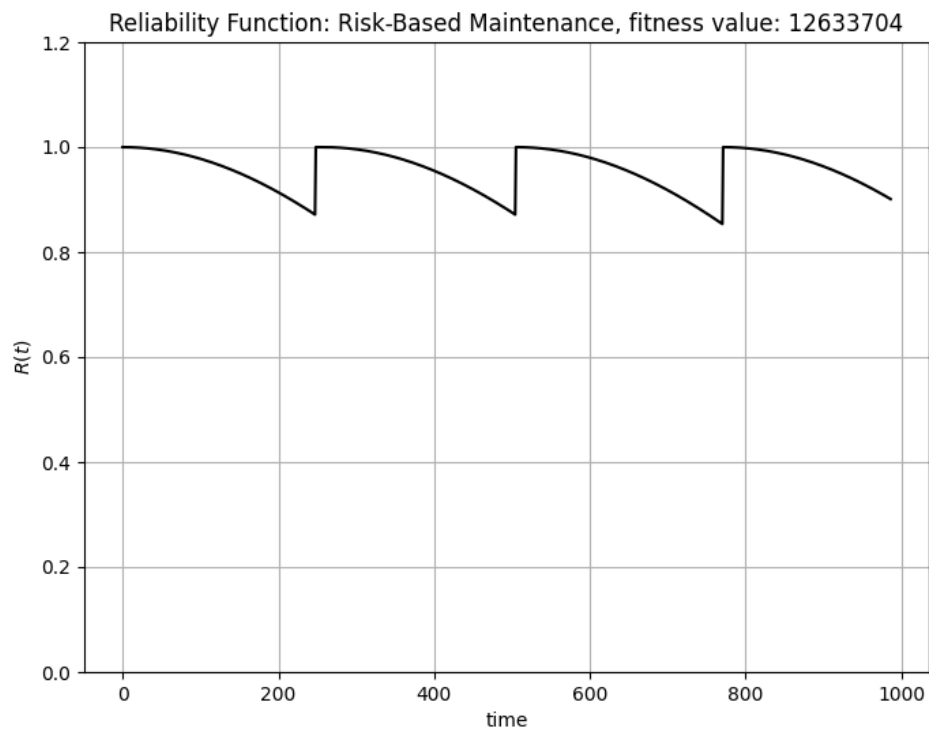


Figure 4.4: Reliability of tool 0 if we use the RP model

Chapter 5

Conclusion

Flexible manufacturing systems have several advantages, but the disadvantage is that they are more difficult to maintain properly than traditional systems, as tool usage, wear and tear and condition vary greatly due to the flexible scheduling. Therefore, in this thesis, instead of looking at the system as a whole, I take advantage of the fact that the tools are unique and simple, and create a maintenance schedule for each tool individually.

In order to minimize the risk of both under-maintenance and over-maintenance, I have combined two models. The Renewal Process (RP) model is used when the asset is restored to its as-new condition at each maintenance, and the Arithmetic Reduction of the Intensity (ARI) model is used when only improving reliability but not restoring the asset to its as-new condition. Perfect maintenance takes longer and costs more, but imperfect maintenance needs to be performed more often to ensure the same reliability. However, if we use both perfect and imperfect maintenance, we can benefit from both types of maintenance and have less overall risk than if you just use one or the other model, as demonstrated by testing on example data.

Acknowledgments

I would like to thank my supervisors, Dr. Tamás Ruppert and András Darányi for their help, guidance and patience. Without them this thesis would not have been possible.

This research was supported by the Ministry of Culture and Innovation of Hungary from the National Research under the 2020-1.2.3-EUREKA-2022-00021 project.

Bibliography

- [1] Mehdi Kaighobadi and Kurapati Venkatesh. Flexible manufacturing systems: an overview. *International Journal of Operations & Production Management*, 14(4):26–49, 1994.
- [2] Peter Kostal and Karol Velisek. Flexible manufacturing system. *International Journal of Industrial and Manufacturing Engineering*, 5(5):917–921, 2011.
- [3] Yanhu Pei, Zhifeng Liu, Jingjing Xu, Baobao Qi, and Qiang Cheng. Grouping preventive maintenance strategy of flexible manufacturing systems and its optimization based on reliability and cost. *Machines*, 11(1):74, 2023.
- [4] Deke Liu Jr. Application of risk based inspection (rbi), reliability centered maintenance (rcm) and risk based maintenance (rbm). Master’s thesis, University of Stavanger, Norway, 2013.
- [5] Ranganath Kothamasu, Samuel H Huang, and William H VerDuin. System health monitoring and prognostics—a review of current paradigms and practices. *The International Journal of Advanced Manufacturing Technology*, 28:1012–1024, 2006.
- [6] Amitkumar Patil, Gunjan Soni, Anuj Prakash, and Kritika Karwasra. Maintenance strategy selection: a comprehensive review of current paradigms and solution approaches. *International Journal of Quality & Reliability Management*, 39(3):675–703, 2022.
- [7] Faisal I Khan and Mahmoud M Haddara. Risk-based maintenance (rbm): a quantitative approach for maintenance/inspection scheduling and planning. *Journal of loss prevention in the process industries*, 16(6):561–573, 2003.
- [8] Bernd Bertsche. *Reliability in automotive and mechanical engineering: determination of component and system reliability*. Springer Science & Business Media, 2008.
- [9] Manoj Kumar, Dr Mohammad Husain, Naveen Upreti, and Deepti Gupta. Genetic algorithm: Review and application. *Available at SSRN 3529843*, 2010.

- [10] Tom V Mathew. Genetic algorithm. *Report submitted at IIT Bombay*, 53, 2012.
- [11] Tobias Blickle. Tournament selection. *Evolutionary computation*, 1:181–186, 2000.
- [12] Anant J Umbarkar and Pranali D Sheth. Crossover operators in genetic algorithms: a review. *ICTACT journal on soft computing*, 6(1), 2015.
- [13] Laurent Doyen and Olivier Gaudoin. Classes of imperfect repair models based on reduction of failure intensity or virtual age. *Reliability Engineering & System Safety*, 84(1):45–56, 2004.

Chapter 6

Appendix

Maintenance type	Cost	Length
Perfect	1 000 000 cost units	5 hours
Imperfect	100 000 cost units	1 hours
Reactive	110 000 cost units	1 hours

Table 6.1: Maintenance costs and durations

Tool ID	shape parameter	scale parameter
tool 0	3	110
tool 1	2.9	115
tool 2	3.1	109
tool 3	3.8	101

Table 6.2: Shape and scale parameter of the tools

Product ID	Price	Production time	Needed tools
product 0	2000 cost units	1/60 hours	tool 0, tool 1
product 1	3000 cost units	2/60 hours	tool 0, tool 1, tool 2
product 2	1500 cost units	5/60 hours	tool 0, tool 2
product 3	2100 cost units	3/60 hours	tool 0, tool 3
product 4	2500 cost units	4/60 hours	tool 0, tool 2, tool 3

Table 6.3: Price and production time of the products

Cycle ID	Product	Quantity
cycle 0	product 0	2100 pieces
cycle 1	product 1	3000 pieces
cycle 2	product 0	1500 pieces
cycle 3	product 1	2100 pieces
cycle 4	product 2	4000 pieces
cycle 5	product 4	3000 pieces
cycle 6	product 3	1500 pieces
cycle 7	product 4	2100 pieces

Table 6.4: Cycles with the required product and quantity

Fitness value		
1	11 076 302	cost units
2	11 085 435	cost units
3	11 116 053	cost units
4	11 157 081	cost units
5	11 222 179	cost units
6	11 285 264	cost units
7	11 288 880	cost units
8	11 315 023	cost units
9	11 349 674	cost units
10	11 536 928	cost units
11	11 557 239	cost units
12	11 570 421	cost units
13	11 605 677	cost units
14	11 703 470	cost units
15	11 766 813	cost units
16	11 766 972	cost units
17	11 776 396	cost units
18	11 860 892	cost units
19	11 864 149	cost units
20	11 929 496	cost units

Table 6.5: Fitness values of the best 20 solutions running with the combined model, sorted in ascending order