NTNU
Norwegian University of
Science and Technology

Coupling AltaRica models with optimization heuristics for the assessment of maintenance policies

Huweiyang Jin

Reliability, Availability, Maintainability and Safety (RAMS)
Submission date: June 2018
Supervisor: Antoine Rauzy, MTP

Norwegian University of Science and Technology
Department of Mechanical and Industrial Engineering

RAMS

Reliability, Availability,
Maintainability, and Safety

Coupling AltaRica models with optimization heuristics for the assessment of maintenance policies

Huweiyang Jin

June 2018

Master Thesis

Department of Mechanical and Industrial Engineering

Norwegian University of Science and Technology

Supervisor : Antoine Rauzy

Preface

This master thesis was written during the spring semester 2017 at the Department of Mechanical and Industrial Engineering, Norwegian University of Science and Technology (Trondheim). It is part of the two-year international master's program in RAMS (Reliability, Availability, Maintainability and Safety). The topic is based on the specialization project done in the autumn semester 2017 titled " Coupling AltaRica models with optimization heuristics for the assessment of maintenance policies". In this thesis, simulation tools from AltaRica project are used instead of python script. The thesis is supervised by Professor Antoine Rauzy. It is written for readers with basic understanding in the fields of reliability analysis.

Huweiyang Jin

Trondheim 2018-06-08
Huweiyang Jin

Acknowledgment

Thanks to my supervisor professor Antoine Rauzy for his help. He helps me very much with the learning of computer science technologies, which I lack from my background.
Thanks to my classmate and friend Guilherme Vale for his support.

H.J

Abstract

The main objective of this thesis is do maintenance optimization for a system using AltaRica simulation tools. Maintenance optimization is based on the assessment of maintenance policy. Stochastic simulation is applied to assessment the performance of the maintenance policy.

The simulation process consists of several steps. First, build the AltaRica model. Second, compile the AltaRica model using compiler from AltaRica project. Third, prepare indicator description file and generate the simulator. Fourth, prepare mission description file and execute the simulator. After this process, the simulation is conducted and the result will be stored in a csv file.

Maintenance optimization is based on the simulation process. Among a large number of candidate solutions for maintenance policy, metaheuristic algorithm can lift the searching efficiency for the optimum solution. Some algorithms help to skip unnecessary simulation and avoid repeated work.

The combination of AltaRica simulation tools and metaheuristic algorithm realizes the efficient automation of maintenance optimization process. This project makes the connection of these two parts and achieves the objective.

Contents

Chapter 1

Introduction

1.1 The AltaRica 3.0 Project

In safety analysis, there are two categories of formalisms.

- Boolean formalisms: Fault Trees, Event Trees, Block Diagrams…
- Transition Systems: Markov Chains, Stochastic Petri Nets…

Virtual experiments are extremely resource consuming

- Approximations
- Tradeoffs accuracy of models/ability to perform computations

Classical formalisms of safety analyses stand at a very low level

- Distance between system specifications and models
- Models are hard to design and even harder to maintain throughout the life cycle of systems

AltaRica promises to model systems at higher level so to reduce the distance between systems and models, without increasing the complexity of calculations.
The features of AltaRica is formal, event-based, textual & graphical, with multiple assessment tools. (Rauzy, 2014)

AltaRica 3 is the latest evolution of event based modeling language AltaRica. In this language, the state of the system is described by means of variables, so, the modification of the system state can only happen when its variables values change. Also, the value of these variables can only change when an event is triggered. Events can be associated with deterministic or stochastic delays. Models consist of hierarchical components interfaced in a discrete event system: their inputs and outputs can be connected and their transitions can be synchronized.(Mortada et al., 2014)

1.2 Background

AltaRica performs well in safety assessment. However, the tools distributed in the AltaRica project should be executed in command line. AltaRica Wizard is a software designed by Antoine, which makes a graphical user interface for the users. This way of using AltaRica tools is limited to the current functions and methods. For different parameter data in a system, the configuration files of stochastic simulation need to be modified. In AltaRica Wizard, this modification needs to be done manually. Thus, the idea of simplizing the modification and making the process automatically arises.

Automation in the stochastic simulation process can make assessment efficient. Much repetitive work can be neglected. A good choice is to write scripts in python controlling the process. Python is a simple and minimalistic language. The pseudo-code nature of python allows user to concentrate on solutions rather than syntax. It is possible to embed Python within C/C++ programs to give 'scripting' capabilities for your program's users, which is essential in this project. The embeddable feature of python makes this project adjustable. The code can be modified according to the needs easily.

The combination of python and AltaRica makes it possible to use algorithms to lift the efficiency. The AltaRica tools are C++ programs with high efficiency. The efficiency of the process in python can be lifted by applying some certain algorithms, such as metaheuristic algorithm.

1.3 Objective

The objective of this project is to conduct maintenance optimization efficiently using AltaRica simulation tools. The main objective consists of two parts: One is to do stochastic simulation using AltaRica tools in python, the other is to do assessment and optimization in python applying algorithms.

Several main challenges are on the way to achieving the first part of the objective. First, the AltaRica tools are C++ programs, which need to be called in python. While the simulation tools are running, the python program has to wait for its ending and then make use of the result. Second, the values of parameter for different simulation changes. The configuration files need to be modified accordingly, which should be done in python synchronized.

For the second part of the objective, the application of algorithms is the main challenge. When the number of candidate solutions of maintenance policy is too large, the efficiency of optimization process is influenced. Using algorithm in a good way can help to skip some unnecessary simulations.

1.4 Structure of the thesis

Literature review
This chapter is a brief summary and abstract of related work.

Theoretical description
This chapter introduces the technologies used in this project.

Experiment
This chapter is the process of realization of the project.

Summary

Chapter 2

Literature review

Application of stochastic simulation

Stochastic simulation provides the estimated probability of possible scenarios of a certain system or event. It is widely used in industry and other areas, in which real experiments with the system is impossible or too pricy to conduct. General Electric Company uses stochastic simulation in their western wind and solar integration study.

The deepening penetration of intermittent renewable resources presents major challenges in power system planning and operations in light of their highly time-varying nature and their associated geographical and climatological sources of uncertainty. Indeed, unlike conventional resource outputs, wind and solar resource outputs cannot be controlled by the operator except to be curtailed. The high variability in wind speeds and insolation patterns, both temporal and spatial, results at times in intermittent wind and solar resource outputs. (Energy, 2010)

These complications illustrate the critical need to appropriately represent the temporal and spatial correlations of the wind and solar resource outputs in the assessment of the power system performance. Such need implies that the various renewable resource outputs, as well as the demands and conventional generation available capacities, must be modeled by random processes (r.p.s) so as to capture the impacts of their variability across time and space. These requirements drive the need for a comprehensive simulation tool that can effectively quantify the expected economic, reliability and emission variable effects of power system with integrated renewable resources.(Degeilh and Gross, 2015)

The uncertainty of geographical and climatological sources makes it hard to predict the timely situation. Stochastic simulation is used to predict the scenarios and possibilities. Of course, the model is never accurate, as all models are wrong. The simulation result provides some useful information for analysis.

The system model needs to be updated as the data collected during the work process. An automatic way of updating the model, mainly the degradation model, is machine learning. The main concept of machine learning here is to use the collected data to update the system model with some algorithm automatically. The status data is collected through continuous monitoring.

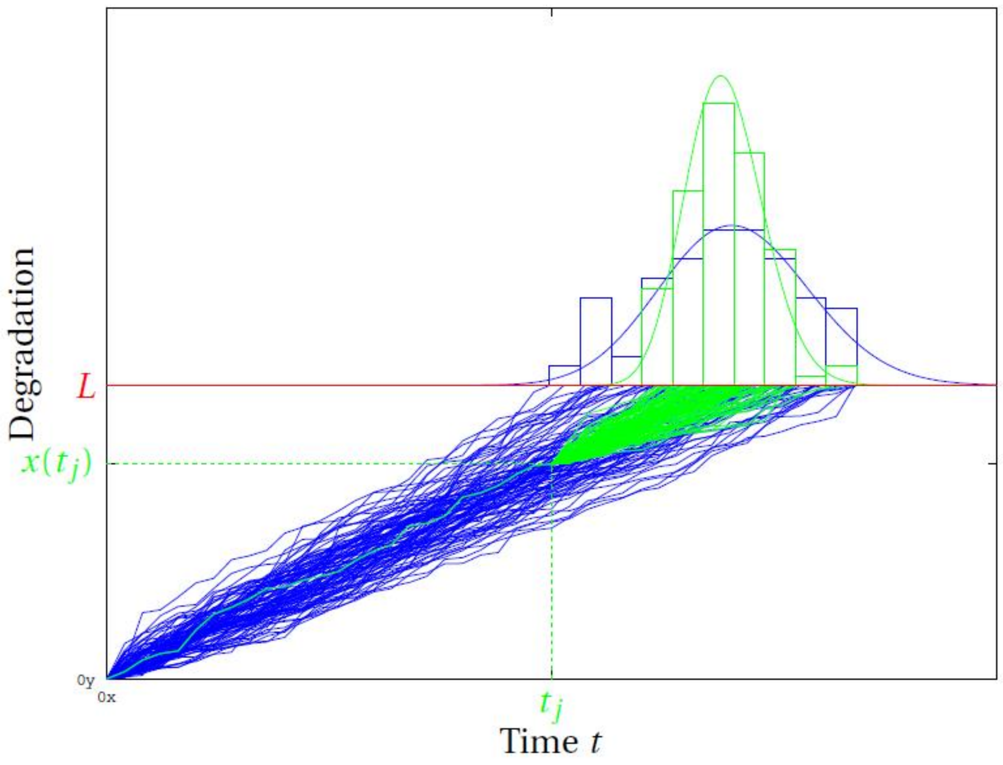


Figure 2.1 Machine learning (Barros, 2017)

As shown in the figure above, the degradation model is based on a lot of previous data. The algorithm learns from the previous failures and derives of a possible degradation trend for the current component and its status.

Machine learning can provide a timely updated degradation model, which makes the result of simulation more trustworthy.

Maintenance optimization

Given a certain cost and reward structure an optimal repair and replacement strategy will be derived. (Aven and Jensen, 1999) Maintenance optimization is to derive of the maintenance policy, which has the best performance considering reliability, production, etc.

Proper and safe operation is a goal for any electric power generation plant, especially one powered by nuclear fuel. An inefficiently functioning plant costs more to operate and produces less energy than it potentially could. Replacing or repairing items too often may substantially increase the cost of operating the plant. A decision-making rule that defines when and how an item is maintained is known as a maintenance policy.(Belyi et al., 2017)

In an industrial setting, planned and unplanned maintenance stops can have a significant impact on sustainability and short and long-term profitability. During a stop, fixed costs of equipment, real estate and labor remain constant while production is essentially zero. Maintenance is in itself also costly, and much of it is unnecessary and avoidable. Substantial efforts have therefore been invested in minimizing the expected total cost due to failures and preventive maintenance of industrial equipment. In industry, most preventive maintenance approaches include the use of fixed schedules, which are (more or less) optimized for minimum cost in advance. (Bohlin and Wärja, 2015)

The main information of maintenance policy is when to conduct what kind of maintenance.
A balanced maintenance schedule can keep the system reliability at a high level while keeping the maintenance cost at a reasonable number. The life cycle cost is shown as figure 2.2.

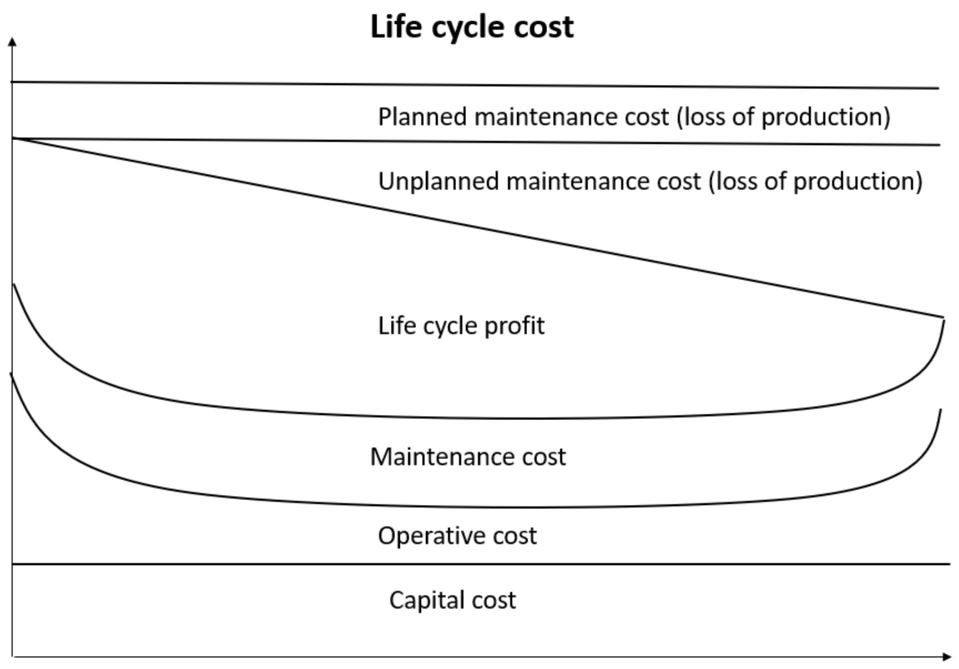

Life cycle cost
Planned maintenance cost (loss of production)
Unplanned maintenance cost (loss of production)
Life cycle profit
Maintenance cost
Operative cost
Capital cost

Figure 2.2 Life cycle cost

The maintenance cost is a large part in the life cycle cost. The maintenance cost varies from predictive and corrective maintenance. The best maintenance policy is to do preventive maintenance right before the system fails. It is impossible to be so accurate in the prediction. Thus, the target of optimization is to plan the preventive maintenance reasonably insuring stable reliability and low maintenance cost.

Nowadays, the maintenance policy in the industry is normally clock-based. The preventive maintenance is conducted at a fixed schedule. The drawback of such maintenance policy is waste of resources. A preventive maintenance may be performed right after corrective maintenance. A better management of the maintenance can contribute to the efficiency of maintenance and better usage of resources, including human resources.

As followed is a case of maintenance optimization of power systems, which proposes a quantitative maintenance optimization problem for developing reliability centered maintenance for a power system with renewable energy sources.

Reliability and cost are two important interlinked aspects considered by system operators in many deregulated power systems. Reliability centered maintenance is an effective method to consider both of these aspects when performing the maintenance optimization. Nevertheless, this method has not adequately studied for a power system with renewable energy sources included. First, the most critical components of the system are selected. Then, a set of maintenance strategies are proposed for all critical components. After that, the total cost of each maintenance strategy for all critical components are calculated as the summation of operation, maintenance, environmental, and interruption costs. Finally, the best maintenance strategy for each critical component is selected by identifying the lowest total cost of different maintenance strategies. (Shayesteh et al., 2018)

In this case, the performance of maintenance policy is based on calculation and formulas. This way of estimation has its own strengths. The formulas are tested and influencing factors are controllable. However, the fixed formula means lack of changes. Some factors may be neglected. Stochastic simulation is therefore a better solution than estimation with formulas. The main challenge of stochastic simulation is to build a model of the system and all the scenarios of the system can be developed from the model, even those which are not expected. The model can be updated and matured with the data collected through conditioning monitoring. The optimization process of the case above is done manually. As the number of possible maintenance policy is not large in that case, manual comparison is feasible. However, when the problem comes to a large system, even a factory, the complexity and difficulty of optimization rises exponentially. For a large system and a large number of candidate maintenance policies, it is not wise to perform it manually.

Optimization algorithm

To solve optimization problem, efficient search or optimization algorithms are needed. There are many optimization algorithms which can be classified in many ways, depending on the focus and characteristics.

If the derivative or gradient of a function is the focus, optimization can be classified into gradient-based algorithms and derivative-free or gradient-free algorithms. Gradient-based algorithms such as hill-climbing use derivative information, and they are often very efficient. Derivative-free algorithms do not use any derivative information but the values of the function itself. Some functions may have discontinuities or it may be expensive to calculate derivatives accurately, and thus derivative-free algorithms such as Nelder-Mead downhill simplex become very useful.(Yang, 2011)

Metaheuristic algorithm

A metaheuristic can be considered as a "master strategy that guides and modifies other heuristics to produce solutions beyond those that are normally generated in a quest for local optimality" (Glover et al., 2003)
Two major components of any metaheuristic algorithms are: intensification and diversification, or exploitation and exploration.(Blum and Roli, 2003)

Diversification means to generate diverse solutions so as to explore the search space on a global scale, while intensification means to focus the search in a local region knowing that a current good solution is found in this region. A good balance between intensification and diversification should be found during the selection of the best solutions to improve the rate of algorithm convergence. The selection of the best ensures that solutions will converge to the optimum, while diversification via randomization allows the search to escape from local optimum and, at the same time, increases the diversity of solutions. A good combination of these two major components will usually ensure that global optimality is achievable. (Yang, 2011)

Chapter 3

Theoretical description

3.1 Subprocess

In computing, a process is an instance of a computer program that is being executed. It contains the program code and its current activity.(Remzi H. Arpaci-Dusseau, 2014) In this project, the program in python for maintenance optimization is the main process. However, the stochastic simulation is not done in python. Communication between processes is needed for such situation. Subprocess is to spawn a process execute missions in other programs under the main process.

The communication between subprocess and main process can be synchronous or asynchronous. For example, subprocess.call(args) is synchronous. The main process will wait for the end of execution of subprocess. As a contrast, subprocess.popen(args) is asynchronous. The main process will continue to execute the code while the subprocess is still under conduction.

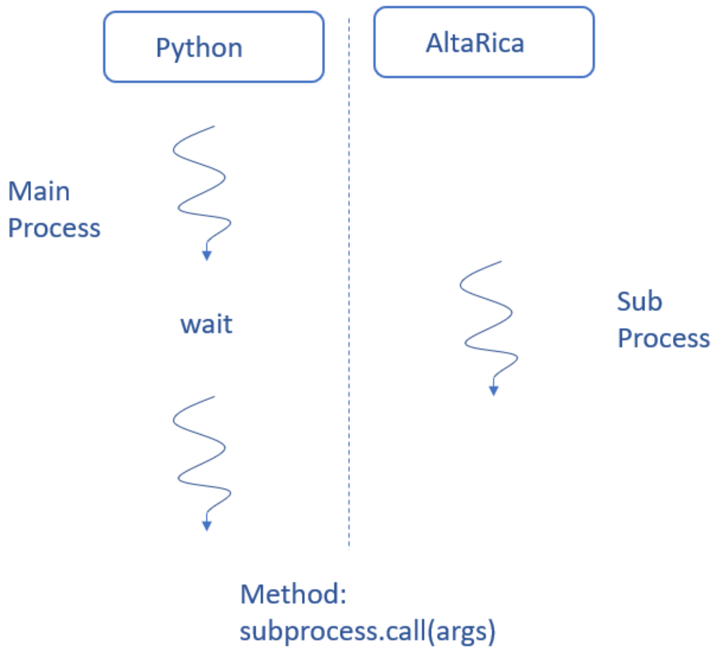


Figure 3.1 Explanation of subprocess

The subprocess module allows user to spawn new processes, connect to their input/output/error pipes, and obtain their return codes. In this project, it is needed for the program to wait for the stimulation result from AltaRica models and do maintenance policy assessment in python. Thus, subprocess module can be used to make the python program wait for the stimulation result and deal with the data.

3.2 XML and xml.dom

Extensible Markup Language (XML) is a simple, very flexible text format derived from SGML (ISO 8879). Originally designed to meet the challenges of large-scale electronic publishing, XML is also playing an increasingly important role in the exchange of a wide variety of data on the Web and elsewhere.(Group, 2013)

XML is self-descriptive and has do nothing with just information wrapped in tags. XML is also extensible. Most XML applications will work as expected even if new data is added. For example, if new observer is added to the indicator description file in this project, the program can still work as expected.

The Document Object Model, or "DOM," is a cross-language API from the World Wide Web Consortium (W3C) for accessing and modifying XML documents. A DOM implementation presents an XML document as a tree structure or allows client code to build such a structure from scratch. It then gives access to the structure through a set of objects which provided well-known interfaces.(Foundation, 2018b)

Node is the basic element for storage of data in XML. The basic information in a node is the name of the node. It can also store more data, such as text, attribution and child nodes.
For example, the XML file and its parsed format in DOM are shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<bookstore>
  <book category="cooking">
    <title lang="en">Everyday Italian</title>
    <author>Giada De Laurentiis</author>
    <year>2005</year>
    <price>30.00</price>
  </book>
</bookstore>
```

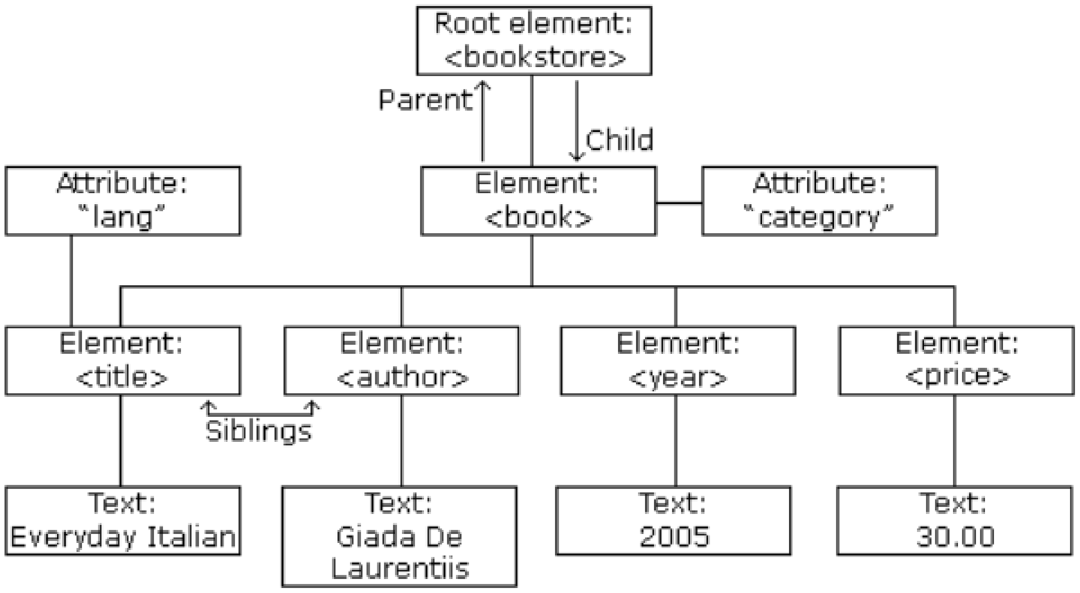


Figure 3.2 DOM tree example(W3Schools)

<bookstore> is the root node, <book category="cooking"> is the child node of <bookstore> with the attribution category equals to “cooking”.

In this project, the data is written in XML file so that it can be stored, modified and used through python through certain module. The module xml.dom is used to process data in the xml file. The data for stochastic simulation in AltaRica is stored in xml format. Thus, the parameters can be modified directly using this module in python.

3.3 AltaRica model

AltaRica is a modelling language focus on safety assessment. The system model written in AltaRica describes the structure and reliability data of the system. As followed is a simple example of a AltaRica model.

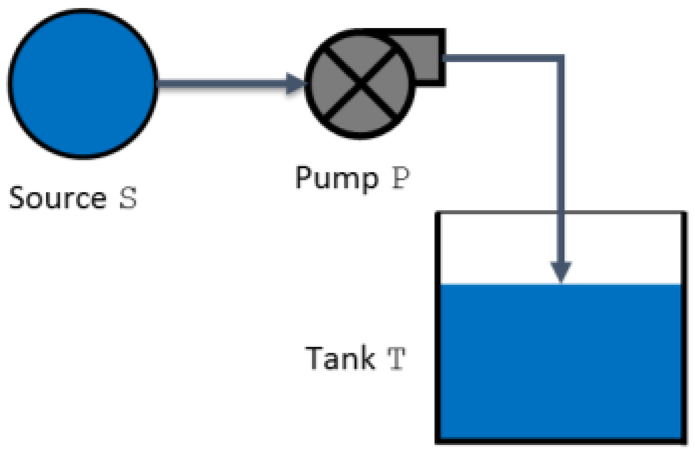


Figure 3.3 Water supply system(irt-systemx, 2017)

The AltaRica model to describe this system consists of the description of the elements, the connection of the elements and the observers to monitor certain conditions. In this simple system, the model should include the reliability and status data of the three elements, the connections of them and observer of input to the tank.

The class RepairableComponent describes a simple kind of element which is repairable with certain data. It has two status, working and not working. The information of transition between these two status and the probability of transitions are also included in the model.
This class stores the mutual information of the three elements.

```
class RepairableComponent
    Boolean working (init = true);
    parameter Real lambda = 0.0001;
    parameter Real mu = 0.01;
    event failure (delay = exponential(lambda));
    event repair (delay = exponential(mu));
    transition
    failure: working -> working := false;
    repair: not working -> working := true;
end
```

The source and pump extends the class RepairableComponent, which means the class source and pump have all the characteristics of repairable component with some specific extra structure or information.

```
class Source
    extends RepairableComponent;
    Boolean output (reset = false);
    assertion
    output := if working then true else false;
end
```

```
class Pump
    extends RepairableComponent;
    Boolean input, output (reset = false);
    assertion
    output := if working then input else false;
end
```

With the three element classes, the block is to describe the system and how the elements are connected.

```
block SimpleWaterSupplySystem
    Source S;
    Pump P;
    block Tank
        Boolean input (reset = false);
    end
    assertion
        P.input := S.output;
        Tank.input := P.output;
End
```

The component objects are created and added to the system. In the assertion, the components are connected. The input of the pipe is connected to the output of source and the input of tank connected to output of pipe.

The AltaRica model describes the system at length. This model describes the structure of the system, the reliability data of the components and how the components are connected.

3.4 AltaRica Tools

The AltaRica tools distributed among the open AltaRica projects can do analysis based on the system model, including stepwise simulation, fault tree assessment, stochastic simulation, etc.
The stochastic simulation process is shown below.

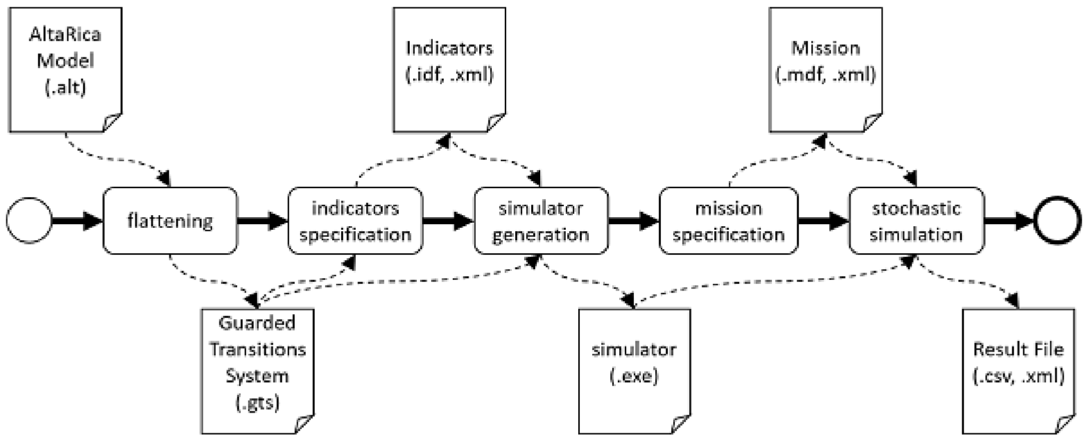


Figure 3.4 Stochastic simulation process (Association, 2016)

The process of stochastic simulation is:

1. The AltaRica model is compiled into a guarded transitions system. This step is actually common to most of the assessment tools and can be performed once for all.

2. Indicators to be calculated are specified. The result of this phase is an XML file, typically called myproject.idf. In the indicator description file stores the data of observers of the states.

3. The stochastic simulator is generated and compiled. For the sake of efficiency, the AltaRica stochastic simulator is actually a generator of stochastic simulators. It generates a dedicated executable file (an application) from the guarded transitions system and the indicator specification files.

4. The mission, or simulation profile, is specified. The result of this step is a XML file, typically called myproject.mdf, that describes the mission time, the observation dates, the number of runs, the seed of the random number generator, the format and the name of the file in which results are printed out.

5. Finally, the stochastic simulation itself is launched, i.e. the executable file generated at step 3 is called with the mission description file generated at step 4 as a parameter. The result is stored in a generated csv file.

The important files created or generated in the process are listed as followed.

1. AltaRica model (.alt)
   The model is written in AltaRica and is the basis of the process. It should be created in the beginning.

2. Guarded transition system (.gts)
   The data of parameters is stored in this file. It will be generated after flattening. The data format is xml.

3. Indicator description file (.idf)
   The indicators are sessions of certain observer in the AltaRica model. It can be, for example, mean sojourn-time of dangerous state. The data format is xml. The idf file should be prepared before the generating of simulator.

4. Simulator (.exe)
   The simulator is an executable generated through the process. Execution of simulator with mission description file as argument will generate a csv file storing the result of simulation.

5. Mission description file (.mdf)
   Mission description file contains the data of the simulation mission, such as mission time, seed, etc. It is an input file for generating the result with simulator. Thus, it should also be prepared before calling simulator. The data format is also xml as indicator description file.

6. Result file (.csv)
   The result file is generated from the simulator with the data of indicators and general mission information.

The AltaRica toolkit consists of several executables that are necessary for the stochastic simulation.

1. ar3c.exe

The executable ar3c.exe is used for flattening. The flattening process is actually performed in two steps: first, the AltaRica model is flattened into a guarded transitions system; second, a type checking tool is applied to this guarded transitions system.

The advantage of this approach is that it makes it possible to detect type errors that would be quite difficult to catch on the AltaRica model. Its drawback is that error messages refer to the guarded transitions system and not to the source model. It is however relatively easy to retrieve the location of errors in the source model from the messages.

Aside type errors, the type checker emits warnings. These warnings do not prevent the guarded transitions system to be simulated or assessed, but should be considered with care, as they reveal in general a modeling mistake. (Association, 2016)

In command line, flattening of the alt file need some arguments:

- The name of the generated guarded transition system file, typically myproject.gts.
- The name of the alt file, typically myproject.alt.

An example of the command line calling ar3c.exe for flattening is "ar3c.exe --gts-xml *myproject.gts* *myproject.alt*". The command –gts-xml is to generate the guarded transition system file in XML format. The names of the alt file and generated gts file are provided as arguments.

2. gtsstocmp.cmd

The batch file gtsstocmp.cmd is used for generating the simulator. It should be called with four arguments: the directory address of the toolkit, the name of guarded transistion system file, the indicator description file, the name of the generated simulator executable. An example of the command line calling gtsstocmp.cmd for generating simulator is "gtsstocmp.cmd Tools\AR3Tools *myproject.gts* *myproject.idf* myproject.exe".

3.5 Character-separated values file

A character-separated values file (CSV file) is delimited text file using comma to separate values. Tabular data can be stored in plain text. Thus, CSV file and Excel file can be transformed between each other easily.

The result file generated from simulator is in CSV format. To derive of the data of indicators, the python module csv is a good choice to conduct the mission. The csv module implements classes to read and write tabular data in CSV format. It allows programmers to say, “write this data in the format preferred by Excel,” or “read data from this file which was generated by Excel,” without knowing the precise details of the CSV format used by Excel. Programmers can also describe the CSV formats understood by other applications or define their own special-purpose CSV formats.(Foundation, 2018a)

3.6 Regular expression

As in the project the parameter values in the AltaRica model need to be set and changed for different simulation task, it is needed for python to locate the position of the parameter value in the model text and make the modification.
A regular expression is a sequence of characters that define a search pattern. It can be used to search and replace some content in a string or text file. For example, the line in the AltaRica model describing the parameter is “parameter Real delayBetweenTests = 2178” originally. The regular expression pattern used to search this line is "(\s+)parameter(\s+)Real(\s+)delayBetweenTests(\s+)=(\s+)(\d+) ". (\s+) can match with one or multiple space or tab. (\d+) can match single digit or multi-digit.

3.7 Algorithms

Metaheuristic algorithm is the main type of algorithm used in this project.
Metaheuristics is a major subfield, indeed the primary subfield, of stochastic optimization. Stochastic optimization is the general class of algorithms and techniques which employ some degree of randomness to find optimal solutions to hard problems. Metaheuristics are the most general of these kinds of algorithms, and are applied to a very wide range of problems.(Luke, 2013)

3.7.1 Gradient descent algorithm

The gradient descent algorithm is developed from a traditional mathematical method for finding the maximum of a function, Gradient Ascent.

Algorithm Gradient Ascent: (Luke, 2013)

```
1: $\vec{x} \leftarrow random\ initial\ vector$
2: repeat
3:   $\vec{x} \leftarrow \vec{x} + \alpha \nabla f(\vec{x})$
4: until $\vec{x}$ is the ideal solution or we have run out of time
5: return $\vec{x}$
```

Gradient descent algorithm basically is just to change the step 3 to make it descent instead of ascent. If the optimization is in many dimensions with many variables, the gradient function is hard or almost impossible to get. As the function is unknown, the gradient is unobtainable. The change of variables won't be based on gradient. Thus, we introduce a term, tweak.
The candidate solution can be in many forms. One simple and common representation for candidate solutions is a fixed-length vector of real-valued numbers. To tweak a vector, we might add a small amount of random noise to each number. Here is a simple way of adding bounded, uniformly distributed random noise to a vector.

Algorithm Bounded Uniform Convolution (Luke, 2013)

```
1: $\vec{v} \leftarrow vector < v1, v2, \dots vl > to\ be\ convolved$
2: $\vec{p} \leftarrow probability\ of\ adding\ noise\ to\ an\ element\ in\ the\ vector$
3: $r \leftarrow half\ range\ of\ uniform\ noise$
4: $min \leftarrow minimum\ desired\ vector\ element\ value$
5: $max \leftarrow maximum\ desired\ vector\ element\ value$
6: for i from 1 to l do
7:   if p ≥ random number chosen uniformly from 0.0 to 1.0 then
8:       repeat
9:           $n \leftarrow random\ number\ chosen\ uniformly\ from - r\ to\ r\ inclusive$
10:          until min ≤ vi + n ≤ max
```

```
11:         $vi \leftarrow vi + n$
12: return $\vec{v}$
```

Applying the tweak algorithm to the gradient descent algorithm, the algorithm can be applied to more general cases.

Algorithm Steepest Ascent Hill-Climbing With Replacement (Luke, 2013)

```
1: $n \leftarrow number\ of\ tweaks\ desired\ to\ sample\ the\ gradient$
2: $S \leftarrow some\ initial\ candidate\ solution$
3: $Best \leftarrow S$
4: repeat
5:     $R \leftarrow Tweak(Copy(S))$
6:     for n-1 times do
7:         $W \leftarrow Tweak(Copy(S))$
8:         if Quality(W) > Quality(R) then
9:             $R \leftarrow W$
10:    $S \leftarrow R$
11:        if Quality(S) > Quality(Best) then
12:            $Best \leftarrow S$
13: until Best is the ideal solution or we have run out of time
14: return Best
```

The application of this algorithm in this project is showed as below in the graph. The optimum test interval can be found after a search applying gradient descent algorithm. However, this algorithm has its drawback that the search may be stocked in the local optimum.

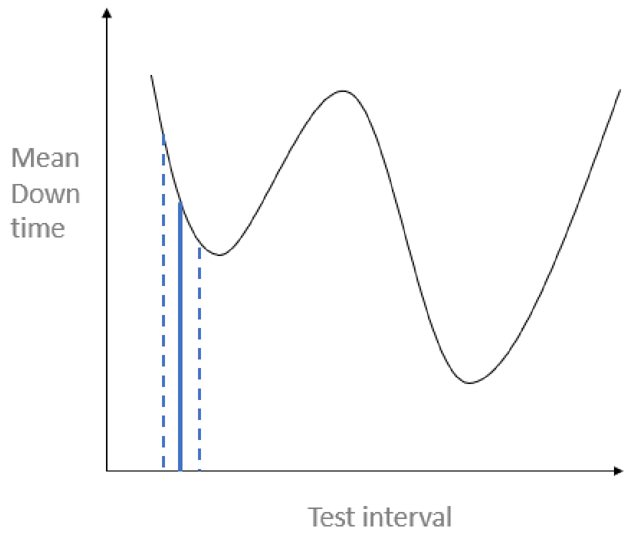


Figure 3.2 Gradient descent algorithm

3.7.2 Tabu search

Tabu search employs a different approach to doing exploration: it keeps around a history of recently considered candidate solutions (known as the tabu list) and refuse to return to those candidate solutions until they're sufficiently far in the past. Thus if we wander up a hill, we have no choice but to wander back down the other side because we're not permitted to stay at or return to the top of the hill. (Luke, 2013)
The way to apply tabu search is to maintain a cache that stores the recent candidate solutions. The structure of cache in is this case is showed as below.

Tau(1)	C1
Tau(2)	C2
Tau(3)	C3
…	….
Tau(n)	Cn
…	….

Table 3.1 Tabu cache

If the number of candidates is large, the efficiency of finding optimum will increase significantly with tabu search. The tabu search algorithm is showed as followed.

Algorithm Tabu Search (Luke, 2013)

```
1: $l \leftarrow Desired\ maximum\ tabu\ list\ length$
2: $n \leftarrow number\ of\ tweaks\ desired\ to\ sample\ the\ gradient$
3: $S \leftarrow some\ initial\ candidate\ solution$
4: $Best \leftarrow S$
5: $L \leftarrow \{\}\ a\ tabu\ list\ of\ maximum\ length\ l$
6: Enqueue S into L
7: repeat
8:      if Length(L) > l then
9:          Remove oldest element from L
10:         $R \leftarrow Tweak\big(Copy(S)\big)$
11:         for n-1 times do
12:             $W \leftarrow Tweak(Copy(S))$
13:             if W∉L and (Quality(W) > Quality(R) or R∈L) then
14:                 R←W
15:         if R∉L and Quality(R) > Quality(S) then
16:             S←R
17:             enqueue R into L
18:     if Quality(S) > Quality(Best) then
19:             $Best \leftarrow S$
20: until Best is the ideal solution or we have run out of timw
21: return Best
```

However, tabu search only works in discrete spaces. In this project, the range of test intervals consists of limited number of choices. Thus, the tabu search algorithm can be applied. If the real function is like the graph below, the flat area in middle can produce a lot of repeat search. However, the application of tabu search algorithm can help to reduce the repeated work. The search can recognize the flat area and break from this simulation or start with another start point.

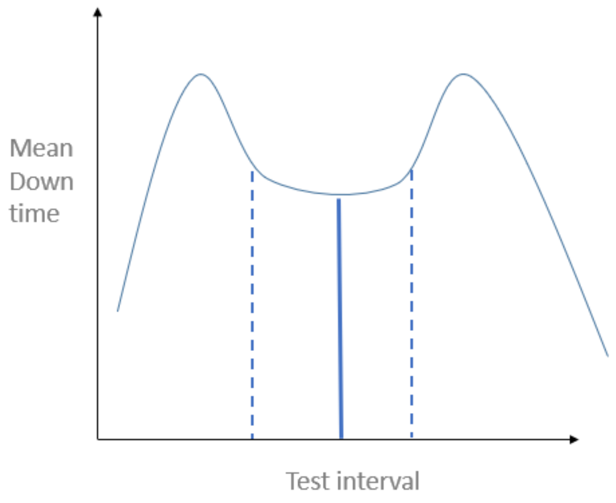


Figure 3.3 Tabu search algorithm

3.7.3 Simulated annealing algorithm

Simulated Annealing was developed by various researchers in the mid 1980s, but it has a famous lineage, being derived from the Metropolis Algorithm. The algorithm varies from Hill-Climbing in its decision of when to replace S, the original candidate solution, with R, its newly tweaked child. Specifically, if R is better than S, we'll always replace S with R as usual. But if R is worse than S, we may still replace S with R with a certain probability.(Luke, 2013)

Simulated annealing algorithm can help the search to escape from some local optimum. As shown below, when the search reaches the left point, the search will stop without simulated annealing algorithm. However, if simulated algorithm is applied, the search have a certain probability to keep search forward and manage to escape from the local optimum.

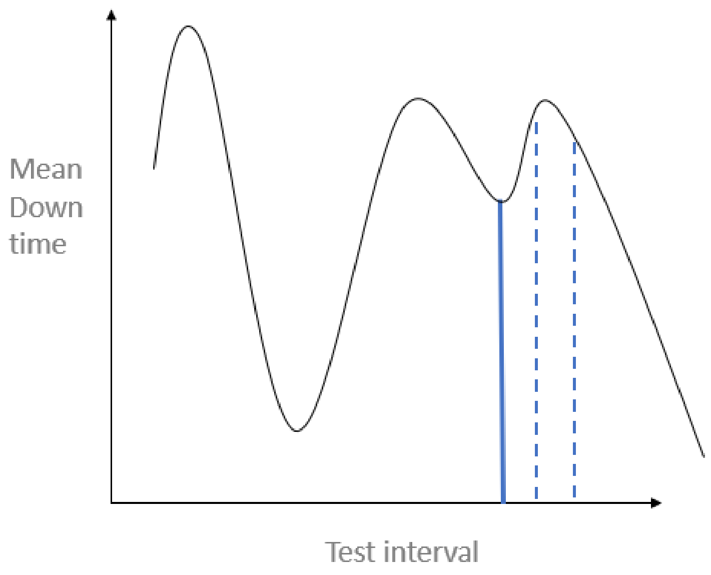


Figure 3.4 Simulated annealing algorithm

Algorithm Simulated Annealing (Luke, 2013)

```
1:  $t \leftarrow temperature, initially\ a\ high\ number$
2:  $S \leftarrow some\ initial\ candidate\ solution$
3:  $Best \leftarrow S$
4: repeat
5:          $R \leftarrow Tweak(Copy(S))$
6:          if Quality(R) > Quality(S) or if a random number chosen from 0 to 1<P then
7:                  $S \leftarrow R$
8:          Decrease t
9:          if Quality(S) > Quality(Best) then
10:                 $Best \leftarrow S$
11: until Best is the ideal solution or we have run out of time, or t ≤ 0
12: return Best
```

3.7.4 Population method

Population-based methods differs from the previous methods in that they keep around a sample of candidate solutions rather than a single candidate solution. Each of the solutions is involved in tweaking and quality assessment, but what prevents this from being just a parallel hill-climber is that candidate solutions affect how other candidates will hill-climb in the quality function. This could happen either by good solutions causing poor solutions to be rejected and new ones created, or by causing them to be Tweaked in the direction of the better solutions.(Luke, 2013)

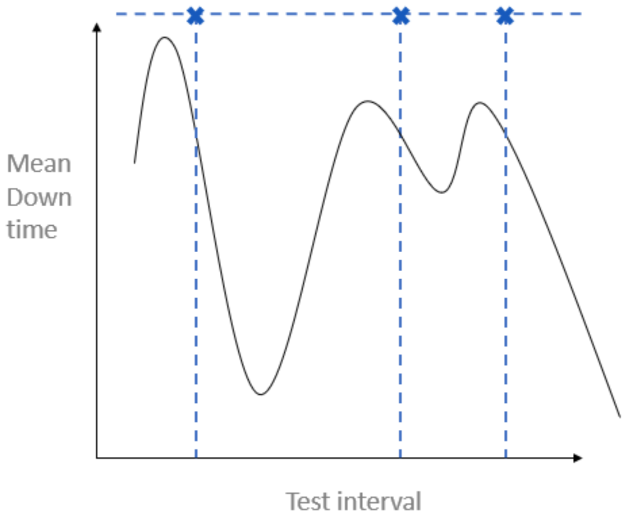


Figure 3.5 Population method

For one simulation, the result can be a local optimum. As the target is to find the local optimum, one approach to it is to do lots of simulations with random start in the data range. Among all the local optimums found, the best one has a high possibility to be the global optimum. The more simulations done, the higher possibility to get the ideal solution.

Chapter 4

Experiments

4.1 Introduction

The target of this project is to maintenance optimization of some certain system. The process is as followed.

1. Build the system model in AltaRica
2. Compile the system model file and get the gts file
3. Create the indicators description file
4. Generate the simulator
5. Create the mission description file
6. Generate the csv file
7. Read and assess the result
8. Choose other candidates and repeat 2 to 7

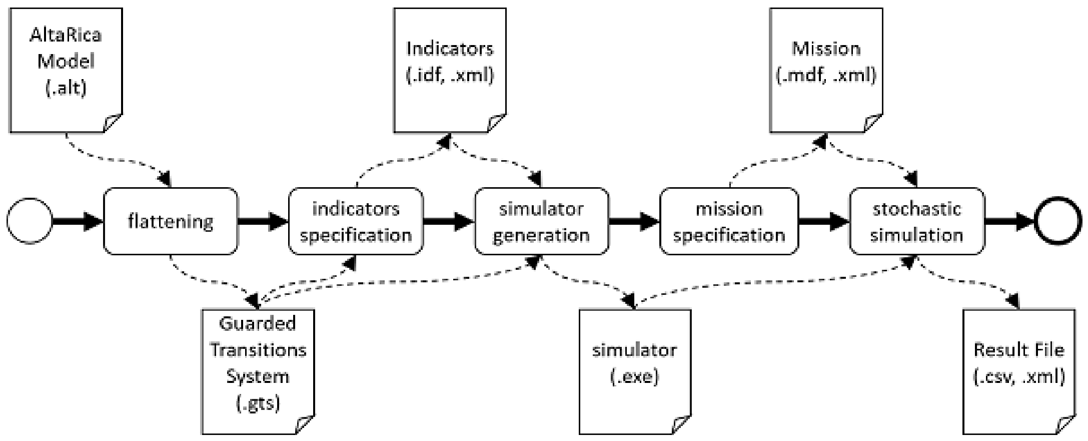


Figure 4.1 Stochastic simulation process (Association, 2016)

For every candidate maintenance policy, the stochastic simulation should be conducted independently. Therefore, the indicators description file and the mission description file should be modified accordingly. The results of the simulation are stored in library in python. Metaheuristic algorithms are used to lift the efficiency of step 8, which is the process of choosing candidate solutions to simulate and assessing the result.

The outcome of simulation is normally not predictable. That’s why the simulation process is to some extends a black box. Some candidate solutions are provided to the system. After stochastic simulation, the result is given, but not predictable. It is hard or even impossible to analyze the relation between result and input.

In this case, optimization needs some algorithm for assessing the result of simulation. The relationship between optimization and simulation is shown in figure 4.2. The stochastic simulation is a black box, that gives unpredictable outcome with input. The optimization process needs to explore as many candidate solutions as possible. In the meantime, the efficiency of exploration needs to be considered as well.

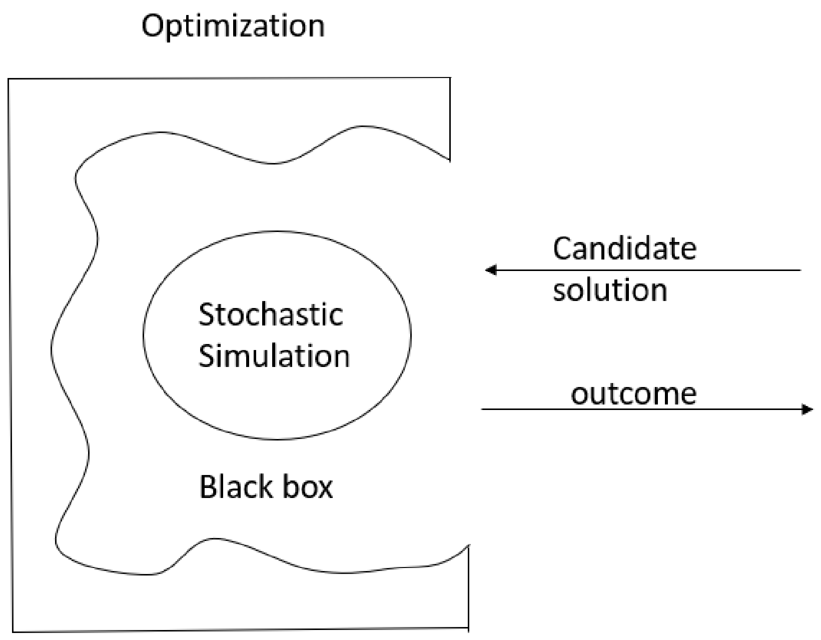


Figure 4.2 Black box

4.2 Single element system

4.2.1 Introduction

The first system to be analyzed is a system consisting of an element. The system is periodically tested. The system has an exponentially distributed probability of failure. The failure will only be revealed at a test and repaired afterwards. Thus, the system has following statuses, which are working, test when working, undetected failure, detected failure, repair. The transitions among these statuses are shown in the figure 4.3.

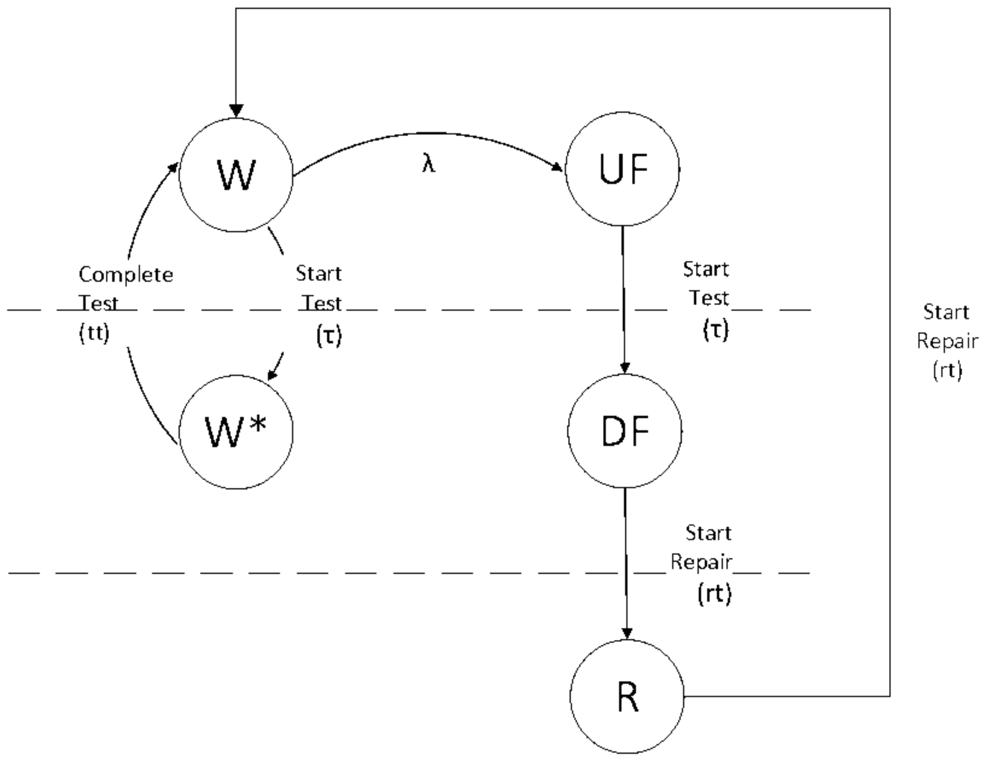


Figure 4.3 Status transitions in single element system

In figure 4.3, the status abbreviations are used.

1. W: Working.
2. W*: Tested when working
3. UF: Undetected failure, which is the period between the system failure and next test.
4. DF: Detected failure
5. R: Repair

In these transitions, repair time and time to failure are exponentially distributed. Test time and test interval are fixed. The test interval is the key parameter to be optimized. The preset values of the parameters are listed in table 4.1.

Failure rate	1.0e-04
Test duration	12
Repair rate	1.0e-02

Table 4.1 Parameter data

For such a system, optimization is needed for the balance between preventive maintenance and corrective maintenance. Low test interval will lead to loss of production in frequent tests. Whereas, large test interval may lead to larger amount of failures in between tests. To optimize the maintenance policy, test interval is the key parameter. The rest parameters are mostly determined. Repair rate is determined by the efficiency of the maintenance team. There is not much optimization that can be done through stochastic simulation and analysis. Test duration is determined by factors outside the system as well. Test interval is the only parameter can be optimized for maintenance policy here.

4.2.2 Store parameters

As determined previously, the only varying parameter for optimization is test interval. The candidate values are stored in xml format. (Appendix A.1)

To read and store the parameter values in python, module xml.dom is used. The key point is to search the node parameter with attribute name equal to the desired parameter. Here the parameter name is delayBetweenTests, which is test interval. The child nodes of the node parameter are named candidate with various attribute value. Using the module to derive of the attribute value and storing in a dictionary in python is the final step.

The dictionary in python storing the parameter data will be look like:
ParaDict = {“delayBetweenTests” : [718, 1448, 2178, 2098, 3638, 5098, 5828, 6558, 7288, 8018, 8748]}

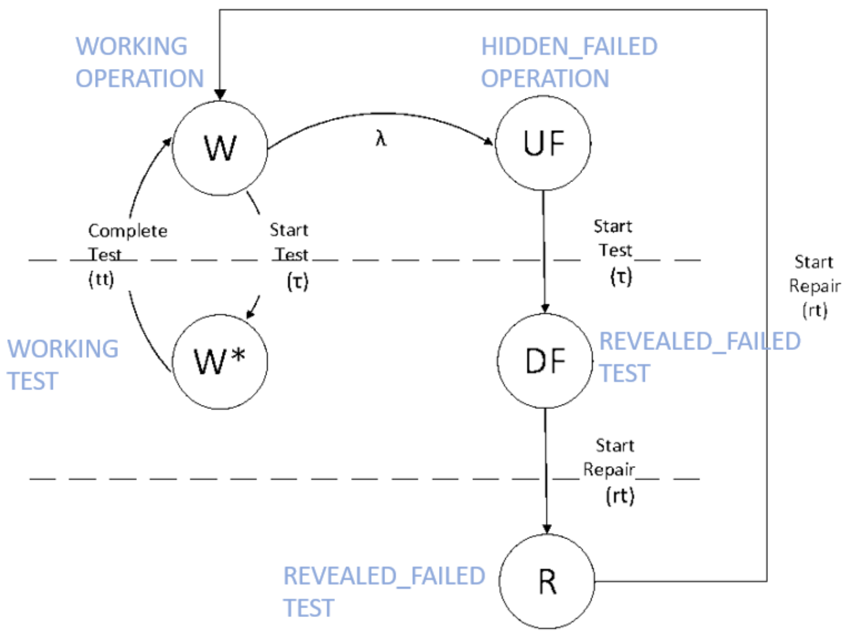
This function can be used to store different parameter candidate value lists. In this case, there is just one parameter to be optimized, which is the test interval for the single element. However, the test interval of different sections in a more complex system may vary from each other. There can also be other parameters to be optimized other than test interval. This function can fulfill these potential needs in the industry.

4.2.3 Build system model

The first step is to build the system model in AltaRica. The transitions within the system statuses are shown in figure 4.3. The AltaRica model is in Appendix.

The system has two domins, State and Phase. State domin has three possible values, WORKING, HIDDEN_FAILED, REVEALED_FAILED. Phase domin has two possible values, OPERATION, TEST. The combination of these two domin can represent all possible status of the system.

The transitions are corresponding to the events. Event failure is transition from WORKING to HIDDEN_FAILED, which is from W to UF in figure 4.3. Event start test is transition from OPERATION to Test. In figure, it is from W to W* or from UF to DF, which depends on the initial status. Event complete test is transition from TEST to OPERATION, which in figure is from W* to W or from DF to R. When the system is



failed, the event complete test is equal to 'start repair' in figure. Event repair is transition from REVEALED_FAILED to WORKING, which is from R to W.

Figure 4.4 Relation between domins and states

There are two types of occurrences of the events in this system. One is that the event happens after some certain time. Event start test and complete test are within this category. This kind of preventive maintenance is clock-based. The test and preventive maintenance is conducted at a certain frequency regardless of the condition monitoring. The other type of occurrences of the events is exponential distribution. The event failure occurs at a exponentially distributed probability.

The function of the two observers is to monitor the status of the system. Observer unavailable can observe the unavailability of the system, when domin state is not WORKING or domin phase is not OPERATION. The observers are the foundation for the indicators later in the process. The observed states are shown in figure 4.5.

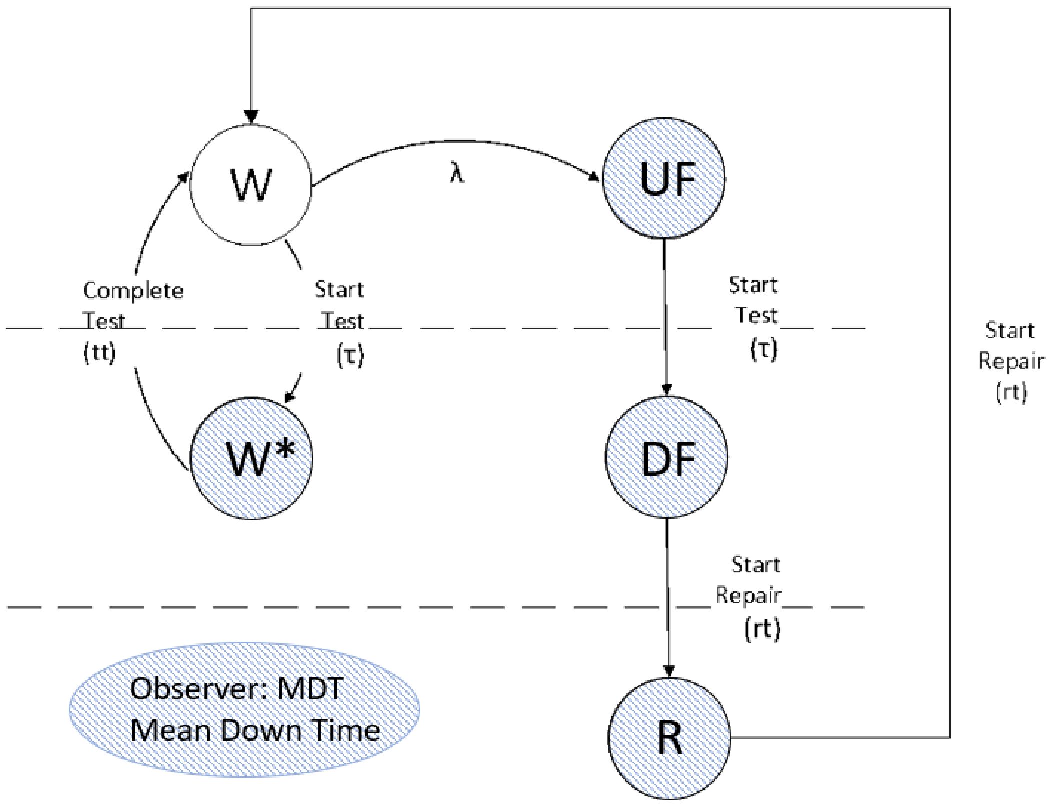


Figure 4.5 Observed states

The system model is built with detailed reliability data and structure. The filename extension is .alt.

4.2.4 Modify alt file

The alt file contains the original data of the parameters. For the stochastic simulations for different candidate values, the parameter needs to be set to the candidate value. This preparation work should be done for each simulation before compile.

Regular expression is used to search and change the text where stores the parameter data. In the AltaRica model of this system, the line determines the test interval is:

parameter Real delayBetweenTests = 3638;

The words ‘parameter’, ‘Real’, ‘delayBetweenTests’ and the equal sign ‘=’ are fixed. The number of tab and spaces in between the words is not necessarily fixed. The candidate value is not fixed. After figuring the features of the text, the regular expression can be determined. (/s+) can match one or more space or tab. (/d+) can match one or more digits. As the name of the parameter may change for simulation of different system, string format() method is used here.

In the end the regular expression pattern for the parameter line is:

(\s+)parameter(\s+)Real(\s+){name}(\s+)=(\s+)(\d+)".format(name=VariableName)

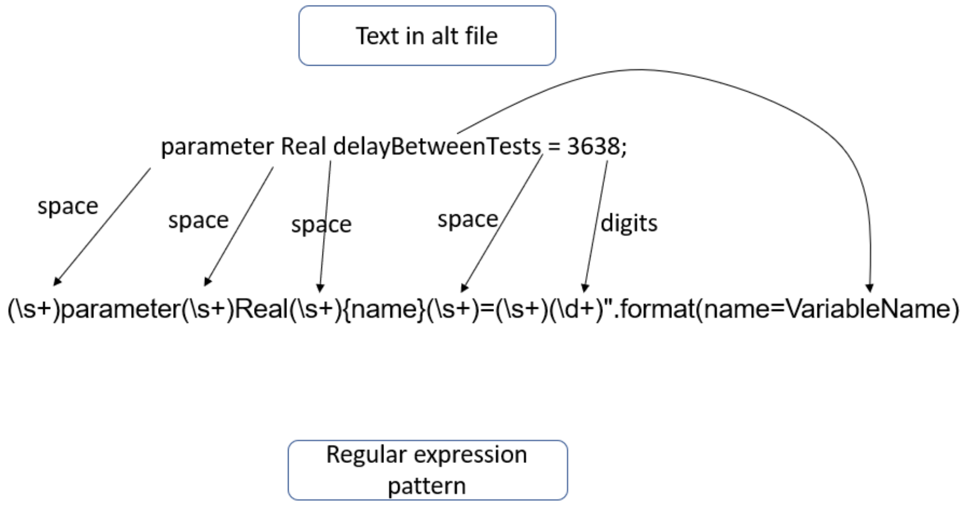

Text in alt file
parameter Real delayBetweenTests = 3638;
space
space
space
space
digits
(\s+)parameter(\s+)Real(\s+){name}(\s+)=(\s+)(\d+)".format(name=VariableName)
Regular expression pattern

Figure 4.6 Regular expression pattern

Figure 4.6 shows the relationship between expression in alt file and the regular expression. The variable name is changeable. The format function in python can make use of the input variable name.

4.2.5 Compile alt file

The AltaRica tool kit is used for compiling the system model. In the AltaRica tools, ar3c.exe can be used for compiling. As introduced before, this executable file needs to called with several arguments in command line.

In the in-line help of executable ar3c.exe, the usage of it is provided. The command format is ar3c.exe [option]* <AltaRica-filename.alt>*. After checking the list of command options, the following command suits the case.

```
--gts-xml <filename>:
 Print the GTS model at XML format into <filename>.
```

The finalized command line is "ar3c.exe --gts-xml *periodicallyTestedComponent.gts* *periodicallyTestedComponent.alt*". As the executable is an outside program, subprocess is needed. The purpose of subprocess is to execute the command line while keeping the main process, stochastic simulation in python, waiting. To test the subprocess module, a simple experiment is performed.

The main purpose of subprocess module is to make the python program wait for the result of simulation in AltaRica. Here a simple program written in C is used for test:

```
#include <stdio.h>
#include<windows.h>
int main()
{
 int ran_num;
 sleep(ran_num=rand() % 3000);
 printf("Hello, World!");
 return 0;
}
```

As the time for simulation varies, the time to notify the main process is unsure. I make the hello program wait for a random time.

```
import subprocess
## main
if __name__=="__main__":
        for i in range(100):
                str = "0";
                str = subprocess.check_output("hello.exe");
                print(str);
```

In the python program, I set a value to str and assign a new value to str in the subprocess and then print str. If main process continues to run without waiting for the result from subprocess, there will be some “0” in the output. After compiling and running, the results only consist of “Hello, World!”. The function of subprocess module is proven to be eligible for the stochastic simulation program.

4.2.6 Create indicator description file

The indicators are some certain observed status and value of observers created in the AltaRica model. In this experiment, there are two observers for the system as introduced before. The target of optimization is to minimize the mean down time of the system. The related observer is the unavailable observer. Mean down time is the mean sojourn-time of unavailable state. The indicator and detailed settings are determined.

The indicator description file is in xml format. To create a xml file, the module xml.dom is used. The root node is ‘ar3ccp’, relating this xml file to its use case. The first layer of child nodes is ‘calculation’, which states the related observer. In this project, the calculation node is created with attribute observer equal to ‘unavailable’. All indicators using this observer for this simulation are child nodes of the calculation node. The basic information is set as attributes of the indicator node. In this project, the indicator node is created with several attributes, which are ‘type’ equal to ‘sojourn-time’, ‘name’ equal to ‘MeanDownTime’, ‘value’ equal to ‘true’. As the required value is mean sojourn-time, a child text node, ‘mean’, is appended to the indicator node.

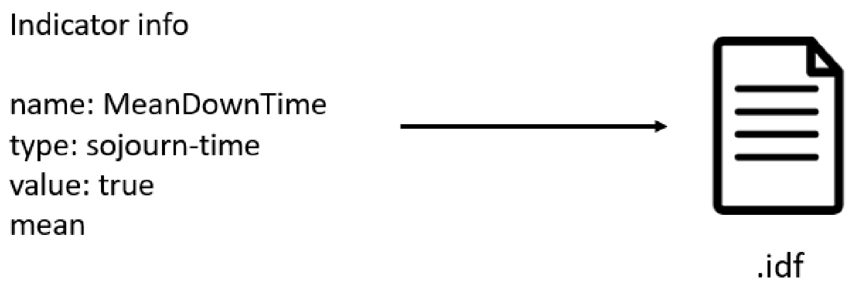


Figure 4.7 indicator description file

The nodes are created and connected according to the structure through module xml.dom. Then, the DOM tree is written in to a txt file with extension name .idf. The xml file can be used as argument to the next steps.

4.2.7 Generate simulator

To generate the simulator, the tool gtsstocmp.cmd is used. The arguments for calling this command can be figured out in the batch file. (Appendix A.4)

The first argument is the directory of the compiler folder. The tools for compiling and other necessary files must be stored in the same directory. Some of the generated files will be put in the subfolders of the tool kit folder. The second argument is the name of the gts file and the third argument is the name of the idf file. These two files must be prepared before generating the simulator. The last argument is the name of the executable file to be generated.

The command line in this project is:

*"gtsstocmp.cmd Tools\AR3Tools *periodicallyTestedComponent.gts* *periodicallyTestedComponent.idf* periodicallyTestedComponent.exe".*

The executable will be generated in the same folder.

4.2.8 Generate mission description file

The simulator is used to do the simulation with mission description file. The data format of the mission description file is also xml. Thus, the xml.dom is used as well.

The root node of this file is the same as indicator description file, ‘ar3ccp’. Then, the child node ‘simulation’ is the node containing the mission information of the simulation. The three attributes of the simulation node are ‘result-csv’ with value of the name of the result file, ‘number-of-runs’ with the value of the simulation runs, ‘seed’ with the seed value for simulation. Thus, the node simulation will be like <simulation number-of-runs="10000" results-csv="results.csv" seed="12345">.

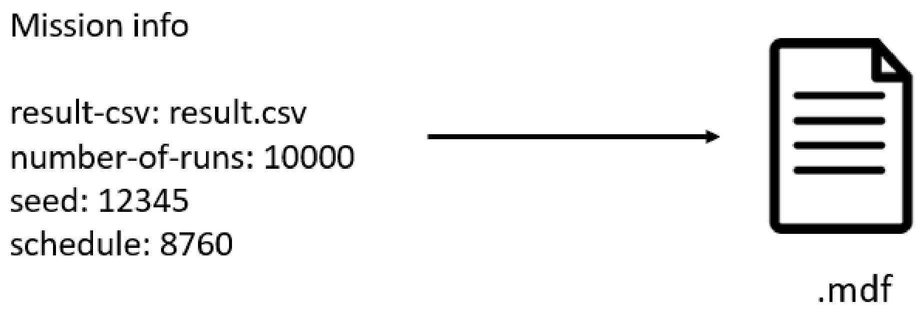


Figure 4.8 mission description file

More setting can be set for the simulation in the mission description file. One other node added in this experiment is a child node ’schedule’ with attribute mission-time. In this experiment, the mission time is set to one year, which is 8760 hours.

4.2.9 Conduct simulation

The simulation is executed using the command line as well. The simulator is generated in last step. Calling the simulator requires one argument, which is the name of the mission description file. The simulation will be conducted accordingly and the result file will be generated. The command line for this experiment is:

“_periodicallyTestedComponent.exe *periodicallyTestedComponent.mdf*”._

4.2.10 Read the result

The result file format of the simulation is comma-separated values. An example of the result of the simulation is shown below.

```
    meta-data
    ;number-of-runs;10000
    ;seed;12345
    ;mission-time;8760.0
    ;model-name;Unit
    ;filename;periodicallyTestedComponent.gts
    ;start-time;Fri May 11 01:34:01 2018
    ;end-time;Fri May 11 01:34:01 2018
    ;steps min;5
    ;steps mean;6.3728
    ;steps max;10
    ;tool version;1.0.0
    ;compiler version;1.0.0
    observer;unavailable;type;Boolean
    ;indicator;MDT;type;sojourn-
 time;value;true
    ;;date;8760.0
    ;;;sample-size;10000
    ;;;mean;1371.95
```

The module csv is used to fetch the useful information from the csv file. Every line is a list stored in the reader dictionary. The delimiter is semicolon ";". The needed line is the last line, ";;;mean;1371.95". It is stored in list as [, , ,mean,1371.95]. Module csv can help to find the value and the data format of number '1371.95' is string. After changing the format of the number to float, the data can be stored and made use of.

4.2.11 Maintenance optimization

The combination of previous functions and steps can conduct any single simulation for the system. The approach to maintenance optimization is a combination of the stochastic simulation and the application of algorithms. Stochastic simulation generates the reliability performance of the system. Algorithms help to find the optimum solution efficiently. For a system, there can be a lot of candidate maintenance policies. The algorithm helps to skip some unnecessary work.

The algorithms used here are metaheuristic. Population-based gradient descent algorithm is the main algorithm used for searching the optimum solution. Tabu search algorithm, simulated annealing algorithm are used for improving the efficiency of search.

The main process of optimization is:

1. Choose a candidate solution as base from the list randomly and do simulation
2. Choose the two nearby solutions of the previous one and do simulation
3. Assess the performance of all the three simulation results
4. Choose the candidate solution with best performance as the base
5. Do the loop of step 1 to 4 until find the local optimum solution.
6. After finding a certain number of local optimum, derive of the global optimum solution among them.

The main problem left is how to assess the result and judge if this candidate solution is a local optimum. As shown in the gradient descent algorithm figure, as the parameter value changes, the mean down time varies.

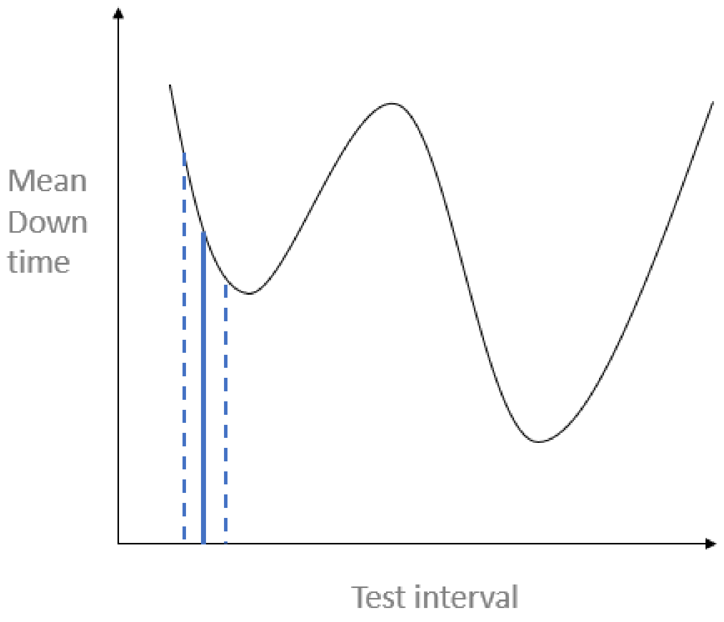


Figure 4.2 Gradient descent algorithm

The local optimum is the solution at the saddle point. The method of assessing if the current solution is local optimum is comparing the performance of the current solution and two nearby solutions. If the current solution is at the saddle point, the nearby two solutions will have higher mean down time than the middle one. However, somewhere in the performance figure may look like figure 4.2. If the search goes a bit further, the program can find a much better solution than the local optimum found according to the gradient descent algorithm.

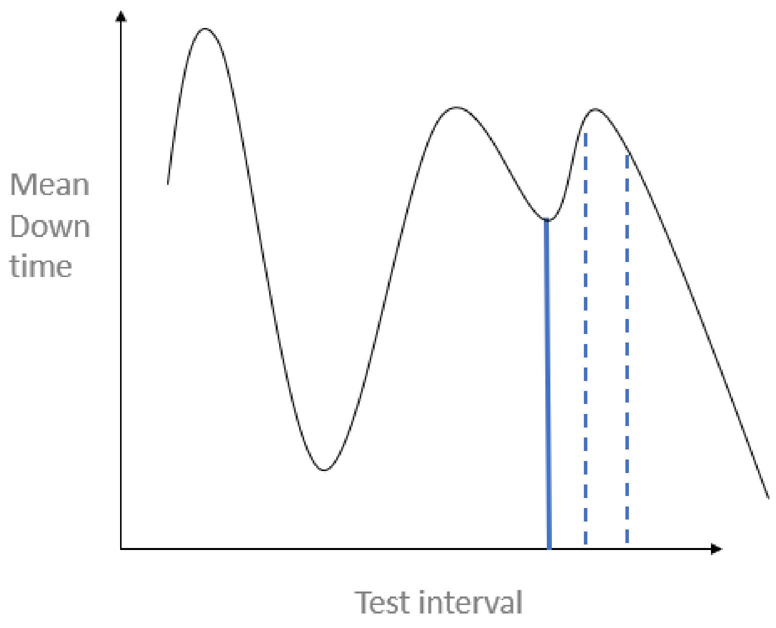


Figure 4.3 Simulated annealing algorithm

Simulated annealing algorithm can help to escape from such a saddle point. This algorithm can help to find better solution in one single simulation. In the program, the search has a probability to go on even a local optimum is found.

Besides Simulated annealing, tabu search is also used to lift the efficiency of the optimization process. In the program, if a cache stores a list of candidate solutions that have been searched recently. If the candidate solution picked is among the list, it will be skipped and a new candidate will be picked. The combination of these algorithms helps to find the best test interval efficiently.

4.2.12 Result

The result of the optimization is derived after 30 runs. The best maintenance policy is conducting test and preventive maintenance every 1448 hours. The estimated mean down time of this maintenance policy in a year is 723 hours.

The whole optimization process is conducted automatically and efficiently. A large list of candidate maintenance policies can be simulated and assessed professionally. It has proven the practicability of this program.

Chapter 5
Conclusion & Future work

The main objective of this project is to do maintenance optimization for a system. The process contains two main parts, stochastic simulation and optimization. Stochastic simulation is done using the AltaRica tools. The simulation is done by AltaRica simulator and the configuration files are generated by AltaRica tools or python script. Maintenance optimization is done through python script. This project makes the combination of two parts and does the maintenance optimization for the system. The process is controlled by python script automatically. The efficiency of the process is guaranteed in the two parts. In the simulation part, the simulator from AltaRica project is written in C++, which has high efficiency. In the optimization part, the optimization process is optimized by metaheuristic algorithm.

This project can be adapted and applied to many other systems. The modification needed to be done for different system is the AltaRica model and some configuration information. This project has high flexibility. It can be transplanted to many systems and the result is reliable.

Bibliography

ASSOCIATION, A. 2016. *AltaRica Wizard -Help* [Online]. Available: AltaRica Wizard [Accessed 2018].

AVEN, T. & JENSEN, U. 1999. *Stochastic models in reliability,* New York, Springer.

BARROS, A. 2017. *RE: Compendium of TPK4450 Datadrevet prognose og prediktivt vedlikehold.*

BELYI, D., POPOVA, E., MORTON, D. P. & DAMIEN, P. 2017. Bayesian failure-rate modeling and preventive maintenance optimization. *European Journal of Operational Research,* 262**,** 1085-1093.

BLUM, C. & ROLI, A. 2003. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys (CSUR),* 35**,** 268-308.

BOHLIN, M. & W RJA, M. 2015. Maintenance optimization with duration-dependent costs. *Annals of Operations Research,* 224**,** 1-23.

DEGEILH, Y. & GROSS, G. 2015. Stochastic simulation of power systems with integrated intermittent renewable resources. *International Journal of Electrical Power and Energy Systems,* 64**,** 542-550.

ENERGY, G. 2010. Western wind and solar integration study. *NREL*.

FOUNDATION, P. S. 2018a. *csv — CSV File Reading and Writing* [Online]. Available: https://docs.python.org/2/library/csv.html [Accessed 2018].

FOUNDATION, P. S. 2018b. *xml.dom - The Document Object Model API* [Online]. Available: https://docs.python.org/2/library/xml.dom.html [Accessed 2018].

GLOVER, F., KOCHENBERGER, G. A. & SPRINGERLINK 2003. Handbook of metaheuristics. Springer US.

GROUP, X. C. W. 2013. *Extensible Markup Language (XML)* [Online]. Available: https://www.w3.org/XML/ [Accessed 2018].

IRT-SYSTEMX. 2017. *OpenAltaRica - Example: A (Simple) Water Supply System* [Online]. Available: https://www.openaltarica.fr/docs/The%20OpenAltaRica%20Platform%20-%20Example%20(Simple)WaterSupplySystem.pdf [Accessed 2018].

LUKE, S. 2013. *Essentials of Metaheuristics,* George Mason University, Lulu.

MORTADA, H., PROSVIRNOVA, T. & RAUZY, A. 2014. Safety assessment of an electrical system with altarica 3.0. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics),* 8822**,** 181-194.

RAUZY, A. 2014. The AltaRica 3.0 Project.

REMZI H. ARPACI-DUSSEAU, A. C. A.-D. 2014. Abstraction: The Process The Process API. *Operating Systems: Three Easy Pieces.*

SHAYESTEH, E., YU, J. & HILBER, P. 2018. Maintenance optimization of power systems with renewable energy sources integrated. *Energy,* 149**,** 577-586.

W3SCHOOLS. *XML DOM* [Online]. Available: https://www.w3schools.com/xml/xml_dom.asp [Accessed 2018].

YANG, X.-S. 2011. Metaheuristic Optimization. *Scholarpedia,* 6**,** 11472.

Appendix A

A.1 XML file of candidate values

Test.txt

```
   <?xml version="1.0"?>
   <optimization>
      <model file="periodicallyTestedComponent" main="Unit">
      <parameters>
         <parameter name="delayBetweenTests">
            <candidate value="718" />
            <candidate value="1448" />
            <candidate value="2178" />
            <candidate value="2098" />
            <candidate value="3638" />
            <candidate value="5098" />
            <candidate value="5828" />
            <candidate value="6558" />
            <candidate value="7288" />
            <candidate value="8018" />
            <candidate value="8748" />
         </parameter>
      </parameters>
      </model>
</optimization>
```

A.2 XML file of Indicator Description File

periodicallyTestedComponent.idf

```
<?xml version="1.0" ?>
<ar3ccp>
   <calculation observer="unavailable">
        <indicator name="MDT" type="sojourn-time" value="true">
            <mean/>
        </indicator>
   </calculation>
</ar3ccp>
```

A.3 XML file of Mission description file

periodicallyTestedComponent.mdf

```
<?xml version="1.0" ?>
<ar3ccp>
    <simulation number-of-runs="10000" results-csv="results.csv" seed="12345">
        <schedule mission-time="8760"/>
    </simulation>
</ar3ccp>
```

A.4 Batch file for generating simulator

gtsstocmp.cmd

```
@echo off
echo change current directory to compiler folder (%1) and setup paths
cd %1
setlocal
SET PATH=%cd%
SET PATH=%PATH%;%cd%/cppcompiler
SET PATH=%PATH%;%cd%/cppcompiler/bin
SET COMPILER_PATH=
SET LIBRARY_PATH=
SET C_INCLUDE_PATH=
SET CPATH=
SET CPLUS_INCLUDE_PATH=
SET OBJC_INCLUDE_PATH=

echo clean up compiler folder
REM rm.exe -f gtssto/obj/*.o
REM rm.exe -f gtssto/bin/gtssto.exe
REM rm.exe -f gtssto/src/GTSSTOModel.cpp
REM rm.exe -f gtssto/include/GTSSTOModel.hpp
del gtssto\obj\*.o
del gtssto\bin\gtssto.exe
del gtssto\src\GTSSTOModel.cpp
del gtssto\include\GTSSTOModel.hpp

echo generate C++ files
gtsstocmp.exe --configuration %3 --output gtssto/tmp %2
REM mv.exe gtssto/tmp/GTSSTOModel.cpp gtssto/src
REM mv.exe gtssto/tmp/GTSSTOModel.hpp gtssto/include
move /Y gtssto\tmp\GTSSTOModel.cpp gtssto\src
move /Y gtssto\tmp\GTSSTOModel.hpp gtssto\include

echo compile C++ files
```

```
g++.exe -w -I gtssto/include -std=c++11 -std=gnu++11 -O2 -s -c -o gtssto/obj/GTSSTOModel.o gtssto/src/GTSSTOModel.cpp
g++.exe -w -I gtssto/include -std=c++11 -std=gnu++11 -O2 -s -c -o gtssto/obj/main.o gtssto/src/main.cpp
g++.exe -w -L gtssto/lib -std=c++11 -std=gnu++11 -o gtssto/bin/gtssto.exe gtssto/obj/GTSSTOModel.o gtssto/obj/main.o -lm -l gtsstosimkernel

echo copy executable and DLLs to the target directory
REM mv.exe gtssto/bin/gtssto.exe %4
move gtssto\bin\gtssto.exe %4
copy /y gtssto\bin\*.dll "%~dp4" > nul
echo compilation completed
```

A.5 AltaRica Model

periodicallyTestedComponent.alt

```
domain State {WORKING, FAILED}
domain Phase {OPERATION, TEST, MAINTENANCE}

block Unit
State _state(init=WORKING);
Phase _phase(init=OPERATION);
event failure(delay=exponential(failureRate));
event startTest(delay=Dirac(delayBetweenTests));
event completeTest(delay=Dirac(testDuration));
event startMaintenance(delay=Dirac(delayBeforeMaintenance));
event completeMaintenance(delay=Dirac(maintenanceDuration));
parameter Real failureRate = 1.0e-4; // the component fails on average about once in a
year
parameter Real delayBetweenTests = 2178; // the component is tested every 3 months
parameter Real testDuration = 12; // the test lasts one shift, i.e. 12 hours:
2178+12=2190=8760/4
parameter Real delayBeforeMaintenance = 72; // 3 days
parameter Real maintenanceDuration = 24; // 2 shifts
transition
failure: _state==WORKING and _phase==OPERATION -> _state:=FAILED; // the
component cannot fail during tests
startTest: _phase==OPERATION -> _phase:=TEST;
completeTest: _phase==TEST and _state==WORKING -> _phase:=OPERATION;
startMaintenance: _phase==TEST and _state==FAILED ->
_phase:=MAINTENANCE;
completeMaintenance: _phase==MAINTENANCE -> {
_state:=WORKING; // the component is as good as new after the maintenance
_phase:=OPERATION;
}
observer Boolean unavailable = _state!=WORKING or _phase!=OPERATION;
observer Boolean dangerousState = _state==FAILED and _phase==OPERATION;
end
```

A.6 ar3c.exe command list

-h, --help:
    This help.
--version:
    Print version of the tool.
-v, --verbose:
    Print runtime information.
-I <directory>:
    Add <directory> to the compiler's search paths.
-e <filename>, --error <filename>:
    Print errors into <file> (default stderr).
--gts-txt <filename>:
    Print the GTS model at text format into <filename>.
--gts-xml <filename>:
    Print the GTS model at XML format into <filename>.
--main-element <elementname>:
    Define the main element of the model (a block or a class).
    By default: the last declared block.
--structure <filename>:
    Print the struture of the model at XML format into <filename>.
--graphical <filename>:
    Print structured elements of the model at XML format into <filename> (for the graphical viewer).
--ast <filename>:
    Print the Abstrat Syntax Tree of the model into <filename>.
--altarica <filename>:
    Print the AltaRica model of the model into <filename>.
-f <filename>, --flatten <filename>:
    Print the flattened model into <filename>.
-l <filename>, --log <filename>:
Print log information into <filename>.

A.7 Optimization program

```
## Import necessary modules
## -----------------------------------
import re
import subprocess
import os
import csv
import xml.dom
from xml.dom import minidom
import random


"""
change the path to the working directory
"""

os.chdir("D:\Softwares\AltaRica\OARPlatform-win64-
v1.0.0\AltaRicaWizard\Try\periodicallyTestedComponent\Tools\AR3Tools");

## Read XML and store candidate values
## ----------------------------------
"""
Store the parameter data in a dictionary provided
"""
ParaDict={};
def storeParameters(dict):
        #Parse the xml file
        dom = xml.dom.minidom.parse('test.txt');
        root = dom.documentElement;

        parameterNodes = root.getElementsByTagName("parameter");
        for parameterNode in parameterNodes:
                valueNodes = parameterNode.getElementsByTagName("candidate");
                valueList=[];
                for valueNode in valueNodes:
                        valueList.append(valueNode.getAttribute("value"));
                dict[parameterNode.getAttribute("name")]=valueList;


## Modify alt file
## ---------------
"""
Modify the parameter data in the alt file
Set the value of certain parameter to a given value
Input: Parameter name
       New value
"""
def modifyAltFile(VariableName,Value):
        #Regular expression needs modification for different needs
        pattern                                                                 =
re.compile(r"(\s+)parameter(\s+)Real(\s+){name}(\s+)=(\s+)(\d+)".format(name=Vari
ableName))
        #Name of alt file set to the used file
        file = open("periodicallyTestedComponent.alt",'r+')
        lines = file.readlines()
        file.seek(0,0)
        for line in lines:
                if re.match(pattern,line):
                s=re.sub("\d+","{}".format(Value),line)
                file.write(s)
                continue;
                file.write(line)
        file.close();

## Generate gts file
## -----------------
"""
Generate the GTS file for a given alt file
Input: alt file
"""
def generateGTSFile():
        file = open("periodicallyTestedComponent.gts",'w');
        file.close();
        subprocess.call("ar3c.exe --gts-xml *periodicallyTestedComponent.gts*
*periodicallyTestedComponent.alt*");


    ## Generate idf file
    ## -----------------

    """
    Generate indicator description file
    The code block should be modified according to the indicators to be generated
    """
    def generateIDFFile():
        doc = xml.dom.minidom.Document()
        root = doc.createElement('ar3ccp')
        doc.appendChild(root)

        nodeCalculation = doc.createElement('calculation')
        nodeCalculation.setAttribute('observer','unavailable')
        root.appendChild(nodeCalculation)

        nodeIndicator = doc.createElement('indicator')
        #Set attribute for the indicator

        #Type is the recorded value of the indicator
        nodeIndicator.setAttribute('type','sojourn-time')

        #name is MDT, mean down time
        nodeIndicator.setAttribute('name','MDT')

        #value is the monitoring state of the indicator
        nodeIndicator.setAttribute('value','true')
        nodeCalculation.appendChild(nodeIndicator)

        #The node 'mean' means recording the mean value of the monitored
indicator value
        nodeMean = doc.createElement('mean')
        nodeIndicator.appendChild(nodeMean)
        fp = open("periodicallyTestedComponent.idf",'w')
        doc.writexml(fp)
        fp.close()

    ## Generate simulator
    ## -------------------------
    """
    Generate the simulator using the compiling batch file from AltaRica tools
    The AltaRica tool directory need to be set.
    The name of the gts file and idf file need to be set.
    """
    def generateSimulator():
        p = subprocess.call("gtsstocmp.cmd Tools\AR3Tools
*periodicallyTestedComponent.gts* *periodicallyTestedComponent.idf*
periodicallyTestedComponent.exe");


    ## Generate mdf file
    ## -----------------------
    """
    Generate the mission description file
    The result file name can be changed
    The number of runs, seed can be changed.
    The mission time should be set according to need.
    """

    def generateMDFFile():
        doc = xml.dom.minidom.Document()
        root = doc.createElement('ar3ccp')
        doc.appendChild(root)

        nodeSimulation = doc.createElement('simulation')
        nodeSimulation.setAttribute('results-csv','results.csv')

        #Set number of runs
        nodeSimulation.setAttribute('number-of-runs','10000')
        nodeSimulation.setAttribute('seed','12345')
        root.appendChild(nodeSimulation);
        nodeSchedule=doc.createElement('schedule')

        #Set mission time
        nodeSchedule.setAttribute('mission-time','8760')
        nodeSimulation.appendChild(nodeSchedule)

        fp = open("periodicallyTestedComponent.mdf",'w')
        doc.writexml(fp)
        fp.close()

    ## Generate result csv file
    ## -------------------------------

    """
    Conduct the simulation with simulator and mission description file
    """

    def generateCSVFile():
        p = subprocess.call("periodicallyTestedComponent.exe
*periodicallyTestedComponent.mdf*");

    ## Read result
    ## ---------------

    """
    Store the result of simulation and value of monitored indicator after each
simulation
    Input: Dictionary storing result,
            Parameter value
    """

    def storeResult(ResultsDict,Value):
        # Read csv
        csvFile = open("results.csv", "r")
        reader = csv.reader(csvFile, delimiter=';');

        # Build dictionary
        result = {};
        for item in reader:
            # Ingore first line
            if reader.line_num == 1:
                continue
            i=1;
            while item[i]=='':
                i=i+1
            result[item[i]] = item[i+1];

        csvFile.close()
        #In this case,it's mean of sojourn time. It should be reset depending on the
case
        strResult=result['mean']
        ResultsDict["{}".format(Value)]=int(float(strResult))

    ## Do one simulation
    ## ------------------------

    """
    Do the whole process of one simulation with certain maintenance policy
    """

    def simulation(VariableName,Value,ResultsDict):
        modifyAltFile(VariableName,Value)
        generateGTSFile()
        generateIDFFile()
        generateSimulator()
        generateMDFFile()
        generateCSVFile()
        #The result of the simulation stored in ResultDict
        storeResult(ResultsDict,Value)



    ## Main
    ## ------
```

```
"""
Main block of the maintenance optimization process
Metaheuristic algorithms are applied here
"""

#Store the candidate parameter values
storeParameters(ParaDict)
valueList=ParaDict["delayBetweenTests"]
sorted(valueList)
ResultsDict={};
GlobalOptimum={};

#Create the tabu list, length is 10
TabuList=[None]*10;
TabuCount=0;

#Number of runs searching local optimum
n=30;
localoptimal=[];
localoptimalMDT=[];

for i in range(n):
        # Randomly choose the starting point
        startIndex=random.randrange(0, len(valueList),1)
        if startIndex == 1:
                startIndex = startIndex+1
        elif startIndex == len(valueList):
                startIndex = startIndex-1

        #Apply Tabu search algorithm
        if startIndex in TabuList:
                continue;
        TabuList[TabuCount]=startIndex;
        TabuCount=TabuCount+1;
        if TabuCount==10:
                TabuCount = 0;

        MinMDTIndex = 0;
        MinMDT =0;
        run = 0;
        LocalOptimums = {};
        OptimalVariable= None;

        #Search the local optimum until finding it or time expired
        while(MinMDTIndex!=1 and run <=10):
            #Simulations for the chosen value and two teaked values
            for index in [startIndex-1,startIndex,startIndex+1]:
                simulation("delayBetweenTests",valueList[index],ResultsDict)

            #Assess the performance of the three candidate values
            #Store the MDT results of three candidates
            MDTList = list(ResultsDict.values())

            #Find the smallest MDT among the three candidates
            MinMDT = min(MDTList)

            #Record the index of th smallest, 0-2
            MinMDTIndex = MDTList.index(MinMDT)

            #Record the corresponding parameter
            MinMDT_Para = list(ResultsDict.keys())[MinMDTIndex]
            startIndex = valueList.index(MinMDT_Para)    # Change the index of
parameter in valueList

            if(startIndex==(1 or len(valueList))):
                break

            OptimalVariable = MinMDT_Para
            run = run+1;

        LocalOptimums[OptimalVariable] = MinMDT;

    #Find the global optimum solution among all the local optimums
    MDTList = list(LocalOptimums.values());
    MinMDT = min(MDTList);
    MinMDTIndex = MDTList.index(MinMDT);
```

```
MinMDT_Para = list(ResultsDict.keys())[MinMDTIndex];

GlobalOptimum[MinMDT_Para]=MinMDT;
print(GlobalOptimum)
```