h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbmk
FACHBEREICH MASCHINENBAU
UND KUNSTSTOFFTECHNIK

Darmstadt University

- Department of Mechanical Engineering and Plastics Technology -
- Mechatronics course -

Bayesian networks and machine learning for operator support using the example of set-up processes for production plants

Investigation of implementation possibilities of a dynamic machine learning method

bachelor thesis for the attainment of the academic degree

Bachelor of Science (B.Sc.)

submitted by

Julian Schneider

Matriculation number: 760221

submitted on: March 14, 2022 Winter

Semester 21/22

Speaker Prof. Dr.-Ing. K. Kleinmann
Co-referee : Prof. Dr.-Ing A. Weigl-Seitz
Operat. Supervisor : Dr. Ing. Alexander Dementyev
Supervisor :Dipl. Ing. Supervisor :
Dipl. Ing. Albrecht Hänel

Blocking notice

This bachelor thesis for the degree of Bachelor of Science with the title "**Bayesian networks and machine learning for operator support using the example of set-up processes for production plants, *investigation of the implementation possibilities of a dynamic machine learning method***" contains internal and confidential information of the **Fraunhofer Institute for Machine Tools and Forming Technology IWU.**

Inspection of this bachelor thesis is not permitted. Exceptions are the supervising lecturers Prof.Dr.-Ing. K. Kleinmann and Prof. Dr.-Ing A. Weigl-Seitz, as well as the authorized members of the examination board of the Department of Mechanical and Plastics Engineering of the University of Applied Sciences Darmstadt. Publication and duplication of the bachelor thesis - even in excerpts - is not permitted.

Exceptions to this rule require the written approval of the company.
Fraunhofer Institute for Machine Tools and Forming Technology IWU.

Dresden, 14 March 2022

affidavit

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

Department of Mechanical Engineering and Plastics Technology
- The Examination Board of the Mechatronics Course of Studies -

Affidavit,
I, **Schneider Julian born on 22.05.1996**, hereby declare that I have written this thesis on my own and that I have not used any aids other than those indicated.

Insofar as I have made use of external materials, texts and trains of thought, my explanations contain complete and unambiguous references to the authors and sources. All other contents of the presented work originate from me in the copyright sense, as far as no references and quotations are made.

I certify that I have not yet submitted this work to any examination or examination authority.

I understand that an attempt to deceive has been made if the above declaration proves to be incorrect.

Dresden, 14 March 2022

_____
Julian Schneider

Summary

In this thesis, a visual operator support for production plants is investigated and realized by machine learning methods and the application of Bayesian networks. Due to the rapid progress of digitalization, the degree of automation in industry is becoming more and more complex. This development results in the requirement to research the newly emerging topics and to develop timely solutions. High employee turnover on the labor market and the resulting constant rejuvenation of employees, has the consequence that many years of knowledge is lost in production companies. This knowledge is needed to set up plants at the start of production or to localize and eliminate the causes of faults as quickly as possible in the event of process malfunctions. The aim of this thesis is to develop an assistance system which stores the knowledge accumulated over years with the help of machine learning methods and on the basis of Bayesian networks and which can be used to support set-up processes at production plants. In this context, dynamic adaptations of the system are investigated and different algorithms are developed, with which it is possible to determine the causes of errors in case of occurring error patterns. In the first step, general approaches for such an assistance system are examined and discussed. In the first step, general approaches for such an assistance system are investigated and different possibilities and variants are shown for the work packages structure, data collection and generation, parameter learning and error cause determination, before an exemplary realization is implemented using the example of the "setup process of a blow molding machine". Due to a lack of real process data, which is needed for training an artificial intelligence, the topic of generating synthetic data is investigated and both random synthetic data and rule-based data with consideration of dependencies and distributions are generated and simulated. In tests performed later, this data is used. A specially developed user interface is used for interaction between humans and the system. Subtasks of the application are categorized via a menu bar and tabs.

For the two phases "preform"' and "blow molded part" during the blow molding process two Bayes net structures have been built. A corresponding environment has been developed and tested to train the nets. It is possible to determine the most probable or influential cause for an active failure pattern. The user interface has a tab for a general overview of the network and a tab for individual fault patterns. These functions are additional options and can serve as an aid for the setter. The realization and implementation based on the example of the set-up process of a blow moulding machine serves as an advanced concept / first implementation. Tests in real operation do not take place within the work. Functional verifications and a qualitative evaluation of the different approaches are carried out.

Abstract

Visual operator support at production plants by machine learning methods and the application of Bayes networks. Due to the fast progressing digitalization, the degree of automation in the industry becomes more and more complex. Out of this development arises the requirement to research the emerging topics and to develop timely solutions.
The high employee turnover on the labour market and the resulting constant rejuvenation of employees, causes the loss of many years of knowledge in production companies. This knowledge is needed to set up equipment at the start of production or to localize and eliminate the causes of faults as quickly as possible in the event of process malfunctions. The goal of this thesis is to develop an assistance system which stores the knowledge accumulated over years with the help of machine learning methods and on the basis of Bayesian networks and which can be used to support set-up processes at production plants. In this context, dynamic adaptations of the system are being investigated and different algorithms are being developed, with which it is possible to determine the causes of errors when error patterns occur. In the first step, general solution approaches for such an assistance system are examined and discussed. Here, different possibilities and variants are shown for the work packages structure design, data collection and generation, parameter learning and fault cause determination, before an exemplary realization is implemented using the example of the "setup process of a blow molding machine".
Due to a lack of real process data, which is needed for training an artificial intelligence, the topic of generating synthetic data is investigated and both random synthetic data and rule-based data with consideration of dependencies and distributions are generated and simulated. In later performed tests, this data will be used. A specially developed user interface is used for the human-machine interaction. Subtasks of the application are categorized via a menu bar and tabs.

For the two phases "preform" and "blow molded part" during the blow molding process, two Bayes net structures have been built. An appropriate environment has been developed and tested for training the networks. It is possible to determine the most probable or most influential cause for an active defect pattern. The user interface has a tab for an overall view of the network or via a tab for individual fault patterns. These functions are additional options and can serve as help for the setup engineer. The realization and implementation on the basis of the example setup process of a blow molding machine serves as a widely executed concept/ first implementation. Tests in real operation do not take place within the work. Functional verifications and a qualitative evaluation of the different approaches are carried out.

Foreword

This bachelor thesis was written at the end of the mechatronics studies with specialization in robotics at the Darmstadt University of Applied Sciences. The starting point of this thesis is a research project at the Fraunhofer Institute for Machine Tools and Forming Technology (IWU) in Dresden, which is being carried out with an industrial partner. The aim of the project is to develop an interactive assistance system that supports an operator during the setup process of a blow molding machine. A twelve-week practical phase (BPP), which is completed at the end of a bachelor's degree in mechatronics, was also carried out in this project and thus serves as a foundation stone.

My personal interest in data analytics and machine learning as well as artificial intelligence (AI) led me to write my bachelor thesis in this field. Looking into the future, topics like data analysis and evaluation as well as machine learning will become more and more important. The volume of collected data is increasing and it is necessary to evaluate the collected data and information intelligently in order to gain benefit from it. It is a matter of making better use of existing knowledge / existing data.

Due to the current Covid19 pandemic, the work was mainly done away from the institute, as on-site work was prohibited at that time. Since most of the work had to be done in digital form and the Fraunhofer Institute has the necessary infrastructures, editing was possible without major cutbacks. I would like to thank my supervisors Dr. Ing. Alexander Dementyev, Dipl. Ing. Albrecht Hänel and Dipl. Ing. Rebekka Zache for their support and help during the work. Despite the not easy circumstances they enabled me to write the bachelor thesis successfully.

I hope you enjoy reading this work.

Dresden, 14 March 2022

Table of Contents

1. Introduction **1**

1.1. Introduction 1
1.2. Motivation 1
1.3. Task and objective 2
1.4. Structure of the work 2

2. Theoretical foundations **3**

2.1. Blow moulding 3
2.1.1. General 3
2.1.2. Applied process: Extrusion blow molding with squeeze over 6
2.1.3. Differentiation between continuous and discontinuous extrusion 7
2.2. Bayesian networks 8
2.2.1. Static Bayes Net (BN) 10
2.2.2. Dynamic Bayes Net (DBN) 11
2.3. Continuous Machine Learning (CML) 12

3. State of the art **13**

3.1. Application of Bayesian networks for root cause analysis and decision making 13
3.2. Application of dynamic Bayesian networks 15

4. System and software status at start of work **17**

5. General solution approach **21**

5.1. Structure Bayes net 21
5.2. Connection to the machine 24
5.2.1. Snap 7 as a means of accessing process data 24
5.2.2. Comparison of further access options to process data 25
5.2.3. Implementation in code 26
5.3. Data generation for learning and training a Bayesian network 27
5.3.1. Data preparation 27
5.3.2. Generate synthetic data 28
5.4. Parameter learning and Bayes net training 32
5.5. Algorithm Root Cause Analysis 34
5.6. User interface 36

Table of Contents

6. Realisation approaches and implementation 38
6.1. Structure Structure Bayes net ... 38
6.1.1. Bayes net, generation of the preform ... 38
6.1.2. Bayes net, finished blow moulded product ... 39
6.2. Data acquisition, generation and processing ... 39
6.2.1. Machine connection ... 39
6.2.2. Acquisition of states of manual nodes/parameters ... 40
6.2.3. Generation of synthetic data ... 42
6.3. Root cause analysis ... 45
6.3.1. Most probable cause of error ... 46
6.3.2. Most influential cause of error ... 47
6.4. Design user interface ... 48
6.4.1. Tab Total Overview ... 49
6.4.2. Tab process intervention ... 50
6.4.3. Tab Error overview ... 53
6.4.4. Tab Parameters Learning ... 54

7. Evaluation, analysis and discussion 55
7.1. Proof of function Parameter learning ... 55
7.2. Algorithms Cause of error most probable or most influential ... 56
7.3. Proof of function Fault cause determination ... 60
7.4. Comparison algorithm most likely cause with and without dynamic adjustment 60
7.5. Comparison cause determination implemented algorithm at the beginning of the work and newly implemented for most influential cause ... 61
7.6. Evaluation of the different implementations of the fault cause determinations and -fixes ... 63
7.7. Discussion ... 64

8. Summary and Outlook 69
8.1. Use of a dynamic Bayes net ... 70
8.2. Working with real data ... 70
8.3. Increase the degree of automation of data acquisition ... 71
8.4. Algorithm for fault cause determination ... 71
8.4.1. Algorithm for determining the most influential cause of error ... 71
8.4.2. Combination of algorithms most probable or most influential cause of error ... 72
8.5. Automatic retraining ... 73
8.5.1. Cyclic retraining ... 73
8.5.2. Automatic training when a certain amount of data is reached ... 74

Table of Contents

8.6. Introduction Quality Control 74

Appendix **75**

A. Bayes nets **76**

B. Source code **80**

C. Data sets, tables and graphics **87**

Literature **89**

List of Figures

2.1. Overview of various manufacturing processes for plastics [31]..... 3
2.2. Extrusion blow molding process (a) conventional (b) conventional with over- squeezing (c) tube delivery process or manipulation process (d) tube manipulation with robot (e) suction blow molding process [33]..... 5
2.3. Process steps of extrusion blow molding [37]. 5
2.4. Process steps of (a) *continuous* or (b) *discontinuous* extrusion of the melt ([37, p. 40-42]) 7
2.5. Example of a Bayes Net (BN)..... 8
2.6. Example of *dynamic Bayesian network (DBN)* structure (left "prior" BN, right "prior" BN). "transition" BN) [46, Figure 5.23]..... 11
2.7. Example of *dynamic Bayes net (DBN)* structure with $T = 2$ from Figure 2.6 [46, Figure 5.23]..... 12

4.1. Architecture and Function *Shiny* 17
4.2. Bayes net structure section of the *preform* phase for the defects "tube run straight" and "preform shiny" at the beginning of the work 18
4.3. Flow chart process of root cause analysis and troubleshooting, status handover/start of work..... 18
4.4. Excerpt from Error Detection and Root Cause Analysis GUI using the example of the error image *Scratch*, Stand Start Work..... 19
4.5. Section of the user interface, "Tab overview", status of start of work..... 20

5.1. Development flow of an interactive system, ML (Bayes net) based..... 21
5.2. Flowchart creation Bayes net structure 22
5.3. Conceptually created Bayes net structure according to source code 5.2, nodes irrelevant . 23
5.4. *Snap7* Architecture of a *client* object 24
5.5. Machine interface approach using *python-snap7* 25
5.6. Flowchart: Machine connectivity and data acquisition approach using *python- snap7*..... 26
5.7. Structure of a machine learning process (based on [55, chapter 1]) 28
5.8. Model of a data generator (based on [17]) 29
5.9. Flowchart for generating random *synthetic data* 30
5.10. Sequence of simulated data using *rbn()*, node *scraper* is the end node. 31
5.11. Trained Bayes net *scratches* with random synthetic data from Table 5.3 . 33

5.12. Flowchart Algorithm for determining the *most probable* cause for the occurrence of a fault 35
5.13. Flowchart Algorithm for determining the *most influential* cause for the occurrence of a fault 35
5.14. Flowchart algorithm to determine the *most probable* cause for the occurrence of a fault, as an iterative process for dynamization. 36
5.15. Concept User interface Tab for root cause analysis process 36
5.16. Concept User interface Tab for root cause analysis process 37

6.1. Process of data acquisition using "snap7 40
6.2. Flowchart to capture the states of manual nodes during the process of the troubleshooting 41
6.3. Manually learned subnet for the errors "scratch", "preform shiny" and "Hose run straight"; Software *GeNIe* 44
6.4. Tab overview from the user interface of the extrusion blow molding knowledge base *ShinyApp* 48
6.5. Tab *General overview* from the user interface of the Extrusion Blow Molding Knowledge Repository, selected Bayesian network "Preform". 50
6.6. Tab *process intervention* from knowledge store user interface extrusion blow molding, with algorithm "most likely" cause, Bayes net "preform". and error pattern "Hose running straight". 51
6.7. Tab *Process intervention* Modal window; (a) Query of the present state of a manual parameter, (b) Query of the manual parameters at the end of the process.
Troubleshooting process where the condition is unknown to the system 52
6.8. Tab *Error overview* from the user interface of the Extrusion Blow Molding Knowledge Repository; selected error category "blow molded part" and error pattern "slug waste" . 53
6.9. Tab *Parameter Learning* from the User Interface of the Knowledge Repository Extrusion Blow Molding; Selected Bayesian Net "Preform 54

7.1. Comparison of distributions after parameter learning, manually learned Bayes net, training dataset created with *rbn()* 56
7.2. Conditional probability table of the node "scratch" after parameter learning . 56
7.3. Calculations of the assistance system to determine the *most probable* cause of the error, results sorted in descending order. 57
7.4. Process of determining the cause of faults and correcting faults 60
7.5. Flowchart Algorithm Determination Root cause determination at the beginning of the work 62
7.6. Fault pattern *wrinkles in the article* from main Bayes net "blow-molded part" with nine edges
on the error node 66
7.7. Introduction of intermediate nodes using the example of heating zones for machine component head 67
7.8. Determination of the most influential cause Error pattern "Article hangs in shape", error

pattern
trained with random synthetic data 68

8.1. Example cause determination algorithm most influential including automatic state adjustment of the real process ... 72
8.2. Flowchart algorithm to determine the most influential cause of failure, 100% automatic data collection ... 72
8.3. Flowchart of the combined algorithms for determining the cause of the error (most influential and most probable) ... 73
8.4. Flowchart cyclic retraining integrated into root cause identification process 73

A.1. Bayes net structure for "preform" phase ... 76
A.2. Bayesian network structure for phase "blow-moulded part ... 77
A.3. Manually learned Bayesian network, section of error image "Hose run-straight run-out", bar chart representation using software GeNIe ... 78
A.4. Manually learned Bayes net with occurred error, section error image "Hose runstraight runout", bar chart display using GeNIe software ... 78
A.5. Manually learned Bayesian net with occurred error and node "TempHead" in "niO", section error image "Hose runstraight runout", bar chart representation by means of software GeNIe ... 79
A.6. Manually learned Bayesian network highest probability for occurrence of the error under "CenteringDuese" in "niO", section of error image "Hose run straight-out", Bar chart display using GeNIe software ... 79

C.1. Excerpt training dataset for function proof "parameter learning", created with function *rbn()* from manually learned Bayes net ... 87
C.2. Example benchmarking: Number of defect causes until defect elimination after number of training runs for the defect patterns scratch, preform glossy and hose run straight ahead 88

List of tables

2.1. Overview of advantages and limitations of blow molding [59]. 6

5.1. Excerpt from the matrix for the creation of the Bayes net structure 22
5.2. Extract from classification for *TempDuese* data set 28
5.3. Extract (10 out of 1,000,000) from the generation of random synthetic data for the error *scratch* 30
5.4. Extract of synthetic data generated with the function *rbn()* for the error *Scratch* 31

6.1. Extract from file "Parameterliste.xlsm", matrix structure Bayes net "Preform". 38
6.2. Extract from file "Parameterliste.xlsm", Matrix Structure Bayes Net "Blow Moulded Part 39
6.3. Example of state derivation of manual nodes with error pattern scratch, red states have been derived by function *filldataset()* 41
6.4. Extract from extended file "Parameterliste.xlsm", matrix structure Bayes net phase "blow mould part" and possible states of a node 43
6.5. Extract of a *CPT* entered manually, node "scratch 44
6.6. Extract from synthetic dataset simulated using function *rbn()* for manually learned Bayes net with three errors 45
6.7. Output of the most probable cause of error for the "Scratch" error pattern 46
6.8. Output of the most probable cause of error in the case of the "Scratch" error pattern 47
6.9. Overview of adjustments and enhancements within the user interface, division into *user interface and server* 49

7.1. Protocol "Parameter learning" function verification 55
7.2. Results from calculating conditional probabilities with evidence → *Hose runstraightout = Yes & TempHead = niO* 58
7.3. Results from calculation of conditional probability with event → *Hose runstraight runout = Yes*, to determine the most influential cause of error 59
7.4. Determination of the most probable cause in the case of the "scratch" fault pattern, in the case of the second before-
says node state of "TempDuese" taken into account 61
7.5. Most influential cause for error pattern "hose runout"; (**left**) newly implemented algorithm, (**right**) old algorithm at the beginning of the work 62
7.6. Qualitative evaluation scheme of existing algorithms and implementations for fault detection and correction 63
7.7. Measurement of the computing time of the implemented algorithms 64

C.1. Record of function verification "Determination of cause of error.....88

Source code directory

5.1. Code example in **R** using formal equation to create a DAG 21
5.2. Code example in **R**; creating the complete Bayes net structure as a concept 23
5.3. Connection to the machine, code example *python-snap7* Connection setup and request Temperature 27
5.4. Traninating a Bayes net structure using *bn.fit()* 34

6.1. Generation of random synthetic data 43
6.2. Generation of rule-based synthetic data using the "bnlearn" function *rbn()* 45
6.3. Determining the cause of the fault in the case of the "scratch" fault pattern, algorithm for the most probable cause of the fault 46
6.4. Determining the cause of the fault in the case of the "scratch" fault pattern, algorithm for the most influential cause of the fault 47
6.5. Tab *Total Overview* User Interface Shiny 50
6.6. Tab *Process Intervention* User Interface Shiny, Adjustments and Enhancements 52

B.1. Function to classify numerical data into three classes *low, medium* and *high* 80
B.2. Function *excel2ms()* and *createDAG()* to create a DAG from existing Excel spreadsheet 80
B.3. function *simulateRandDataset()* to generate random synthetic data for an error Subnet 81
B.4. function *simulateRandDatasetBig()* for generating random synthetic data for an entire Bayes network 82
B.5. Connection to the machine, concept code *python-snap7* Connection setup and data query 82
B.6. function *filldataset()* completes incomplete datasets by derivation of states of manual nodes 83
B.7. Function *causeanalysis_likeliestdyn()* to determine the most probable cause of the error 84
B.8. Tab *Total Overview* Server Side Shiny 85
B.9. Tab *error overview* server side shiny, adjustment selection error category 85
B.10. Tab *parameters learning* server side shiny, customization selection main Bayes network . 86

List of abbreviations

ASM Abnormal situation management (management of exceptional situations)

BN Bayesian networks

BPE Bayes parameter estimation

BPP Practical phase

CML Continuous machine learning (CML)

CPT Conditional probability table (table of conditional probabilities)

DAG Directed acyclic graph (directed acyclic graph)

DB Data block

DBN Dynamic bayesian network **GUI** Graphical user interface **IP** Internet protocol

IWU Fraunhofer Institute for Machine Tools and Forming Technology IWU

AI Artificial Intelligence

ML Machine learning

MLE Maximum likelihood estimation (Maximum Likelihood Estimation)

MPI Message passing interface

OPC UA Open platform communications unified architecture

PPI Point to point interface **RCA** Root cause analysis

PLC Programmable logic controller

TBNE Temporal Bayesian network of events

ZIM Central innovation programme for small and medium-sized enterprises

1. Introduction

1.1. Introduction

With advancing digitalization, the degree of automation in production companies is increasing. Experienced employees are still necessary to handle many tasks that are not yet fully automated. In the course of demographic change and high employee turnover, many machine operators lack the necessary experience. This problem can be addressed with the help of an assistance system that makes decisions based on machine learning methods and thus provides support.
As an exemplary process, the production of 3D plastic hollow bodies is investigated. The goal is to develop an interactive assistance system that provides the operator with virtual support via a user interface during the setup process of a blow molding machine. The aim is to ensure that the set-up process is as error-free as possible, and that there are short downtimes due to indications of the causes of errors from the assistance system. If an error occurs during a setup process, the assistance system gives the operator visual suggestions as to what the cause of the error is assumed to be. The search window for the cause of the error is thus minimized and even inexperienced employees are able to correct errors in a short time.

The prediction of the assistance system is based on probabilistic methods, which are determined using Bayesian networks (BN). BNs are mostly used for cause diagnosis in process monitoring. There, a combination of historical and current process data as well as expert knowledge is used [38]. BN's are graphical models that represent probability distributions and dependencies between nodes. Based on collected previous data, the most probable cause of failure is determined offline using inference. The prediction based on offline data collection is designed and developed at this stage. In this bachelor thesis a connection to the machine is to be established, the current machine state is to be recorded and this is to be integrated into the process of prediction. Decisions and predictions are to be made condition-oriented. The error that has occurred is no longer the only input parameter.

1.2. Motivation

Big Data is a term that will be familiar to a large number of people by 2022 at the latest. Due to the desire for competitive advantages and the constant automation of production processes, the amount of collected and stored data is increasing day by day. Now it is a matter of using this to correctly analyze and evaluate "stored knowledge". With new methods from the

AI world, scientists are trying to obtain more unambiguous and clearer results from the data. In connection with a real production process and the networking of mechanical engineering, computer science, but also automation, this work is projected to be an exciting and also important topic in the future.

1.3. Tasks and objectives

The subject of this bachelor thesis is the investigation of the implementation possibilities and the conceptual design of a Bayesian network, which can be used to support the setup process of the exemplary selected process blow molding.
The possibility of using so-called dynamic Bayes networks (DBN) in the area of knowledge storage and assistance systems should also be taken into account here. As an example, currently known, process-relevant influencing factors that can be addressed during the setup process and implemented in a Bayesian network are to be worked out on the basis of the sub-processes (thermal process for generating the so-called preform and main stage of molding in the mold and quality control on the finished blow-molded part at the end of the setup process).
It is to be determined how the features and production data for setting up machines can be made available and how a possible dynamic adaptation of Bayesian networks over time could be implemented. In addition, it is to be investigated how the Bayesian network can continue to learn during the setup phases with the help of inputs from the machine setters (continuous machine learning literature research). The network structure (i.e. the predefined influencing factors, as well as the associated product or preform errors) should be recorded. Possibilities to extend the existing database by generating synthetic training data sets shall be investigated.
Finally, a prototypical implementation and realization as well as tests and evaluations of the developed solution are to take place in a suitable development environment. Essential findings will be documented.

1.4. Structure of the work

At the beginning of the thesis, technical and theoretical basics are presented and conveyed before the current state of the art is described in more detail. The following chapter deals with the software and system status at the start of the work, which will be built upon later.
In chapter 5, a general approach to the problem is discussed and demonstrated. Subsequently, an exemplary implementation and realization takes place using the example of the setup process of a blow molding machine. At the end of the thesis, the results are evaluated, analyzed and discussed, before a summary and an outlook are presented in the final chapter.

2. Theoretical foundations

At the beginning, the process of blow molding (extrusion blow molding), which is used as an example in the practical part, is discussed. Subsequently, the basics of Bayesian networks and dynamic Bayesian networks are presented. In the following chapter state of the art (3) the current use and the current limits of the application are discussed. If you are interested in more detailed information, please refer to the given literature.

2.1. Blow molding

General principles about blow molding, the blow molding processes used and the process used here in the project are summarized. In the last section, differences between continuous and discontinuous extrusion are explained.

2.1.1. General

Figure 2.1 shows an overview of various common manufacturing processes with plastics. In the area of *prototyping*, both in the "continuous processes" division and under the "discontinuous process" to find the *blow molding*.

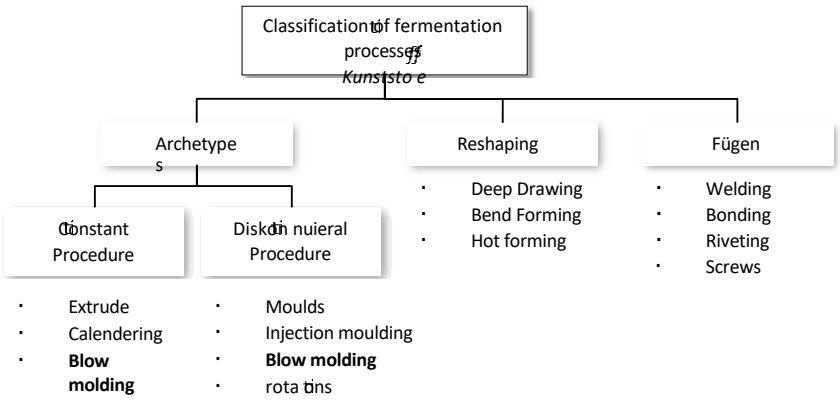


Figure 2.1: Overview of different manufacturing processes for plastics [31].

Blow moulding is a process used to produce thermoplastic hollow bodies [59, Chapter 21.1]. *Helmut Schüle* defines "blow moulding" in [40, p. 274-275] as a process in which *extruded, extrusion-blown or injection-moulded* tubular or test-tube shaped preforms are formed into hollow bodies by means of blowing air in cooled blow moulds. *Bruder*, however, distinguishes in [59] between two main variants. Variant one in which a *preform* is extruded and then inflated and formed between two tool halves.

Variant two, in which an injection-molded heated *preform* is inflated between two mold halves. The production of PET bottles serves as an example for variant two. Here, the preform is produced and cooled in a separate process before the actual bottle is produced in a subsequent second process. Plastics with a high "shear viscosity" [40], such as PE, PP, PVC, PET or PA, are suitable for optimum products [59]. In principle, plastics should have the property that the preform can support its own weight during the extrusion process without showing any significant changes in length or uncontrolled sagging [40].
In this paper, the main focus is on *extrusion blow molding,* since the process used later in the paper is from a machine that uses this process. With this process it is possible to produce products with a capacity of a few millilitres up to 10,000 litres [40]. At the present time, this process has the most important economic significance [37, Chapter 2]. In recent years, however, *stretch blow molding* has continued to grow in importance. This process focuses on mass production (e.g. PET bottles), whereas extrusion blow molding focuses on a wide range of products [37, Chapter 1.3]. Examples there are transport and packaging containers (bottles, canisters, barrels, etc.) or foldable transport boxes, but also technical blow-moulded parts for motor vehicles (plastic fuel tanks, fuel filler pipes, spoilers, dashboards, etc.).

There are various special processes for extrusion blow moulding, which are required for special or complex products:

- *Conventional extrusion blow moulding*
- *Conventional extrusion blow moulding with squeezing over*
- *Tube depositing process* or *manipulation process* - The preform is brought into the appropriate moulding position by means of moulding elements/grippers, so that it has approximately the later contour.
- *Tube manipulation with robot* or *insertion process* - The preform is completely inserted into the mould by a gripper/robot, thus complex 3D contours are possible.
- *Suction blow moulding* - The preform is extruded or sucked directly into the contour of the closed mould halves.

[40, S. 278][33][13]

Figure 2.2 shows the different extrusion blow molding processes. They differ not only in the different tools required and the possible degree of complexity of the mould contour, but also in the excess of material that is squeezed over and thus lost. The aim is to reduce the amount of material that is over-squeezed, as it is a waste product, but also increases the required clamping force [13]. It is clearly visible that with processes
(b) there is the greatest consumption of resources, compared to the purely conventional process (a)

but only slightly increasing the complexity of the mould contour. Processes (d) and (e) have the greatest possible degree of complexity.

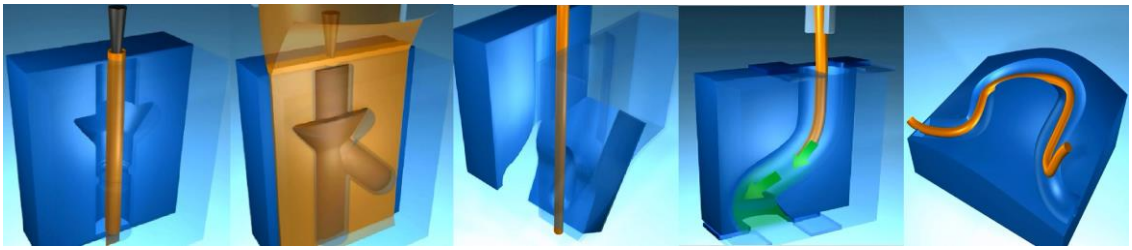


Figure 2.2.: Processes of extrusion blow molding (a) conventional (b) conventional with squeezing over (c) hose delivery process or manipulation process (d) hose manipulation with robot (e) suction blow molding process [33].

In the following, the extrusion blow molding process will be broken down into sub-steps and considered separately (Figure 2.3). In this work, the conventional extrusion blow molding with squeezing is used in the later course, but each process has the identical process steps in the basic features. Thus, the process described below can be assumed to be generally valid.

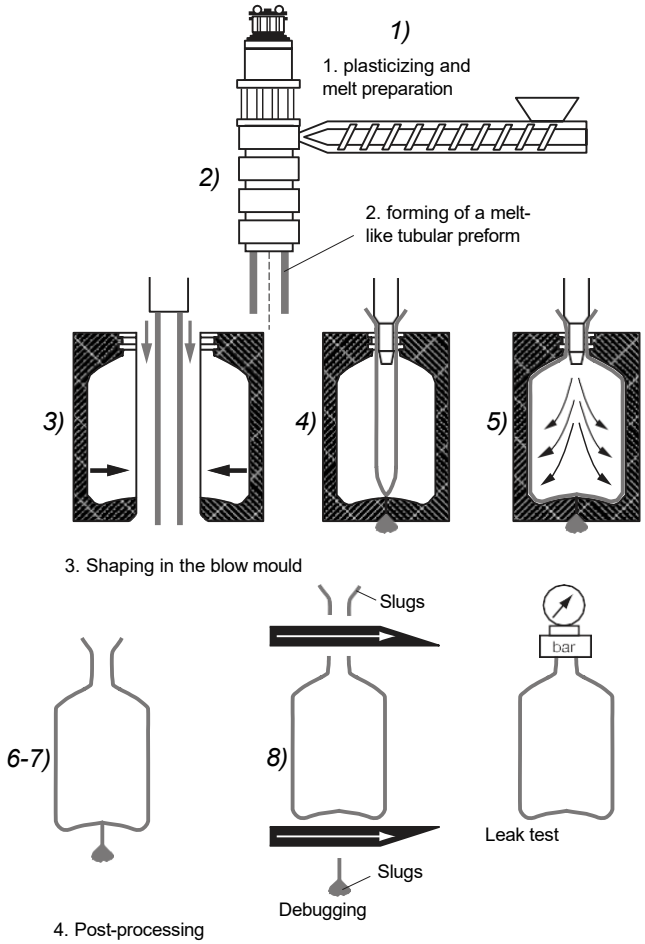


Figure 2.3: Process steps of extrusion blow molding [37].

Process steps plastic extrusion blow moulding

1 Heat plastic in an extruder and plasticize; provide thermoplastic melt.

2 Divert horizontal melt flow into vertical flow and produce a tubular *preform* from the head.

3 Close the blow moulding tool (two-part) around the preform, squeezing the upper and lower ends (if necessary also at the sides).

4 Retract the blow pin/blowing needle(s) from above or below.

5 Using compressed air, blow up the preform against the inner wall of the blow moulding tool, where the plastic cools down, solidifies and takes on its final shape.

6 Vent the blow mould and blow moulding part.

7 Opening of the blow mould halves and removal of the blow moulded part.

8 Top and bottom removal of over-squeezing/bruising and further finishing if necessary.

[37, 13]

Possible product manufacturing processes usually have advantages at first glance, but a more detailed look also reveals limitations and possible disadvantages. Table 2.1 shows the advantages and limitations of blow molding.

Table 2.1: Overview of advantages and limitations of blow moulding [59].

Advantages of blow moulding	Limits of blow molding
Production of large products possible	Not all plastics are suitable
Products with complex shapes possible	high machine and plant costs
Thin-walled products can be produced	Surface quality relatively low
Material combinations (hard and soft plastic) possible	Problems with tight tolerances

2.1.2. Applied process: Extrusion blow moulding with squeezing over

Extrusion blow molding with squeezing is the most trivial manufacturing process next to conventional blow molding without squeezing. The aim is to keep over-squeezing as low as possible. However, in extrusion blow moulding with over-squeezing, this is exactly what is accepted. By using a simple process, it is nevertheless possible to realize "more complex" mould contours. Due to a large percentage of material that is over-squeezed, the required clamping force increases sharply [33]. Due to the large consumption of resources, a correspondingly large

extruder is required [13]. Figure 2.2 (b) shows the process. There it can be seen that a preform with a diameter corresponding to the maximum width of the hollow body to be produced is required. This makes it possible to completely cover the contour despite a "vertically hanging" tube.
The mold is vented via the vent holes in the contour depths. The preform is fixed and stretched by the hose pinches at the upper and lower ends and by the rest of the mold area. The areas where over squeezing takes place have the maximum wall thickness, it decreases towards the contour depth. The hollow part has a full circumferential pinch weld, which requires rework in addition to a weak point. [13]

2.1.3. Differentiation between continuous and discontinuous extrusion

In the process used, a distinction is made between two operating modes depending on the hollow body to be produced and the plastic used. On the one hand there is *continuous extrusion* and on the other hand *discontinuous extrusion* [37].
Hollow bodies with a capacity of about 25 litres or a mass of up to two kilograms, are produced with continuous ejection (Figure 2.4(a)). Sufficient extru- der throughput limits the maximum output [40]. Tough thermoplastics are used, which have low mass. High masses or less tough plastics would promote an unacceptable change in hose length or wall thickness [33]. Thermally plasticized melt flows continuously out of the head and is formed into a preform. After reaching its required length, this is cut off and moved to the side so that the next preform can be produced without interference [40]. The length or ejection speed of the preform is strongly dependent on the extruder speed.

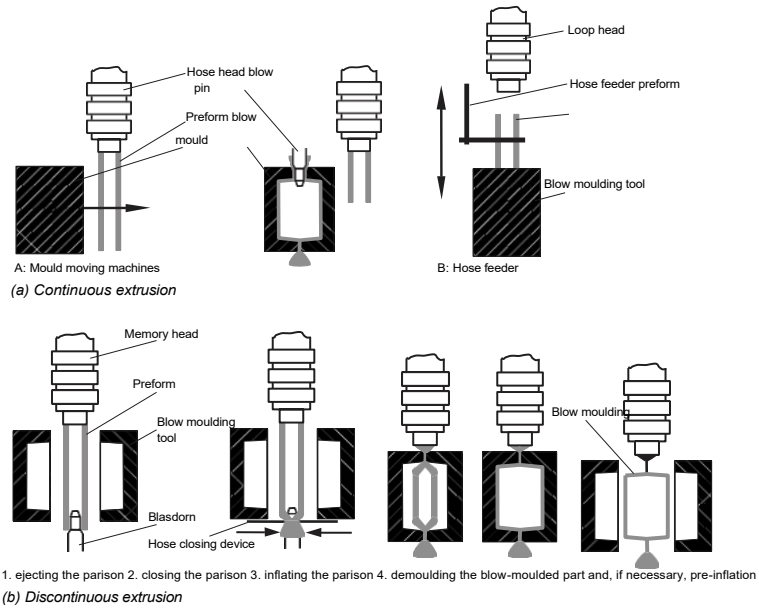


Figure 2.4: Process steps of the (a) *continuous* or (b) *discontinuous* extruding of the melt ([37, p. 40-42])

If large hollow bodies with capacities of up to 10,000 litres are required, dis- continuous ejection is used in the production of the preform. The extruder output is too low for this size and the preform would elongate and cool uncontrollably due to the required duration [40]. In the case of discontinuous ejection, one works with an accumulator (battery). The required melt is collected within the accumulator. This is then ejected in a shock-like manner, where it forms the preform. In addition to a high mass, thermoplastics used usually have a low toughness [33]. In Figure 2.4
(a) the individual process steps are shown. *Thielen* presents in [37, p 41](Table 2.1) an overview for application cases of both modes of operation.

Blow molding and the production of 3D plastic hollow bodies describe a supposedly "one-size-fits-all" process, whereby this process has many uncertainties when looked at in more detail. With recorded and stored process data, an analysis is to be carried out in order to generate maximum stability. The use of *Bayesian networks* as a data analysis tool should help to increase the quality. It is possible to use such a structure to show causal relationships and influencing factors and relate them to process stability. [36]

2.2. Bayesian networks

In the late 1970s, Bayesian networks and their use found their way into computer science, where they were preferentially used in the field of AI [32]. BNs can be found where logic and uncertainty/unknowledge are present at the same time. They combine probabilities of events with *dependence* or *independence* from each other [51, p. 33]. BN are used in many areas, e.g. in medical diagnosis, speech recognition and image processing, but they are also increasingly used in process technology. The name is characterized by three essential points:

- Subjective nature of input information
- Use of Bayesian formulas
- Distinction between causal and evidence-based reasoning

[32, S. 57]

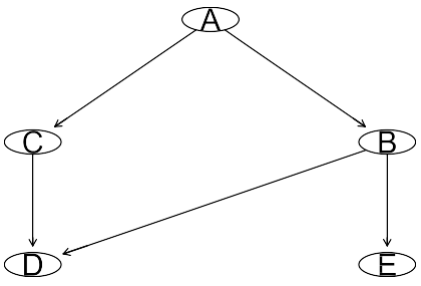


Figure 2.5: Example of a Bayes Net (BN)

Bayesian networks are so-called graphical models (Figure 2.5), which represent probabilistic relationships (*dependencies or independencies*) between variables (nodes).
[48] and have their origin in statistical modeling. According to *Kraaijeveld,* BNs have a qualitative and a quantitative component [41]. He refers to the acyclic directed graph (DAG) as qualitative, which typically reflects the causal relationships and structure of a domain. The quantitative part, on the other hand, is described by the BN through the joint probability distribution of the variables (nodes). Each variable or uncertain quantity represented by a node A_i has a conditional probability table (CPT) for two or more possible values [42, chapter 2.2.5]. There, states are defined in terms of their probability depending on higher-level variables (nodes). If a variable (node) is independent, then the prior probabilities can be found in its CPT [41]. To represent dependencies, a node A_i is represented by an edge with a
node $par(A_i)$, which is the cause of A_i . The arrowhead of the edge is located at the end of the node A_i . $par(A_i)$ is designated as the *parent* node and A_i as the *children* node [42, Chapter 2.2.5].

For calculations at the BN, formulas and equations from the field of classical and conditional probability theory are required. *Bayes' theorem* serves as a basis for further calculations.

Sentence 2.2.1
Bayes' theorem applies (based on Figure 2.5):

$$P(B|A) = \frac{P(A|B) \cdot P(B)}{P(A)}$$

[46, S. 12]

In addition to *Bayes' theorem,* the statement about *conditional independence* is also of great importance.

Sentence 2.2.2
Conditional independence holds (for E is independent of A if B is given; based on Fig. 2.5):

$$P(E|B) = P(E|B, A)$$

[46, S. 11]

If joint probability distributions are computed over the nodes $\{A_1, ..., A_n\}$, then the product of all *prior* and *conditional* probability distributions is formed.

Sentence 2.2.3
Joint probability distribution over $\{A_1, ..., A_n\}$ holds:

$$P(A_1, \ldots A_n) = \prod_{i=1}^{n} P(A_i | par(A_i))$$

[60, S. 8]

If Theorem 2.2.3 is applied to Figure 2.5 to calculate the joint probability distribution of all nodes $\{A_1, ..., A_n\}$, this is written as follows:

$$P(A, B, C, D, E) = P(A) \cdot P(B|A) \cdot P(C|A) \cdot P(D|B, C) \cdot P(E|B)$$

For reasons of comprehensiveness, further basics of probability theory are omitted. For more detailed information, please refer to the following literature [46], [45, Chapter 24] and for application to [60].

Bayes networks can be divided into two categories:

- Static Bayes Net (BN)
- Dynamic Bayes Net (DBN)

In general, however, Bayesian networks must be modeled and learned or trained before they can be applied. It is thus possible to make predictions through inference calculations and algorithms.

2.2.1. Static Bayes Net (BN)

The advantage of BN lies in the handling of uncertain information. Through dependencies and "knowing" other nodes, it is possible to infer the uncertain / unknown node. Also of advantage is that through graphical visualization, complex relationships can be represented and applied in a way that is understandable to the viewer. Based on these properties, the Bayes net is particularly suitable for the determination of error causes and anomalies (diagnostic area) [41].

Static Bayesian networks neglect a possible time factor and treat the nodes as discrete. This means that values are used at a concrete point in time / time span. If modeling is considered, the developer is given three different methods to implement it. In the case of modeling, both the qualitative and quantitative parts of the BN are developed.

M.1: Structure (qualitative part) and parameters/ CPT (quantitative part) are defined by experts

M.2: Structure (qualitative part) and parameters/ CPT (quantitative part) are defined automatically by means of algorithms from existing data

M.3: Structure (qualitative part) and parameters/ CPT (quantitative part) are defined by experts and data (combination of methods 1 and 2) [48].

Automatic modeling using data is only possible if a very large amount of data is available. For new applications, this is usually not the case. Therefore, method 1 is used in most cases, which is very time-consuming. Subject matter experts and "knowledge engineers
[41] define nodes and dependencies independently.

2.2.2. Dynamic Bayes Net (DBN)

In the previous section, the *static BN* was described, in which a concrete point in time is considered. Temporal relationships between nodes (variables) are not considered or modeled. It is possible, for example, that the variable value at time t *is* dependent on the value at time t_1 . These dependencies can occur within a node, but also among each other. In real applications it is often necessary / advantageous to model a reference to the temporal track. In medicine, for example, the temporal aspect is mostly relevant to make correct statements in diagnoses and prognoses or decisions in treatment [46, p. 272]. Dynamic Bayes Networks (DBN) now offer these possibilities. They extend the static BN by a temporal component t and link different points in time with each other [48, p. 27].

A DBN is composed of two components:

1 An *initial Bayesian network* (Figure 2.6 left) consisting of a DAG and CPTs. This part corresponds to a static Bayesian network.

2 A *transition Bayesian network* (Figure 2.6 right), where temporal dependencies between nodes are defined and specified.

The underlying mathematical equations and further information can be found in [46, Chapter 5.3.1].

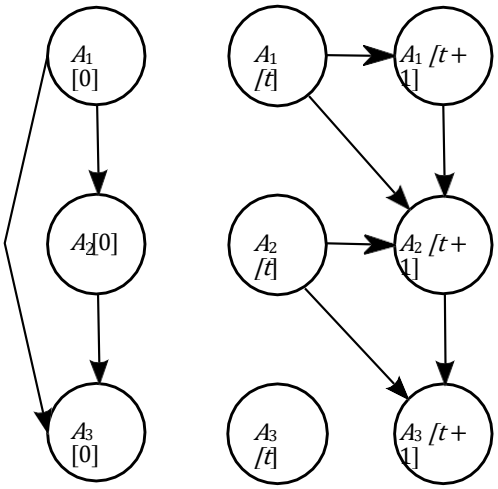


Figure 2.6: Example of *dynamic Bayesian network (DBN)* structure (left "prior" BN, right "prior" BN)
"transition" BN) [46, Figure 5.23].

If the two components are combined, the result is the dynamic Bayesian network (example Figure 2.7). Thus, it is possible to model a temporal course including temporal dependencies. For the prediction of the state at time t, only the description at time t - 1 is required. All necessary information is contained there [46, p. 274].

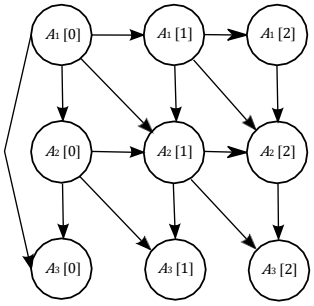


Figure 2.7.: Example of *dynamic Bayes net (DBN)* structure with $T = 2$ from Figure 2.6 [46, Figure 5.23].

For inference computations, the algorithms known from static Bayes nets are used. However, if a process runs for a very long time, the dynamic Bayes net grows rapidly and the running time of the algorithms used becomes inefficient. *Neapolitan* describes in [46, pp. 275-276] a way to circumvent this problem. A dynamic Bayes net becomes a *static one if* the transition probability *distributions* do not change between different time steps [60, p. 103]. As an example of a DBN, *Kjærulff* uses a network to make inferences about the cause of leaf loss in an apple tree. In [1], *Ropponen* investigates ways to improve paper production quality and operator interactions with stochastic evidence. Calculations and predictions are made on a large amount of measurement data using DBN.
Created networks need to be trained and parameters need to be learned. Performing this process once would leave an application on a possibly outdated state, therefore it is necessary to continuously renew and extend the knowledge states. In this context, the term *continuous machine learning (CML)* can be found in the literature.

2.3. Continuous Machine Learning (CML)

Machine learning applications are trained with training data and they expect the application data to match the training data. In reality, however, this is often not the case. A solution to this problem can be found in CML. A model is continuously retrained with updated data and thus optimally adapts to its environment [22]. *Kleinings* in [22] points out the difference between *continuous training* and *continuous machine learning*. Continuous training is applied in cases where the data shows "drift" and thus the predictions of the model used become unreliable. There, continuous and automatic re-training is used to respond to the changes. CML, on the other hand, offers a wide range of possible *tools* for further automating machine learning processes. Among other things, it is possible to automate the training or evaluation of models. *Liu* summarizes in [7] the relevance and importance of continuous learning. However, this topic is not the focus of this paper. In the section Outlook it is shown how a possible implementation / concept for the application could be designed.

3. state of the art

This chapter presents the state of the art of Bayesian networks for the use of fault detection and root cause analysis and decision making processes in different contexts and domains. Known from the chemical industry, these are the first two steps of exceptional situation management (ASM).
The three steps are:

(1) early/ timely detection of a fault/ malfunction

(2) Diagnosis of the "root" cause

(3) Restoring the normal operating/process state

[43]

3.1. Application of Bayesian networks for root cause analysis and decision making

Production plants and industrial systems are becoming increasingly complex, whereas the number of experienced personnel in a company is decreasing due to the high fluctuation of employees on the labor market. Required expertise and knowledge are lost. The demands on monitoring and diagnostic systems of production plants are increasing. There is a growing desire and need for systems that can be used to analyze the causes of faults and make decisions on how to correct them. The goal of such a system is to identify failure and process malfunction causes and make automatic conclusions/predictions for remediation [20, pp. 1-2]. The advantage over decision trees is the reduced memory and computation time requirements.

Many research papers deal with the topic of root cause analysis in the event of a fault and use Bayesian networks as the basis for this method. In [61], for example, a method for root cause analysis based on Bayesian networks is developed and tested. Here possibilities are examined, with which a cause can be derived from the available data sets. This approach is also investigated and tested in this paper. With the help of collected data from past processes, Bayes nets are to draw conclusions about possible causes of errors and make dynamic adjustments during the process.
In [20] the possibility of an automatic root cause analysis of errors and faults in

rolling mills are investigated. Bayesian networks are used for decision making. Furthermore, it is described that graphical models based on probabilistic methods, such as Bayes nets, prove to be the most suitable method and technology. A great advantage is the transparency and comprehensibility of the reasoning through the graphical appearance.

The scheme and procedure are independent of the area in which a root cause analysis is to be built. In the first step, the structure of the subsequent Bayesian network is built. Two different methods are used for this. On the one hand the structure is learned from existing data and on the other hand the structure is defined by expert knowledge. In the next step the conditional probability tables (CPT's) of the nodes/variables are determined. This is also done on the basis of the existing data. In the last step, inference calculations and algorithms are used to finish the root cause analysis and fault prediction. Based on probability distributions, it is possible to predict a fault cause. In [61], a combination of Bayesian network and "fuzzy cognitive map" is used.
In the area of predictive and preventive maintenance, the use of Bayesian networks is being tested and researched [48]. In his diploma thesis, *Ralf Pühringer* deals with the goal of developing a *probability-based event detection and monitoring system* for predictive maintenance.
-analysis model.

When it comes to root cause analysis and troubleshooting, the term "root cause analysis (RCA)" is usually used in this context. This is an umbrella term for several different methods that deal with the systematic localization and elimination of an error or cause [11].
In the field of medicine, Bayesian networks are also frequently used to determine causes and to support decision-making. For example, *van der Gaag* in *Proba- bilities for a probabilistic network: a case study in esophageal cancer addresses* the possibility of using Bayes networks to develop patient-specific therapy selection [30]. In addition to *van der Gaag*, *Flügge* also deals with the use of Bayesian *networks* in *the* area *of* diagnosis, therapy and medication recommendations in *Knowledge Representation and Diagnostic Inference Using* Bayesian Networks in *Medical Discourse* [50].

Interactive and intelligent assistance systems based on BN have a wide range of applications in the field of driver assistance systems. In addition to braking and acceleration behavior, steering behavior is also investigated using so-called "lane change systems". In [12] *Kasper* deals with the improvement of adaptive cruise control and the detection of lane change situations. In [26], *Rüdenauer* investigates the possibilities of confirming the actions of an autonomous mobile robot with the help of Bayesian networks and its predictions.

3.2. Application of dynamic Bayesian networks

Further, failure diagnosis/cause analysis is about determining the "root" cause of a failure, whereas failure detection determines whether a process is in a normal or abnormal state. Early detection helps in timely execution. Diagnosis guides the operator to the faulty part of the process in which intervention is required [38]. The application of "Dynamic Bayes Networks" is an extension of the concept of Bayesian networks in terms of a tempolar process and dependencies [3]. In areas such as medicine, industry, speech recognition, planning or prediction, the timing of an observation plays an important role for predictions and diagnoses [18].
Brandherm investigates the possibility of detecting a user's stress or time pressure using biosensor and environmental data in dynamic Bayesian networks [6]. In his model, new time slices are instantiated by event-driven instantiation, triggered by the detection of speech utterances or device inputs. This creates a temporal dependency. The creation of new time slices makes the network grow continuously, which causes an increasingly longer runtime for computations and a more extensive memory requirement [6].

In [38], fault detection and root cause analysis are carried out and tested on the basis of dynamic Bayesian networks, among others. As exemplary examples, *Amin* uses processes on a "binary distillation coulumn" and on a "continuous stirred tank heater". The comparison to static BN shows that DBN is a more efficient tool for fault detection and root cause analysis.
-diagnosis provide. In [54], *Wu* writes about the approaches for risk assessment of hazardous events and their consequences on oil rigs, for which models DBN's are used. *Yang Pang* in [57] describes the possibility of detecting wear phenomena within mechanical systems where he uses dynamic Bayes networks. It is possible to investigate complex systems and relationships with time histories. As an example and test, the implementation is run through on a *lock mechanism*. DBN are used for fault diagnosis to replace mathematical models and signal patterns, which reach their limits in complex systems. *Arroyo-Figueroa* describes in [18] another extension of Bayes networks, he calls it *Temporal Bayesian Network of Events* TBNE. A node represents an event or state change of a variable and an edge represents a causal-temporal relation. Temporal intervals can vary in size and number for each node, making them different from classical DBN's. *Arroyo-Figueroa* uses an example from medicine as an exemplary comparison. In [49], *Dienst* addresses the utility of using DBN's in product development to draw conclusions from the product life cycle. As an example a rotation spindle/rotary spindle is examined. Also the necessary steps to develop a DBN are shown there. In [36] the influence of cutting forces on tool path deviations is investigated. Here, the whole process is divided into individual sub-instances and several digital twins. In [5], *Brandherm* has designed a way to convert DBN into nth-order polynomials, thus saving memory and computation time.

work.

The term *dynamic* is emblematic of modeling a dynamic system with changing distributions of variables over time and non-changing network [49, 28].

The literature review shows that dynamic Bayesian networks are used in many areas for root cause analysis and diagnosis or decision making. It was also shown that it is possible to achieve more accurate predictions and results. The advantage that a DBN can take the temporal component into account and model it, is offset by the disadvantage that the Bayesian network grows continuously with the runtime and thus the longer runtime when executing algorithms. In this thesis, however, the temporal dependency is neglected and we continue to work with static Bayes nets. A conversion to dynamic Bayes networks, which would be advantageous for the exemplarily applied process of blow molding, since it concerns thermal processes and wear phenomena, for which the consideration of the temporal course would achieve more exact and/or more correct results and predictions, is renounced due to the extent, which does not permit it. A "live connection" to the machine is established and thus the current machine/process state is taken into account in the decisions/predictions. A possible concept for the development of a DBN is explained in the outlook on the basis of the investigation results and analyses.

4. System and software status at the start of the work

The present work is created building on an existing system and software status. The status after the end of this phase and at the beginning of the work is explained in this section.

The assistance system is based on a static Bayesian network and performs predictions based on an offline database. The required program codes for the software are created with the programming language [R1]. For visualization and interaction/communication between human and machine the **R** compatible software *Shiny* [2] is used. This offers the possibility to generate outputs via a dashboard and to receive inputs from the operator.
To create such an "app", both a server side and a graphical user interface (GUI) were programmed. The server side contains functions and executable **R code**, whereas the GUI is responsible for receiving inputs and outputting results/calculations/predictions. The GUI provides the direct interface to the operator [24, Chapter 4]. Figure 4.1 shows the architecture in a graphically comprehensible way.

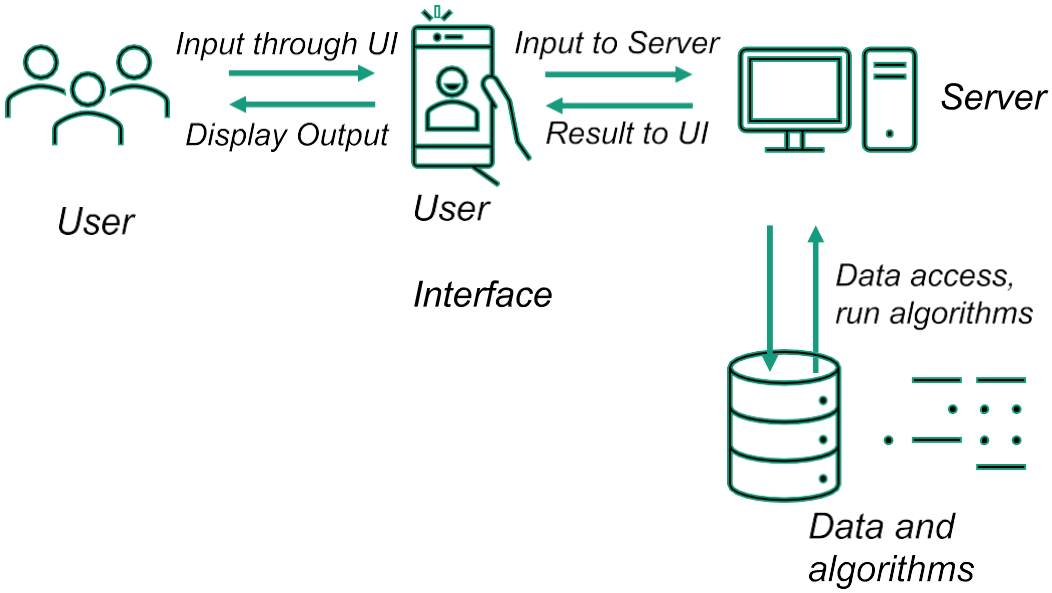


Figure 4.1: Architecture and function *Shiny*

A conceptual design of the BN structure for the production and quality control phase

[1]Homepage **R** https://www.r-project.org/, Homepage **R-Studio** https://www.rstudio.com/
[2]Homepage *Shiny* from **R-Studio** https://shiny.rstudio.com/

of a preform has been created. Based on expert knowledge, possible defects and associated influencing factors (nodes) and dependencies (edges) are defined. The structure must be finalized and evaluated as complete before phase two, the structure for the generation and quality control of the finished end product, can begin. Figure 4.2 shows a section of the current state of the Bayes net structure for phase one (*control preform*). The lowest level contains possible defects in this phase defined as nodes. The upper nodes represent possible influencing factors/causes.

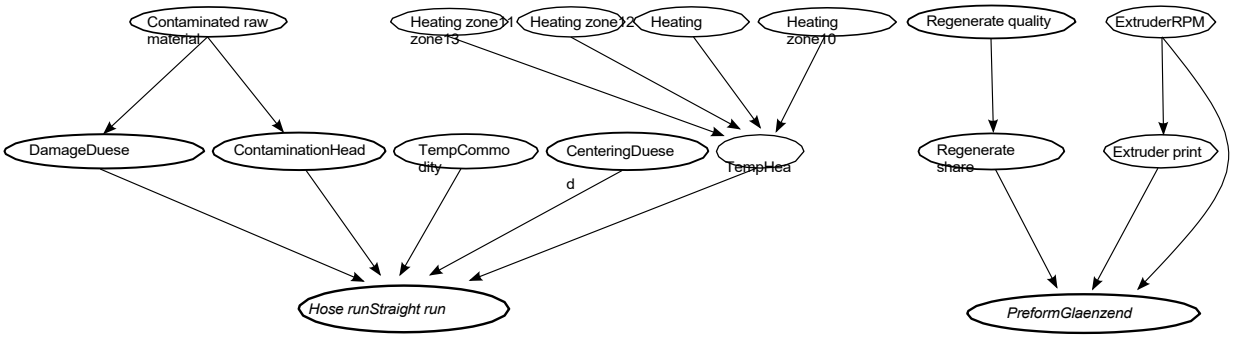


Figure 4.2: Bayes net structure section of the *preform* phase for the defects "tube run straight" and "preform glossy" at the beginning of the work

The conditional probability tables of the individual nodes (CPT) are calculated by so-called "Parameter Learning"[46] filled with values. This requires data that originate from the process history and describe both "normal" and "abnormal" states of the process/product. For the implementation the **R package** "bnlearn" [35] is used. This package offers ready-to-use library functions for working with Bayes nets and provides a function for the "Parameter Learning". On the software side, the possibility of "learning" is implemented, but data for the complete learning of the entire structure is missing. Within the work, in addition to the possible connection to the machine, possibilities are discussed later on with which it is possible to generate realistic synthetic data in order to enable complete learning of the network despite the lack of real process data.

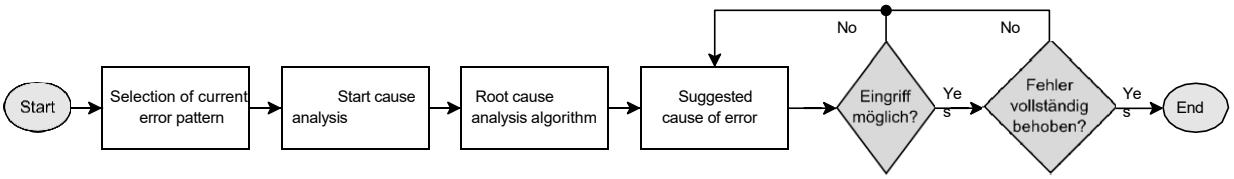


Figure 4.3: Flowchart process of root cause analysis and troubleshooting, status handover/start of work

The process for error detection and cause analysis as well as elimination is implemented as follows (see Figure 4.3): An operator selects a currently detected error/error image/process disturbance (error is not automatically detected by the assistance system) and starts the root cause analysis by means of a button. Now, on the basis of a specially designed algorithm, the probabilities for all possible causes are calculated by means of

Inference calculated. The system gives the operator as a recommendation the cause with the highest probability of minimizing the occurrence probability of the error. Mathematically, the calculation looks like this: On the one hand, the probability that an error occurs (*true*) is calculated, given the condition that a cause ($cause_1$) is also *true* (*true),* and on the other hand, given the condition that a cause ($cause_1$) is not true (*false*). Subsequently, the difference of both probabilities is formed and thus it is shown how much the occurrence probability of the error can be changed by a change of state of the cause.

$$P_{1a}\,(error|cause_1) = P\,(error = true|cause_1 = true)\; P_{1b}\,(error|cause_1) = P\,(\text{error} = true|cause_1 = false)\; P_1\,(error|cause_1) = P_{1a} - P_{1b}$$

Via an input, the system requires the feedback whether an intervention as suggested was possible or the error was completely eliminated. Otherwise, the setter receives next possible error causes until the error pattern could be completely eliminated. Figure 4.4 shows a section of the current GUI for error detection and root cause analysis. An operator has selected *Scratches* as the present error pattern and receives a *damaged nozzle* as a suggestion/recommendation for the most likely cause.

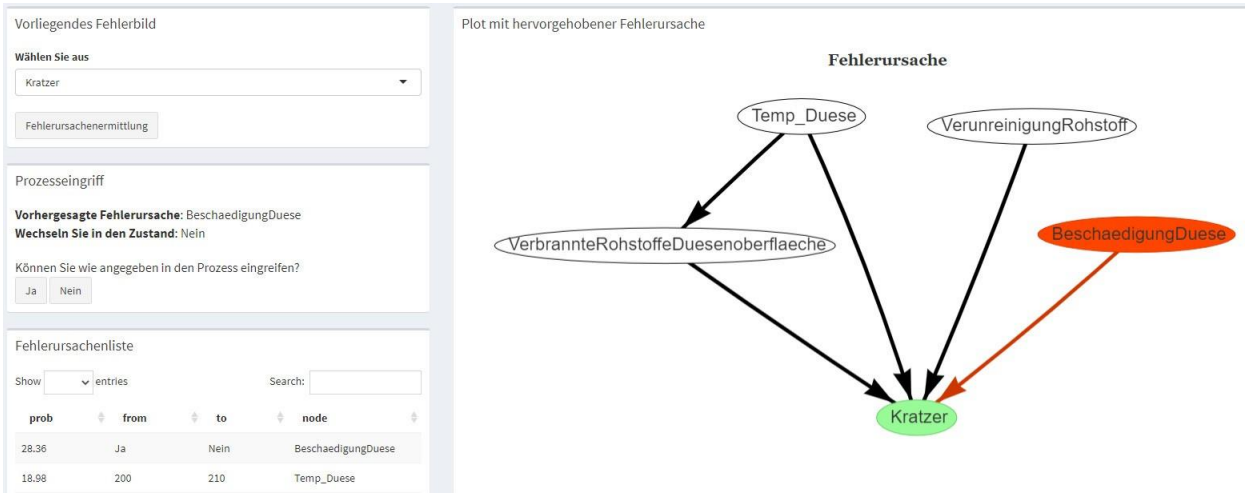


Figure 4.4: Excerpt from error detection and cause analysis GUI using the example of an error image *Scratch*, Stand Start Work

In the current state, there is no "live connection" to the machine and the current machine state cannot be detected automatically. Due to the missing connection, it is possible that the assistance system suggests a possible cause for an error, which, however, does not correspond to the current machine status, since this is already in the suggested status change of the influencing factor. As a possible example: In the case of a *scratch fault, the assistance* system suggests increasing the *nozzle temperature* to 190° C, but the *nozzle temperature is* located at

already in this temperature range. Since there is no status query and the cause analysis is carried out on the basis of historical process data, such cases cannot be ruled out.

In addition to the "faulty" or "not real" root cause analysis results, another problem occurs due to the non-existent connection to the machine. It is not possible to generate and save process/machine data automatically from the assistance system. The collection of data is thus made more difficult. In the course of this work, possible solutions and implementations will be shown with which it is possible to establish a connection to the machine and to collect data automatically.

The current GUI (see Figure 4.5) provides options for obtaining an overview of the entire network structure as well as for extracting individual fault patterns and viewing them separately. It makes it possible to visualize and output the possible causes. Furthermore, there is a tab for error detection and cause analysis (*process intervention*) as well as a tab for the "*Parameter Learning*" is available. For "Parameter Learning", data is uploaded via the interface in the form of a CSV file. It is then possible to determine new conditional probabilities via a button using algorithms from library functions of the **R package** "bnlearn" and to set the Bayes net to a current state.

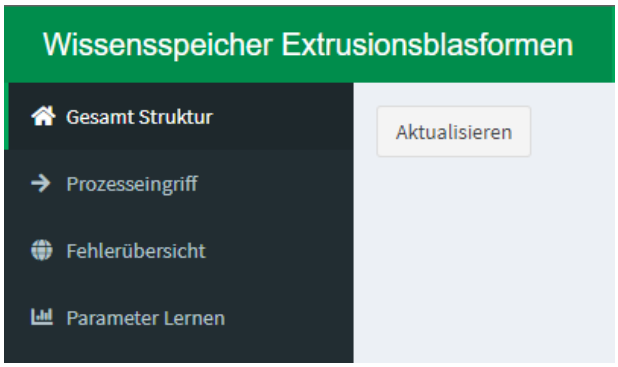


Figure 4.5: Section of the user interface, "Tab overview", status at the beginning of the work

The status described to date is based on literature research as well as on theses and assumptions made by the authors themselves. These have neither been validated nor tested. Therefore, only a reduced comparison with the assistance system at the beginning is made in the later analysis and evaluation section. Based on the present software and system status, the assistance system will be further developed in the following. After showing a general solution approach, the exemplary implementation/realization is shown using the example of the blow molding process and the production of 3D plastic hollow bodies. The currently implemented algorithm for determining the causes is also analyzed and, if necessary, adapted.

5. General solution approach

In this chapter, a generally applicable concept is designed with which it is possible to develop an "inter- active assistance system" that provides support to an operator via a user interface during the setup process of a production plant. Feedback from the operator as well as the condition of the machine are taken into account in the cause analysis. Later on, the implementation will be explained and described using the blow molding process as an example.

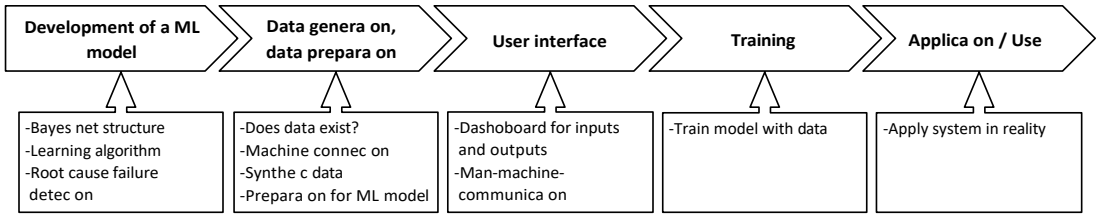


Figure 5.1: Development flow of an interactive system based on ML (Bayesian network)

Figure 5.1 above shows the individual components of the development of an interactive assistance system, starting with the design of the model up to the application and use in reality and in the machine.

5.1. Structure Bayes net

Section 2.2.1 shows the various possibilities for building the structure of a Bayesian network. With the assumption that the assistance system to be developed is the first application that monitors the process using machine learning methods, there will be a small to non-existent or usable database. However, if a structure is to be developed and defined from data, a large and wide data set is required. For this reason, nodes and edges are defined using expert knowledge and impressions from collaborators. In [34, Chapter 1], *Scutari* describes the ways to build a structure for a Bayesian network in **R.** In [34, Chapter 1.3] the variant using formal equation (*modelstring*) is shown.

```
1 # create DAG via modelstring
2 dag <- model2 network ("[ cause _ 1][ cause _ 2][ cause _ 3| cause _ 2][ error|cause _ 1:
      cause _ 2: cause _ 3]")
3 class( dag)
4 [1] " bn"
```

Source code 5.1: Code example in **R** to create a DAG by means of formal equation

In a generally valid concept, a process is analyzed with experts and the causes of possible errors are defined. In addition, dependencies among each other (between nodes) are defined, e.g. an increased *nozzle temperature* influences not only the occurrence of the error *scratch but* also the probability that *burnt raw material* accumulates *on the nozzle surface*. In order to reduce effort and improve the overview, a designed routine/algorithm is used to create the overall structure in **R.** Thus, the formal equation (see e.g. source code 5.1) is established not manually, but with the help of created functions. In Figure 5.2 a flowchart can be seen, which shows the process of the overall structure creation. A matrix defining dependencies among nodes is read in from an Excel list and then converted into a formal equation in **R.**

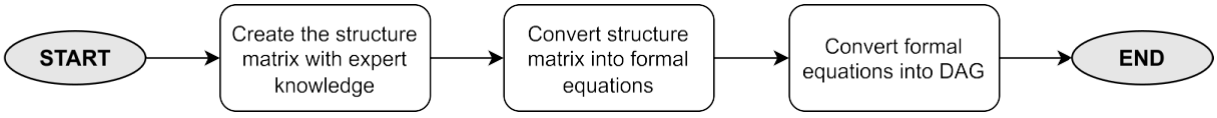


Figure 5.2: Flowchart creation of Bayes net structure

Table 5.1 shows an excerpt from the Excel spreadsheet with the structure matrix included. The mesh created in this way consists of the *cause nodes, which can be* found under "Mesh name", and the three fault nodes listed in this example (*scratch, melt fracture* and *streak formation*). If a cause node has a parent node, this is entered in the "Parent" column. If there is more than one parent, additional columns are used continuously to the right. Only one node may be contained per cell. The respective column below an error is used to assign which causes apply to that error. An *x* means that the cause node has a *direct* edge to the fault. An *s*, on the other hand, indicates that the cause node is assigned to this error, but does not have a *direct* edge, but has a dependency via other nodes.

Table 5.1: Excerpt from the matrix for creating the Bayes net structure

Network name	Parents k. 1	Parents' k. 2	Cat	Melt fracture	Schlierenbil.
VerbraRoh	TempDuese		x	x	x
VerunRoh			x		s
TempDuese	Heating zone4		x	x	x
BeschDuese	VerunRoh		x		x
MatMoist				x	
Regenerate	Regeneratqua			x	x
Output dr				x	
TempExterior	Heating zone12	Heating zone13		x	
TempHead	Heating zone10	Heating zone11			x

By means of the specially written function *excel2ms(excellist, netname, firstfault)* the matrix,

which is previously edited with *customizeExcellist(excelliste)*, into formal equations and saved as a text file in the working directory. *createDAG(msfile)* allows the previously created text file to be converted into a fully-formed Bayes net structure. Source code 5.2 shows the implementation used to generate a full Bayes net structure in **R** from an Excel list with a structure matrix included.

```
1 # read instructure matrix
2 excellist <- readxl :: read _xlsx (" parameter list .xlsm ", sheet=" error preform ")
3 # prepare excel list
4 excellist <- customize Excellist ( excelliste = excellist)
5 # build formal equations and save as . txt
6 excel2 ms( excellist = excellist , netname = " Total", firstfault = " Scratch")
7 # read intoformal equations
8 msTotal <- read . table (" MSTotal. txt")
9 # create DAG
10 dag <- create DAG ( msfile = msTotal)
11
12 # variable type
13 class( dag)
14 [1] " bn"
```

Source code 5.2: Code example in **R**; creating the complete Bayes net structure as a concept

Figure 5.3 below shows the Bayes net structure created by source code 5.2. This figure serves to show the complexity of such a structure. In source code 5.2 line 14 it can be seen that the variable *dag is* one of type "bn". To obtain a complete Bayes net, it requires learning the parameters (result variable type "bn.fit")[34, Chapter 1.4]. Learning requires data sets from the process. The acquisition of these is explained in the following section.

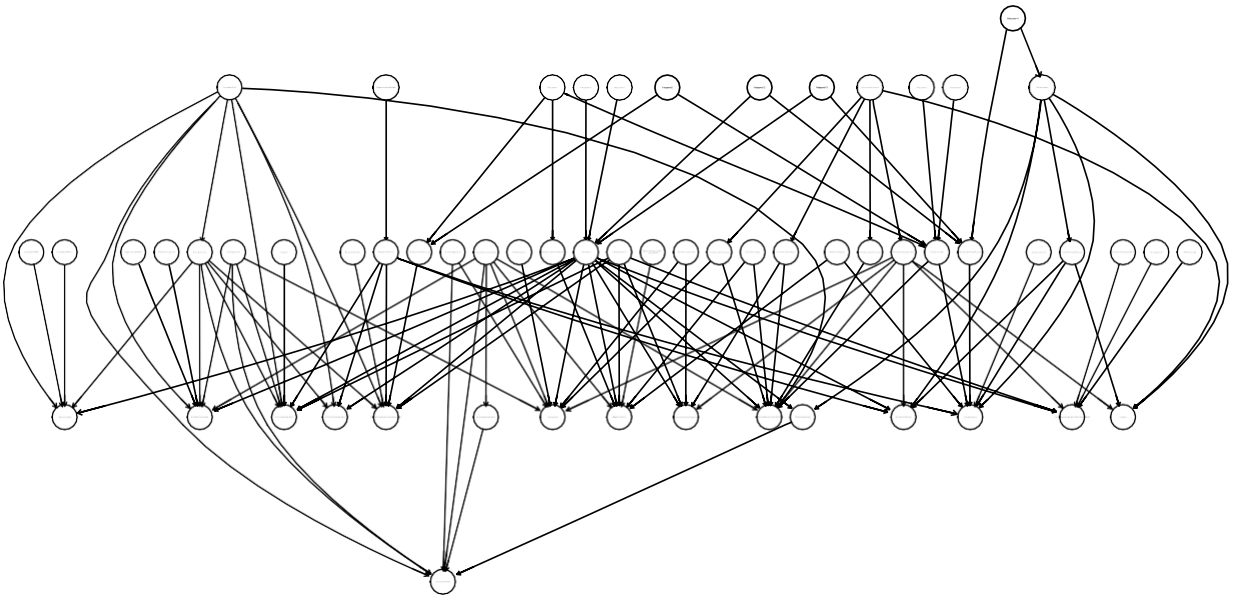


Figure 5.3: Conceptually created Bayes net structure according to source code 5.2, nodes irrelevant

5.2. Connection to the machine

To collect process data and query the process/machine status, it is necessary to be able to access data from the programmable logic controller (PLC). The controllers installed in the machines are Siemens products. For this reason, solutions are considered which allow access to the product family of Siemens controllers. Access options for PLCs from Beckhoff, Allen Bradley or ABB are not examined further in this work.

5.2.1. Snap 7 as access option to process data

Snap7 [1] is an *open source*[2] solution to provide an interface for accessing process data. Communication between the automation device and the PC can thus be established. The controller families S7-1200 and S7-1500 can be used almost completely, for older controllers it is necessary to check the support in the data sheet. Snap7 is developed using *C++* and is also available in other languages through various wrappers.[14]
By using *client*, *server* and *partner* objects, a PC is optimally integrated into the PLC automation chain. It is also platform-independent and is supported by common operating systems such as Windows, Linux, BSD and Apple OSX. At the same time it is completely
scalable and offers the possibility to be used on a *Raspberry PI* [3]. To open
Snap7 uses the *Ethernet S7 protocol communication*[4] *as* an interface. A *client* object can only make requests, a *server object can* only respond, whereas a *partner* object can make both requests and responses.[14]

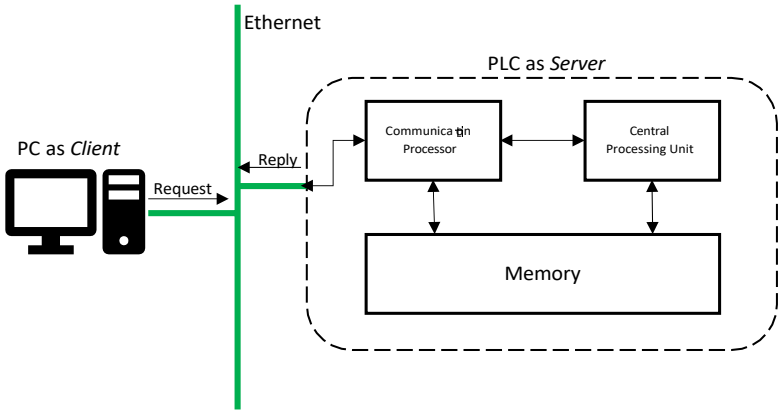


Figure 5.4: *Snap7* architecture of a *client* object

[1]Homepage *Snap7* : *http://snap7.sourceforge.* net/
[2]*Open Source*: Originally, the term comes from Open Source Software. The code is publicly available to everyone and can be used and changed. Open source is mostly a community product that comes from a common development. Problems and possible ways are discussed in a broad group and is created. *https://www.* redhat. com/en/topics/open-source/what-is-open-source (accessed 21.01.22)
[3]Homepage *raspberry Pi* : *https://www.* raspberrypi. com/
[4]*S7 Protocol* : *https://support.* industry. siemens. com/cs/document/26483647/which-features-and-features-offer-the-s7-protocol-?dti=0&lc=en-WW (accessed 21.01.22)

The architecture can be represented as follows (see Figure 5.4): Both the *PC* and the *controller* are connected via Ethernet in the same network. The *client* makes a request and the *server* answers it. The use of Snap7 via Python5 is possible and realizable with the package *python-snap7*. This is a Python wrapper for the open source Snap7 library. Thus, it is possible to create an interface to Siemens controllers via Python [19]. In [2] *Truon-Duc-Duy* investigates the possibility of commissioning and monitoring a thermal process. Here, a Siemens PLC S7-1200 is used and the *python-snap7 library is* used as an interface to the remote PC.
Figure 5.5 shows the approach for implementing an interface to the machine. The aim is to be able to access relevant machine and process data with the implemented communication interface to the controller. Since only data is to be read from "outside", a *client-server* structure is set up. In this case, the controller is the *server* and the PC is the *client*. The *client* makes requests and the *server* answers them. Incidental process data are stored and saved in data blocks (DB's) via *inputs of the controller*. The PLC and the *client* PC are located in the same network. If the *client makes* a *request*, the *server responds and the* data from the requested data block addresses are transferred to the *client* via Ethernet using the S7 protocol.
Detailed documentation and code samples are available at [19].

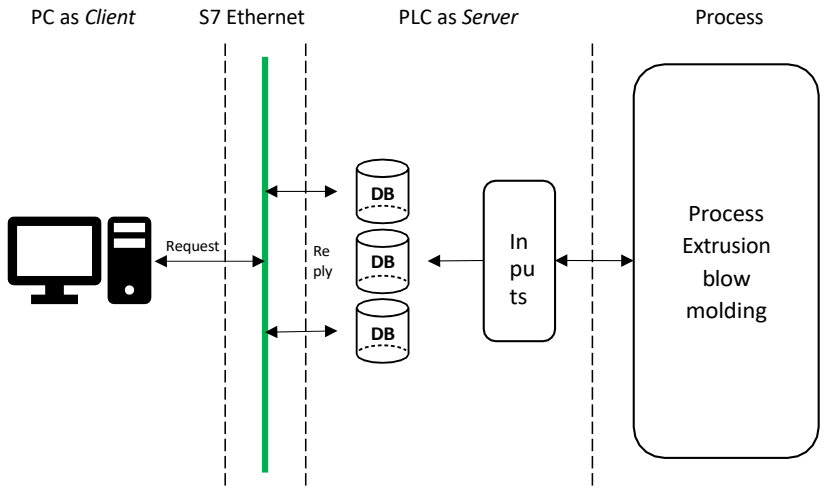


Figure 5.5: Machine interface approach using *python-snap7*

5.2.2. Comparison of further access options to process data

In addition to the above-mentioned possibility of using *Snap7* to establish an interface for communication with a Siemens controller, there are a number of other *open source* solutions. Among others, *Libnodave6* should be mentioned, the free library supports the Siemens S7-200/300/400 controllers. *ISO on TCP* [58] and *S7Online7* are available as communication protocols.

[5] *Python* homepage: https://www.python.org/ (accessed 21.01.22)
[6] Hompage *Libnodave*: Documentation and code samples http://libnodave.sourceforge.net/ (accessed 21.01.22)
[7] S7Online interface: https://www.ipcomm.de/protocol/S7ISOTCP/en/sheet.html (accessed 21.01.22)

are available. In addition to Ethernet, the serial interfaces *MPI* and *PPI* are also supported there. In [52], *Moecker* examines the possibilities of an application for an Android system for interaction with a Siemens controller. Furthermore, *IP-S7-LINK* [8] and *SAPI* [9] should be mentioned. The possibilities of accessing controllers and process data via *OPC UA*[10] *Server* or *SIMATIC Web Server* originate from Siemens. If the infrastructure is not available, these two options require a great deal of effort to implement, since changes have to be made in the control programs and configurations. Among other things, this requires the activation of a variable for the **OPC!** -interface. In [56], possibilities of data acquisition on "older" machines are examined. The topic of OPC UA is also discussed there. The time required to read out a large number of parameters is mentioned there as a disadvantage. For further information, documentation and code examples, please refer to [25, 21, Chapter 21].

5.2.3. Implementation in code

An implementation of the machine connection is shown below. Figure 5.6 shows the flow chart that illustrates the process. At the beginning, a connection to the controller is established. Subsequently, requested data is read with a corresponding library function of the transferred address (*request*). Since an entire data block can be requested, further library functions are used to select individual required data from this block. The connection is automatically terminated by the end of the program [19, Chapter: Client].

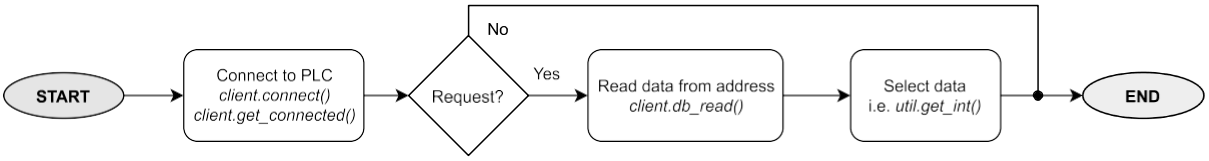


Figure 5.6: Flowchart: Approach of machine connection and data acquisition by means of *python-snap7*

Source code 5.3 shows the exemplary connection to a machine as well as the request of the temperature.

Lines 5 and 6 are required to establish the connection with the controller. The first parameter passes the IP address of the PLC, followed by the *rack* and *slot* number. Finally, the *port* number is required, which is initialized with 102 by default. Line 9 checks the successful connection establishment and outputs it. In line 12, data from the controller

[8] Homepage *IP-S7-LINK* : https://www.traeger.de/software-driver-simatic-s7/pc-and-s7-with-tcp-ip-connect-ip-s7-link.html?gclid=CjwKCAiA0KmPBhBqEiwAJqKK416lxAjufKL3iVb0VzYwcaWpi_opO2A5lEpYySOztbuebazdY1gs1RoCo6AQAvD_BwE (accessed 21.01.22)

[9] Documentation on *SAPI* : https://cache.industry.siemens.com/dl/files/203/13649203/att_38729/v1/mn_s7api_e.pdf (accessed 01/21/22).

[10] Homepage Documentation to *OPC UA*: https://new.siemens.com/en/products/automatisierung/industrial-communication/opc-ua.html?gclid=CjwKCAiA0KmPBhBqEiwAJqKK4-VpLFBjVsJDaS6-xtkQPZ8CLoWlkFfH_LyrjyteKyRRKBelad_R8hoCiS4QAvD_BwE (accessed 21.01.22)

read. With *db_read()* the following 8 bytes are read starting from address 213. Possible parameters are *db_number, start_byte* and *size*. The return format is the type *bytearray*. To select the received data separately, the class *Util* provides some functions. In line 15, for example, the temperature is selected, which is located at address *DB213.DBW* 2. A data block can thus be requested by a controller, the corresponding variable selected and stored for further use.

```
1 import snap7
2 from snap7 import util
3
4 # connect to plc
5 client = snap7.client.Client()
6 client.connect("192.168.0.31",0,0,102)
7
8 # check connection
9 print('Connected:',client.get_connected())
10
11 # get data from data block
12 data = client.db_read(213,0,8)
13
14 # DB213.DBW2
15 temp = util.get_int(data,2)
16 print('Setpoint temperature HZ2 \n!',temp)
```

Source code 5.3: Connection to the machine, code example *python-snap7* Connection setup and request temperature

5.3. Data generation for learning and training a Bayesian network

5.3.1. Data preparation and preparation of data

In most cases, it is not possible to use data collected from a machine or a process for training a Bayesian network without further ado [47]. *Paluv* describes the six steps of the ML process chain, including the path from "raw data" to data that can be used for the Bayes net. *Zheng* describes in [4] data as individual pieces which represent an aspect of reality. The totality gives a picture of the whole process. However, the picture is chaotic, since it involves hundreds/thousands of individual pieces and mostly parts are missing.
The nodes located within a Bayes network can assume different states,
For example, the *Contaminated raw material* node can be in the "Yes" or "No" state. The goal in preparing the data is to recognize characteristics, keep the number of possible states within a node as small as possible, and thereby reduce the amount of data required. By minimizing the states per node, the required CPT's are also minimized in size. For numeric data, the option of classifying or dividing into intervals is a good choice. For binary or arbitrary variables, on the other hand, logical functions lend themselves [4, Chapter 2].

In general, machine and process data have two different types of data (*numeric* and *binary*). Binary data does not require discretization and processing. Numerical data are "continuous" variables that require discretization. There, discretization is possible for temperatures, times, velocities, etc. in three classes. However, no intervals of the same size are chosen, but different ones, e.g. 10%/80%/10%. This ensures that each interval receives a sufficiently large number of data, but also takes into account effects of smaller or larger measurements [36]. For example, the classes are *low*, *medium*, *high*. Low and high are used to define "negative" states and medium to define "positive" states.

Table 5.2: Extract from classification for data set *TempDuese*

TempDuese	187	203	194	199	192	182	193	199
TempDuese_class	medium	high	medium	medium	medium	low	medium	medium

Table 5.2 shows an excerpt from an example classification of a *nozzle temperature* data set. The top row contains the original data and the bottom row after the classification into three classes. Source code B.1 shows the function *classifyInto3()*, which classifies a data set with numerical data into the three intervals.

5.3.2. Generate synthetic data

The further development of AI, the widespread use of machine learning and its application in many applications has led to an immense increase in the demand for data [55]. Data (process data, production data, measurement data, etc.) have been recorded and stored in recent years, but the necessary structure as well as possible required assignments to error states were missing and thus the quality of the data available today is often insufficient. The quality of the data significantly determines the delivered results of an AI application [16]. The newly discussed issue is where to obtain data from and how to solve the problem of lack of data. A possible solution besides the tedious and time-consuming recording of real process data seems to be the generation of "synthetic data". Figure 5.7 shows the structure that is required for machine learning. In the first place, the topic "data" can be found. This represents the significant starting point of an ML model.

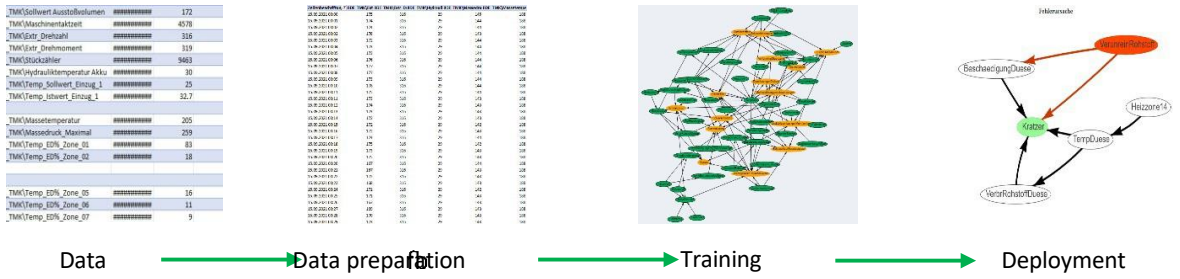


Figure 5.7: Structure of a machine learning process (adapted from [55, Chapter 1])

Developers are confronted with two questions: What data and sources are available and can they be accessed? What additional data is needed? This section deals with the creation and generation of so-called *synthetic data*. These represent a kind of "artificial twin", they exhibit the same patterns, structures and behavior [39]. Optimally generated synthetic data identically reflect a production process. The generation and creation of synthetic data can be divided into three categories.

- Random data generation; no consideration of dependencies or distributions
- Rule-based data generation; partial consideration of dependencies and distributions
- Data generated by AI methods; consideration of dependencies and distributions

Rule-based data contain a statistical control technique, which has the task of ensuring that synthetic data acquire the same statistical properties as the original data and the quantita- tive estimate of a variable of a given state [17]. Many of the available data generators are general purpose or developed for areas such as biological data, geospatial data, mathematics and finance or computer vision applications. These are not used to generate production or machine data. However, the concept is identical. Figure 5.8 shows the structure. A database as well as additional information is required from which the process is modeled. Subsequently, the generation of the synthetic data can take place from a data model, which is simultaneously the output of the generator.

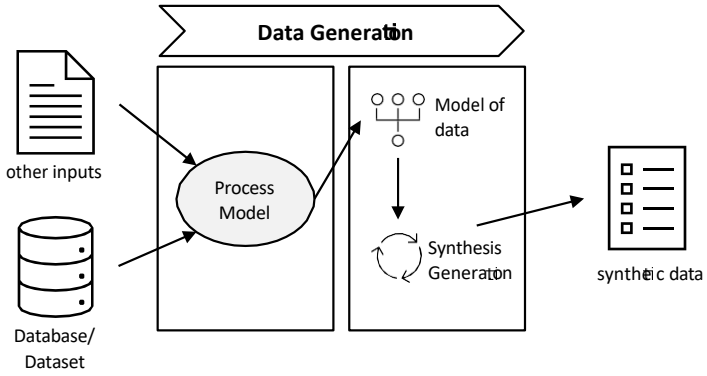


Figure 5.8: Model of a data generator (based on [17])

Fernandes describes in [17] a data generator which has been developed especially for industry and production. This offers the possibility to select the data individually over six levels and to adapt it to the own application. Despite the possibility of adaptation, this generator has not been developed for every case. Thus, it is not possible to use it in this work.
The simplest way of generating data is the variant of *random data*. Here, neither dependencies between individual variables nor distributions are taken into account. The information required is the *name of* the individual variables/nodes and the possible *values/states that* they can assume. Figure 5.9 shows the flowchart of a generation process.

of *random data*. In **R, the** nodes and node states are extracted from the Bayes net structure and then passed to a function. This function generates the synthetic data within two loops by means of random distributions.

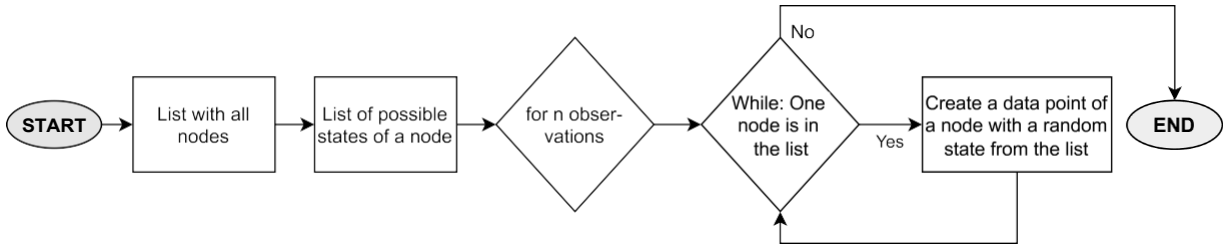


Figure 5.9.: Flowchart for generating random *synthetic data*

Table 5.3 shows the first ten rows from the generation of synthetic data for the *scratch* error. Dependencies between variables are not considered and the distributions are chosen randomly. With this algorithm it is possible to generate data for individual faults (*subnetworks*) as well as for the entire network. The data generated in this way can be used for learning and training a Bayesian network without further processing.

Table 5.3.: Extract (10 out of 1,000,000) from the generation of random synthetic data for the defect *Scratches*

BeschaeDuese	Heating zone14	Scratch	TempDuese	VerbRohDuese	Contaminated pipe
Yes	*medium*	*No*	*iO*	*No*	*No*
No	*low*	*No*	*iO*	*No*	*No*
Yes	*medium*	*No*	*iO*	*Yes*	*No*
No	*medium*	*No*	*iO*	*No*	*No*
No	*medium*	*Yes*	*iO*	*No*	*No*
Yes	*medium*	*No*	*iO*	*No*	*No*
Yes	*medium*	*No*	*niO*	*No*	*Yes*
No	*medium*	*No*	*niO*	*Yes*	*No*
No	*high*	*No*	*iO*	*No*	*No*
No	*high*	*No*	*iO*	*No*	*No*

Rule-based synthetic data are many times more complex to generate than *random data,* but they also have a stronger relationship to real process *data*. In [15] *Oehm* describes the possibility to generate synthetic data from a trained Bayes net. This variant offers the option to generate pure *rule-based* data as well as data with *AI* methods. If a Bayesian network is used to generate the data and no further adjustments are made, the data is *rule-based*. However, if the generated data is used for further learning/training with the inclusion of real process data, it is data created with *AI* methods. The creation of data from AI methods is not considered in more depth, as the effort required to generate a real process model for this work is considered to be

is considered too large. However, these allow to generate the most realistic data.
The possibility described in [15] is the function *rbn()*[11] from the well-known **R** package "bnlearn". It takes as parameters a learned Bayes net (structure and filled CPT's) and the number of observations to be generated. The structure determines the order in which the data of the individual nodes are generated. Data of the root nodes are generated first, followed by the children of them. This step is iterative until data for each node is generated (see Figure 5.10). The conditional probability tables (CPT) are required to maintain the quantitative estimation [35]. The following figure shows in parentheses the order of the processed nodes and the data generated by them.

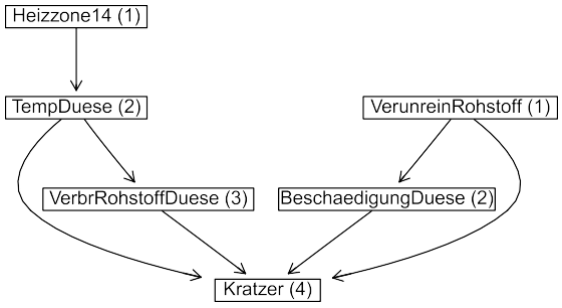


Figure 5.10.: Sequence of simulated data using *rbn()*, node *Kratzer* is the end node

The function *rbn()* requires a learned Bayes net as described above. If unlearned networks are available, it is possible to manually assign values to the conditional probability tables using "expert knowledge". This is feasible for a single subnet (see Figure 5.10 for individual faults), but in a large Bayes net the effort is too great and the knowledge is not detailed enough to be able to fill in a CPT in a differentiated manner.

Table 5.4: Extract of synthetic data generated with the function *rbn()* for the error *scratch*

BeschaeDuese	Heating zone14	Scratch	TempDuese	VerbRohDuese	Contaminated pipe
No	*medium*	*No*	*iO*	*No*	*Yes*
Yes	*high*	*No*	*iO*	*No*	*No*
Yes	*medium*	*No*	*iO*	*Yes*	*No*
No	*medium*	*No*	*iO*	*No*	*No*
No	*high*	*No*	*iO*	*No*	*No*
Yes	*medium*	*No*	*iO*	*No*	*No*
No	*high*	*No*	*iO*	*No*	*No*
No	*medium*	*No*	*iO*	*No*	*No*
No	*medium*	*No*	*iO*	*Yes*	*No*

Table 5.4 shows the excerpt of a data set of synthetic data, which was generated by means of the function

[11] *rbn(x, n = 1):* Simulation of samples from a given Bayesian network

rbn() have been simulated. Since the function is passed a learned Bayesian network, dependencies and distributions are taken into account when creating the data.

5.4. Parameter learning and Bayes net training

In addition to creating the Bayes net structure (qualitative part), a complete net needs conditional probability tables (quantitative part) filled with values. These represent the distributions of the nodes in reality. For learning and training, data sets are required that contain the nodes that represent a dependency on the node to be learned and, if possible, the entire spectrum of combinations (different node states). Here, a record of a current state is called an observation. However, a large number of observations is not synonymous with a high quality of data.
Various algorithms are available that can be used to determine the parameters from a data set. Since in this work the programming language **R is used** with the package "bnlearn" is used, the focus is on those used there. A variant is the determination of empirical frequencies.

$$P(TempDuese = iO | Heating\ zone14 = high) = \frac{P(TempDuese = iO, heating\ zone14 = high)}{P(heating\ zone14 = high)}$$

$$= \frac{\text{Number of observations } TempDuese = iO \text{ and } heating\ zone14 = high}{\text{Number of observations } heating\ zone14 = high}$$

[34]

Here, quantitative frequencies are "counted" and thus parameters are determined. The algorithm with the most frequent application is the *maximum likelihood estimation* MLE, which uses the function *bn.fit()* from "bnlearn" as a standard method. The central idea is to choose parameters that are most likely to fit the observed data. *Likelihood* is a synonym for joint probability. *Maximization* aims at maximizing the result of the likelihood function [10].

Record 5.4.1
Likelihood function:

$$L(\theta) = \prod_{i=1}^{n} f(X_i | \theta)$$

θ : parameter to be estimated
X_i : Data
n : Number of data

[10]

As a second option, *bn.fit()* provides the *Bayes Parameter Estimation*(BPE) method. This serves

also the estimation of probability densities of unknown parameters [23].

Record 5.4.2
Bayesian parameter estimation:

$$P\Theta|X(\theta|D).$$

Θ : parameters to be estimated

X : Date

D : Record

A major difference between the two methods is that in MLE θ is not a random variable, but in BPE it is. For more information and mathematical explanations, please refer to [23] referenced. The application of *bn.fit() can be* seen in source code 5.4. In the first lines the structure for the subnet scratch is created. In line 7, the synthetic data randomly created from section 5.3.2 (Table 5.3) is loaded. Finally, in line 9 the parameter determination by means of *bn.fit() can be* seen. On the one hand the structure (DAG) as well as data and on the other hand the algorithm to be used (in this case MLE) are passed. The difference between MLE and BPE within the function *bn.fit()* is the handling of missing data. If a data set is insufficient, MLE assigns *NA* to parameters that cannot be determined. This is different with BPE, where uniform distributions are assumed in this case [35]. With two possible states, this would be a 50/50 distribution.

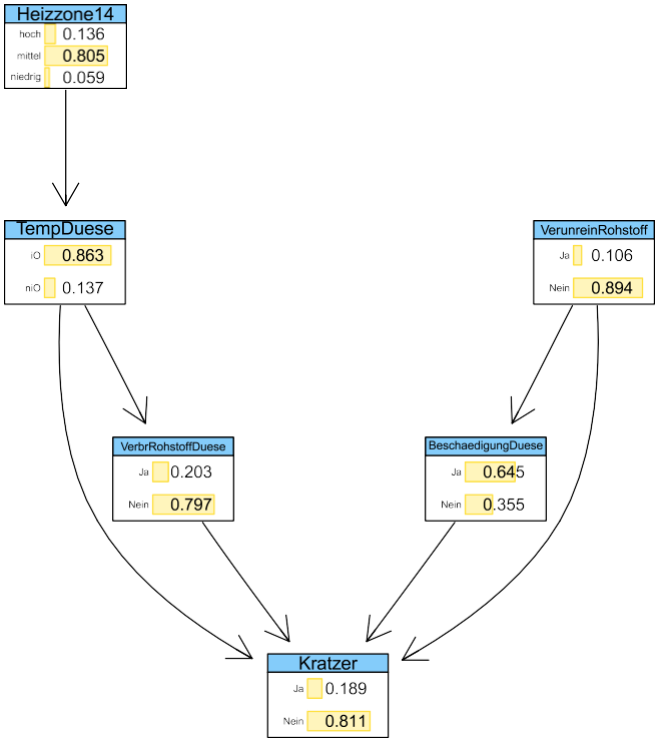


Figure 5.11.: Trained Bayesian network *scraper* with random synthetic data from Table 5.3.

```
# create a DAG
paraliste <- readxl :: read_xlsx (" parameter list .xlsm ", sheet=" error preform ")
paraliste <- customize Excellist ( excelliste = paraliste )
modelstring <- create MStrings ( paraliste ," scratch")
dag <- model2 network ( modelstring )
# read in a data set
dataset <- read .csv2 (" test data / test data _edge _scratch. csv ")
# learn parameters
bn <- bn. fit( dag , dataset , method = " mle")
```

Source code 5.4: Traninating a Bayes net structure using *bn.fit()*

The learned and now complete Bayes net can be seen in Figure 5.11. The distributions of the individual nodes are visible through the bar charts shown. It is possible to display the individual CPTs of a node separately. A Bayes net must be learned at least once before it can be used. However, since the environment and real data change over time, it is also necessary to adapt the Bayes net to new circumstances (retraining). This adaptation by new data sets is done manually, since an automatic retraining involves some risks. In the outlook, however, a concept is shown how automatic retraining could look like. In this context, possible risks are also discussed.

5.5. Algorithm root cause analysis

The most important question of a setter is what is the root cause of a fault that has occurred. This question is to be answered with a root cause analysis (RCA) algorithm. A first root cause analysis algorithm has already been implemented. This algorithm uses inference calculations to determine the cause that minimizes the probability of occurrence of the fault as much as possible (see chapter 4 incl. figure 4.3). The described and implemented algorithm calculates the probability from "top to bottom" (from parent node to child node). Hereby the *most influential* cause is determined. However, this may have a low probability of occurrence under certain circumstances. For this reason, a second algorithm is developed, which calculates the Bayes net from "bottom to top" (from child nodes to parent nodes) and thus determines the *most probable* cause. Developed algorithms are specially designed ones based on conditional probabilities. Figure 5.12 shows the flowchart for this algorithm. Possible causes are extracted for an input fault pattern. Subsequently, probabilities of occurrence for causes in different states are determined using equation 5.5.2.

$$P\,(event|evidence) \tag{5.5.1}$$

$$P\,(cause = state_{bad}\,|error = true) \tag{5.5.2}$$

It uses the approach of conditional probability: Calculate an event, given a condition. The event is the cause of a certain state and the condition is that the error occurs ("is true"). The list resulting from the loop is sorted in descending order. Thus, in the first place is the cause with the associated "bad" condition, which is most likely to occur with this error.

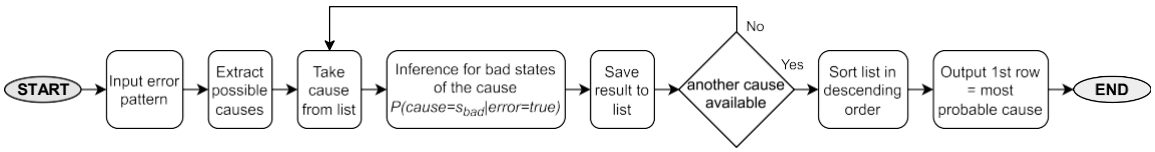


Figure 5.12.: Flow chart algorithm for determining the *most probable* cause for the occurrence of a fault

The dynamic modification mentioned in the task offers different possibilities. **Implementation option 1:** The cause of the fault is calculated by the algorithm for the *most influential* cause and then compared with the current machine or process state. Here, it is checked whether the predicted cause exists in reality. If this is the case, it can be eliminated by the operator, otherwise it is automatically sorted out and the next cause from the list is compared with the current state. This ensures that the probability of occurrence of the error is minimized by correcting the cause and the operator only receives suggestions that can be implemented in reality. Figure 5.13 shows the flow chart for the described implementation. The prerequisite for this implementation is that there is a possibility to automatically compare the suggested cause with the process or machine status.

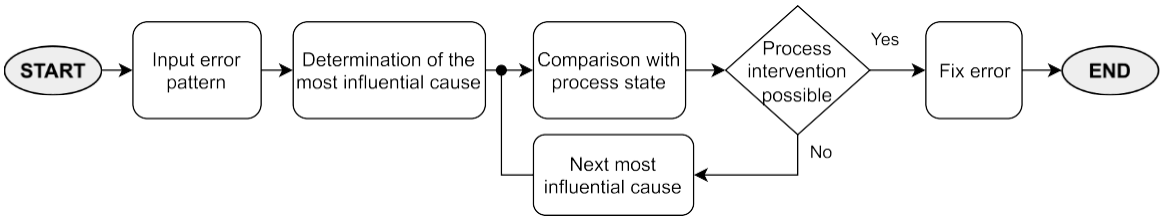


Figure 5.13.: Flow chart algorithm for determining the *most influential* cause for the occurrence of a fault

Implementation option 2: The cause of the error is calculated by the algorithm for the *most probable* cause. Subsequently, the operator/setter sends feedback to the system as to whether the error has been corrected or not. In case the error persists, a new cause of error is determined. This time, however, under the condition that the state of the previous cause (child node) is known and can therefore be taken into account in the inference calculation (see equation 5.5.3).

$$P\left(cause_2 = state_{bad} \mid error = true \;\&\; cause_1 = state_{known}\right) \tag{5.5.3}$$

The second cause is not extracted from the list, but a new calculation is performed under the new known circumstances (see Figure 5.14). With this conversion option, no automatic adjustment with the process or machine status is required. The feedback is done manually by the operator. In addition to dynamization of the assistance system, one can also speak of an iterative process.

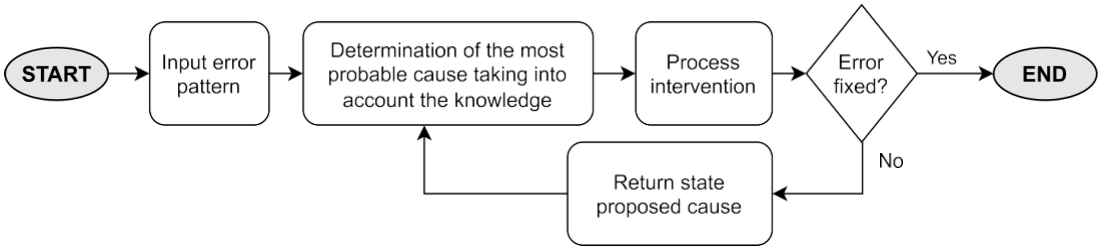


Figure 5.14.: Flowchart algorithm for determining the *most probable* cause for the occurrence of a fault, as an iterative process for dynamization

The choice of the algorithm used therefore depends on the underlying infrastructure of the process connection and the type of nodes contained in the Bayesian network. If the Bayesian network contains nodes whose data/states cannot be captured automatically, which means that no process synchronization takes place automatically, implementation option 1 is invalid or involves additional work for the operator.

5.6. User interface

A user interface enables interaction between the assistance system and the operator. The operator can make inputs and receives outputs in the form of graphics and text. The processing and use of Bayesian networks is done in **R**. *Shiny*, an **R** compatible software solution, offers the possibility to create dashboards. The state at the beginning of this thesis was built using Shiny (see chapter 4). The basic structure and framework are therefore in place. The interface has two main tasks.
Main task 1: Visualization of the process for determining the cause of the error. The setter enters a current defect image via the interface. The assistance system then displays a possible cause both graphically as a plot and textually. By means of inputs, the system is able to provide feedback on predictions. Figure 5.15 shows a possible concept of the "Process Intervention" tab.

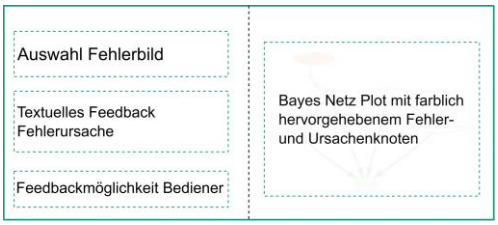


Figure 5.15.: Concept User Interface Tab for Failure Mode Detection Process

Main task 2: Enable the retraining of a Bayes net. The operator is given the possibility to upload new data sets and to adapt the Bayes net to new environmental conditions. For accurate root cause prediction, it is necessary to retrain the parameters with collected new data. Figure 5.16 shows a concept for a possible implementation of the tab.

Figure 5.16.: Concept User Interface Tab for Failure Mode Detection Process

Additional tabs are not mandatory for the use of the system and serve as extensions. Thus e.g. a total overview of the Bayes nets "preform" and "blow molded part" and a display of the subnets of a selected defect to indicate the possible causes of the defect.

6. Realisation approaches and implementation

The general solution approaches of an interactive assistance system for the support of an operator described in chapter 5 are exemplarily applied to the example "setup process of a blow molding machine".

6.1. Structure Structure Bayes net

The structure of the Bayesian networks is defined by expert knowledge due to the lack of available data. So-called "subnetworks" are created/viewed, which are then combined to form a complete Bayesian network. The structures for the phases "preform" and
"Blowing part" are created according to section 5.1.

6.1.1. Bayes net, generation of the preform

The structure for the Bayesian network of the "preform" phase was largely defined at the beginning of the work. Structures (possible causes and dependencies) were defined for open defect patterns. The matrix for the creation of the structure was entered into an Excel table (extract see Table 6.1) according to Section 5.1.

Table 6.1: Extract from file "Parameterliste.xlsm", matrix structure Bayes net "preform

Network name	Scratch	Fusion Br	...	Smear formation	Intestinal Effect
VerbrRawstoffDuese	x	x	...	x	
Contaminated raw material	x		...	s	s
TempDuese	x	x	...	x	
BeschaediDuese	x		...	x	x
MatHumidity		x	...		
...	...	...	...	...	...
Heating zone11			...	s	s
Heating zone12		s	...	s	s
Heating zone13		s	...	s	s
Heating zone14	s	s	...	s	

For this phase, a total of 14 errors and 46 possible causes have been discussed and defined. To generate a DAG, several specially written functions are required. These read the file "Parameterliste.xlsm" and are able to generate the acyclic directed graph (see source code 5.2). The definitions of the two main functions (*excel2ms()*

and *createDAG()) can be* seen in source code B.2. The overall network is shown in Figure A.1. Figure 5.3 serves as a further example, showing a section of a Bayes net created in this way.

6.1.2. Bayes net, finished blow moulded product

The structure for the "blow-molded part" phase has been designed from scratch in this work. In the process, 19 possible defects were identified and over 70 causes were defined. Defining the matrix in Excel was also done with expert knowledge. Table 6.2 shows an extract from the Excel spreadsheet. Analogous to Table 5.1 and 6.1, the first column (*network name*) contains the possible causes, followed by the column parent node. There you have the possibility to define parents for a node. The following columns further describe the structure of the individual faults (labels used: *x* direct edge to the fault node and *s* has a connection to the fault node, but not direct).

Table 6.2: Extract from file "Parameterliste.xlsm", matrix structure Bayes net "blow moulding part

Network name	Parents'	Slug waste	Long cycle time	...	Glued areas
Blowing pressure				...	x
Item weight			x	...	
Blowing pressure			x	...	
Extrusion printing		x		...	
Article wall	Blowing pressure		x	...	
...	...	...	...	...	...
WDS				...	x
Tool vent			x	...	
Tool cooling			x	...	
Pre-blowing time		x		...	

The generation of the DAG for the entire network (see Figure A.2) is possible in the same way as in subsection 6.1.1. After generation two Bayes net structures are available ("preform" and "blow mold part"). To complete the networks, after completing the qualitative part, it is now necessary to estimate the conditional probability tables (quantitative part) from data.

6.2. Data acquisition, generation and processing

6.2.1. Machine connection

General possibilities of a connection to the machine with Siemens control are described in subsection 5.2.1. For the exemplary realization of the connection, the open source solution "snap7" serves as the interface between PC/assistant system according to the structure shown in Figure 5.5. The process of data acquisition is shown in Figure 6.1. A trigger in the **R code** starts the data extraction and executes the Python file.

Using "snap7", the data is read from the controller and stored in Python. The **R program** can now access the data and process it further. For using Python code within the **R environment,** the package "reticulate" [1] is used.

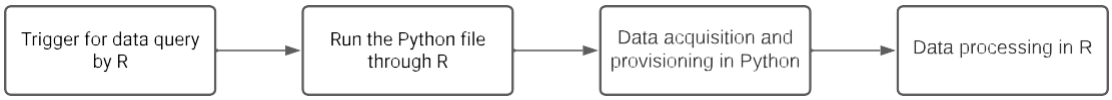


Figure 6.1: Data acquisition process using "snap7

Source code B.5 shows a section of the program code for accessing the machine control. In the main (lines 45-60) a *Clienet ↔ Server* connection is established, then the established connection is checked and finally the current data of the machine is queried. To connect to the PLC, IP address as well as rack and slot number are required. Lines 8 to 42 show the definition of the function *read_data()*, which reads the corresponding parameters from the PLC and makes them available. In the first paragraph, required data blocks are read from data blocks and temporarily stored. In the second paragraph, the individual required parameters are extracted from the data blocks. The library functions for reading a Boolean, an integer and a real variable are shown here. In the last sections, the individual parameters are summarized and provided with a timestamp. As return type the data set is converted into a data frame. The prerequisite for a possible machine connection and completely established connection is that the client (application/system) is located in the same network as the server (machine network).

6.2.2. Acquisition of states of manual nodes/parameters

Section 5.2 deals with the collection of machine data. Data that is not stored on the machine but requires collection is not discussed there. However, since the example process used in this work has a large number of such nodes/data, an optional solution approach is developed and implemented. With this approach it is possible to capture the state of such a node.
The evaluation of the state of a manual node (in the further course nodes are called in such a way, which do not make it possible to seize their state automatically by a CPU) is transacted by the setter and input into the user interface. Since ease of use and reduction of a setter's workload are the primary concerns, partial detection with additional derivations/predictions, rather than full detection, is implemented. During the iterative process of debugging, one observation line is added to the data set in each run. Such an observation maps the current process state (state of error image and states of manual and automatic cause nodes). The query of the states of all manual nodes generates a considerable effort for the setter. To reduce this, only a maximum of the state

[1] **R package** *reticulate*: https://cran. r-project. org/web/packages/reticulate/index. html

of a cause node is queried, which is predicted. Provided that the proposed cause is a manual node. The sequence of the acquisition is shown in Figure 6.2. The goal is to capture the state of a manual node once for the process of error image correction and to be able to derive the states before and after the time of capture without further input from the operator.

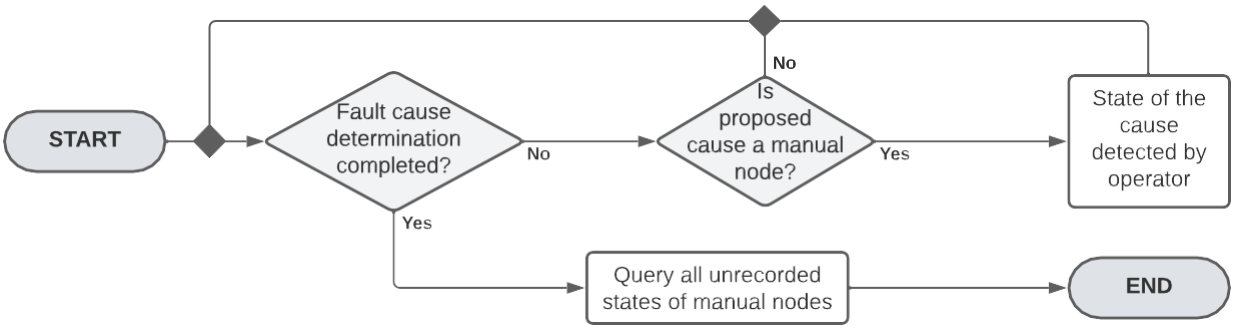


Figure 6.2: Flowchart for recording the states of manual nodes during the process of error image correction

The query of the state of the manual node takes place immediately after the prediction of the system. If the node is in an abnormal state, it is corrected and adjusted by the setter. In the system, the feedback is given that the node was in abnormal state at the time of query. At the same time, it is assumed that the node was also in this "negative" state at past times and will assume a normal state at future times. The assumption has some uncertainty or risk, since past and future states are inferred and not accurately captured. Due to the significant reduction in effort for a setter, the risk present is accepted. An example of a derivation is shown in Table 6.3.
During the setup process, scratches are detected on the preform. The responsible setter reports this to the system and starts the determination of the cause. During the troubleshooting process, the observations shown in Table 6.3 are recorded (red states have been derived by post-processing the function *filldataset()* (see source code B.6)).

Table 6.3: Example of state derivation of manual nodes with error pattern scratch, red states have been derived by the function *filldataset()*

Scratch	ContaminAtion	DamageD	...	Heating zone14	Combustible raw materialD
Yes	Yes	Yes		No	Medium
Yes	Yes	Yes		No	Medium
Yes	Yes	Yes		No	Medium
No	Yes	No		No	Medium

When the cause determination is started, the first observation line is generated (scraper is active and heating zo- ne14 is automatically queried, categorized and stored by the machine). No states of manual nodes are queried here. The assistance system proposes the following as the first cause of error

The system asks the setter about the existing condition. Since it is a manual node, the system asks the setter about the present condition. He acknowledges a normal condition and the further presence of the fault. In the next step, *damage to the nozzle is* predicted. This is checked and the system confirms that there is damage and that the fault has been fully rectified. During the process, the state of the *ImpurityRaw* Node remained unknown. The assistance system prompts the setter to assess the current state of the node. The process of detection is completed.
The incomplete data set is passed to the function *filldataset()* and completed (see Table 6.3 red states). This is a specially designed algorithm that can be used to derive unknown process states. The function can distinguish three scenarios and complete the column of a node accordingly. *Scenario 1:* The state of the node entered by the setter is in any observation (except last observation) and it is a normal state. The function sets all previous and future observations to normal state. Since the node has not been modified by an operator, the assumption is that the state remains unchanged throughout the process. *Scenario 2:* The state of the node entered by the setter is in any observation (except last observation) and it is an abnormal state. Since the operator has independently determined the state and placed the node in a normal state, the function assigns a normal state to all future observations. All previous observations are assigned the abnormal state. *Scenario 3:* The state entered by the setter is in the last observation. This case occurs when the assistance system does not suggest the node during the troubleshooting process and it is recorded after the troubleshooting is successful. Thus, an intervention by an operator during the process has not taken place. The function occupies all previous observations with the state reported back by the setter.

6.2.3. Generation of synthetic data

Due to a missing database at the beginning of the work and the non-existing possibility to collect a sufficiently large amount of data from the real process in a short time, synthetic data are generated and tested with them in the exemplary implementation. For the creation and generation of synthetic data, two approaches (random synthetic and rule-based synthetic data) are implemented on a trial basis.

Random Synthetic Data

For randomly synthesized data neither dependencies between varia- bles/nodes nor distributions of a node are considered during generation. Two functions were written in **R for** this purpose, whereas *simulateRandDataset()* (see source code B.3) generates a dataset for a subnet (error net) and *simulateRandDatasetBig()* (see source code B.4) was programmed for the entire Bayes net. The flow of both functions is identical (see Figure 5.9). For a node, the possible states from the Excel spreadsheet (extended structure Matrix Excel-.

Table see Table 6.4) are extracted. The states of the nodes are determined and defined by experts. It is also possible to select states and classes from data sets. However, due to the lack of data, this step takes place manually. Here, care must be taken to define as few states as possible for a node. This way the size of the CPT's can be minimized. A random draw function sets any one of the possible states in each observation. The distribution of the states is also determined randomly.

Table 6.4: Extract from extended file "Parameterliste.xlsm", matrix structure Bayes net phase "blow mould part" and possible states of a node

Network name	state_default	State_2	State_3	...
Blowing pressure	medium	nierdrig	high	...
Item weight	iO	niO		...
Blowing pressure	medium	nierdrig	high	...
Extrusion printing	medium	nierdrig	high	...
Article wall	iO	niO		...
WDS	iO	niO		...
...	...	...	...	...
Tool vent	iO	niO		...
Tool cooling	iO	niO		...
Pre-blowing time	medium	nierdrig	high	...

The exemplary generation of a dataset for the subnet "Kratzer" is shown in source code 6.1. After reading in the Excel table it is possible to generate random synthetic data by means of the function *simulateRandDataset()* (see Table 5.3).

```
1 # read in excel list with structure matrix and possible states for each node
2 excellist <- readxl :: read_xlsx ("parameter list.xlsm", sheet="error preform")
3 excellist <- customize Excellist (excelliste = excellist)
4 # simulate random synthetic data set
5 dataset <- simulate edge dataset (paralist = excellist, errorname = "scratch", n =
      1000000)
```

Source code 6.1: Generation of random synthetic data

Since neither dependencies nor distributions are considered, but this is necessary for "realistic" synthetic data, a second approach (rule-based) was tested on a trial basis.

Rule-based synthetic data

For the generation of synthetic data, which are closer to real process data in their characteristics, it is necessary to consider dependencies and distributions. The **R** package "bnlearn" offers the possibility to simulate synthetic data (considering dependencies and distributions) using a learned Bayes net (see subsection 5.3.2). Since no real data is available and the Bayes net cannot be learned, the conditional probability tables are filled in using expert knowledge. For the entire

Bayes net, however, this step is very time-consuming and even experts reach their limits with this scope. "bnlearn" offers the possibility to enter distributions in empty CPTs by means of code [34, chapter 1.3]. In this work the software *GeNIe2* was used for this operation, there a graphical interface is available for filling in. As an example, the parameters of the defect subnets "Scratch", "Preform glossy" and "Tube run straight" were manually provided with values. Table 6.5 shows an extract of the CPT for the node "Scratch". These tables have been entered manually (taking reality into account) for each node as an example.

Table 6.5: Extract of a *CPT* manually entered, node "Scratch".

Shameful...					...					...
			Y		...			No		...
es					...			iO		...
TempDuese		Yes	iO	No	...		Yes		No	...
~~VebrRohm...~~	**Ye**	**No**	**Yes**	**No**	...	**Ye**	**No**	**Ye**	**No**	...
ContaR... **Yes**	**s**	0.94	0.91	0.9	...	**s**	0.13	**s**	0.04	...
No	0.98	0.06	0.09	0.1	...	0.51	0.87	0.21	0.96	...
	0.02					0.49		0.79		

Figure 6.3 shows the finished learned subnet of the three defects "Scratch", "Preform shiny" and "Tube run straight". The graphic is taken from the *GeNIe* software. There you have the possibility to display nodes as bar charts. The probability of occurrence in a certain state can be read from the bars and percentage displays.

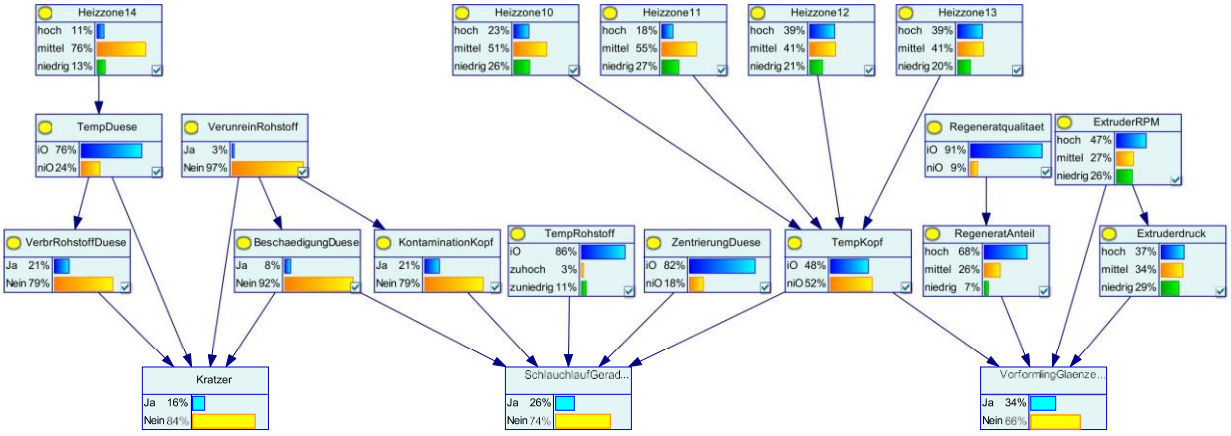


Figure 6.3.: Manually learned subnet for the defects "Scratch", "Preform glossy" and "Hose run straight"; Software *GeNIe*

The manually learned Bayes net makes it possible to generate synthetic data using the "bnlearn" function *rbn(),* where dependencies and distributions are taken into account. Source code 6.2 shows the exemplary simulation of the data. At the beginning, the Bayes net is loaded and stored in a variable (*bayesnet*). Subsequently, the variable
< *bayesnet* > and the length of the required record are passed to the *rbn()* function and

[2]Bayesfusion GeNIe: *https://www.* bayesfusion. com/genius/

a data set is generated. In line 8, the dimension of the variable is determined. It has 20 columns (each node gets one column) and 10,000 rows/observations.

```
1 # read in Bayes Net with three errors
2 bayesnet <- read.net( file = "actual_bn_3 error. net")
3 # simulate synthetic data set
4 dataset <- rbn (x = bayesnet , n = 10000 )
5
6 class( dataset)
7 [1] " data . frame "
8 dim ( dataset)
9 [1] 10000        20
```

Source code 6.2: Generation of rule-based synthetic data using the "bnlearn" function *rbn()*

An extract from the synthetic dataset (generated using function *rbn()*) of the manually learned Bayesian network with the defects "scratch", "preform shiny" and "tube run straight" is shown in Table 6.6.

Two ways are shown to generate synthetic data for training a Bayesian network. In addition to being used as training data, artificially simulated data can also be used to test designed functions and algorithms. Advantages and disadvantages of the two approaches are discussed in a subsequent chapter. The approach of generating synthetic data using AI is not considered in this thesis due to its complexity and scope. This requires exact process/machine models to reflect reality in a digital environment (similar to a digital twin).

Table 6.6: Extract from synthetic data set simulated by function *rbn()* for manually learned Bayes net with three errors

ImpureRo...	Damage to...	Heating zone14	...	PreformGlaenzend
No	No	medium	...	No
No	No	medium	...	No
No	No	medium	...	No
No	No	medium	...	Yes
Yes	Yes	medium	...	No
No	No	medium	...	Yes
No	No	high	...	No
No	No	medium	...	No
No	No	medium	...	Yes

6.3. Root Cause Analysis

The determination of the cause of the error is the main task of the application. Section 5.5 shows and explains two implementation options. Both are exemplified by the example of the

setup process of a blow molding machine and will be investigated and tested in the further course.

6.3.1. Most probable cause of error

To determine the most probable cause of the error, the flowchart from Figure 5.12 is converted into code (source code B.7). Besides the pure implementation of the algorithm, an integration into the user interface is required. By iteratively applying the function *causeanalysis_likeliestdyn()* there is the possibility to consider captured knowledge about the state of a cause node in the calculation. Taking knowledge into account when calculating probabilities represents the "dynamic" adaptation of the system.

The function determines the probabilities that a cause is in a bad state (e.g. *BeschaedigungDuese* = *Yes*), under the condition that the selected fault pattern is present (incl. possible additional knowledge). As transfer parameters, the Bayes net (preform-

ling or blow-molded part) and the existing error pattern. In addition, it is possible to transfer a list of cause nodes and the states in which they are located.

```
1 # read in Bayes Net with three errors
2 bayesnet <- read.net( file = "actual_bn_3 error. net")
3 # estimate the most probable cause
4 causeanalysis_likeliestdyn (bayesnet = bayesnet , error = "scratch")
5
6 P( Impure raw material = Yes | Scratch = Yes ) = 0.1612
7 P( Damage Duese = Yes | Scratch = Yes ) = 0. 4486
8 P( Heating zone14 = high | Scratch = Yes ) = 0. 2086
9 P( heating zone14 = low | scraper = yes ) = 0. 2515
10 P( Temp Duese = niO | Scratch = Yes ) = 0. 4669
11 P( VerbrRohstoffDuese = Yes | Kratzer = Yes ) = 0.458
```

Source code 6.3: Determination of the error cause in case of error pattern "scratch", algorithm for the most probable error cause

In source code 6.3 an exemplary determination of the most probable error cause for the error pattern "scratch" is carried out. Lines 6 to 12 show the calculations performed. The results are sorted in descending order of probability and returned by the function *causeanalysis_likeliestdyn()* (see Table 6.7).

Table 6.7: Output of the most probable cause of error for the "Scratch" error pattern

causeName	causeState	probability
TempDuese	niO	0.4669
VerbrRawstoffDuese	Yes	0.458
DamageDuese	Yes	0.4486
Heating zone14	low	0.2515
Heating zone14	high	0.2086
Contaminated raw material	Yes	0.1612

6.3.2. Most influential cause of error

Analogous to the determination of the most probable cause of error, the function for determining the most influential is implemented. However, in the calculation of the conditional probability there are

"event" and "evidence" are exchanged. The probability of the occurrence of a selected error under given causes is determined. The possibility of taking additional knowledge into account is also possible. Section 5.5 describes that for the complete implementation an automatic alignment with the process or machine state has to take place. However, this is not possible due to the high proportion of causes that cannot be detected automatically. Thus, the matching whether the predicted cause is present takes place manually. The strength of the algorithm is lost by this. Source code 6.4 shows the determination of the most influential fault cause as well as the calculated probabilities for the fault pattern "scratch".

```
1  # read in Bayes Net with three errors
2  bayesnet <- read.net( file = "actual_bn_3 error. net")
3  # estimate the strongest cause
4  causeanalysis_strongestdyn (bayesnet = bayesnet , event = "scratch")
5
6 P( Scratch = Yes | Contaminant = Yes ) = 0.8535
7 P( Scratch = Yes | Damage Duese = Yes ) = 0. 9225
8 P( Scratch = Yes | Heating zone 14 = High ) = 0. 3405
9 P( Scratch = Yes | Heating zone 14 = Medium ) = 0. 1094
10 P( Scratch = Yes | Heating Zone 14 = Low ) = 0. 3186
11 P( Scratch = Yes | Temp Duese = niO ) = 0. 3292
12 P( Scratch = Yes | VerbrRohstoffDuese = Yes ) = 0. 3819
```

Source code 6.4: Determination of the error cause in case of error pattern "scratch", algorithm for the most influential error cause

The return of the function *causeanalysis_strongestdyn()* is shown in Table 6.8. It can be seen that, in comparison with Table 6.7, different predictions were determined for possible fault causes. If *heating zone14* in state "medium" is the most probable cause of failure, the most influential cause is the *damageDuese* in state "yes".

Table 6.8: Output of the most probable cause of error for the "Scratch" error pattern

causeName	causeState	probability
DamageDuese	Yes	0.9225
Contaminated raw material	Yes	0.8535
VerbrRawstoffDuese	Yes	0.3819
Heating zone14	high	0.3405
TempDuese	niO	0.3292
Heating zone14	low	0.3186
Heating zone14	medium	0.1094

Depending on the desired use case, the appropriate algorithm can be selected. The implemented algorithm at the beginning of the work was replaced by the two shown.

6.4. Design user interface

Monitor - without this component, a computer alone would be useless for a user. The user cannot interact with the computer. The user interface in the assistance system has an identical effect. The basic framework was already available at the beginning of the work. Available tabs can be seen in Figure 6.4. These remain, were adapted and extended.

Existing user interface functions are:

Tab: Overall structure; visualization of the two main Bayesian networks "preform" and "blow mold part"; possibility of a zoom function as well as highlighting of connected nodes.

Tab: Process intervention; determination of the cause of the fault for a selected fault pattern present as well as visualization and overview table; possibility of selecting the causes Type

Tab: Defect overview; selection of a defect possible, visualization of individual defect images and possible causes; basic categorization into two main networks "preform" and "blow molding part

Tab: Parameter learning; data upload and post-training of a Bayesian network "preform" or. "Blow molding"; data archiving

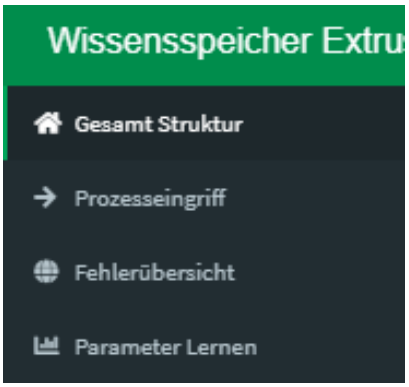


Figure 6.4: Tab overview from the user interface of the knowledge repository extrusion blow molding *ShinyApp*

For information and help on user interface design, please refer to the following literature used [24, 8, 9].
Table 6.9 shows an overview of which adjustments have been made in the respective tab. A distinction is made between the two areas of *user interface*, which is used to design and place inputs and outputs, and *server, which* realizes the functionality and the execution of algorithms. The focus of the visualization is on user-friendliness through clear structures and clearly recognizable "workflows". Likewise, care was taken to provide options for canceling a process or handling incorrect entries and incorrect use.

Table 6.9: Overview of adjustments and enhancements within the user interface, split into *user interface and server*

Tab	user interface	Server
Total Over	- Drop-down menu for selection	- Query the selected main Bayes Net-
View	of the main Bayes net - Display charging process	and corresponding plot
Process intervention	- Cause selection option	- Query of the selected cause type
Error overview	type - Drop-down menu for selecting the error category - Button for process abort - Display charging process - Drop-down menu for selection	- Query of the selected error category and corresponding loadings Main Bayes net and parameter list with matrix of the structure - Process of determining the cause of a fault - Generation of pop-up windows for status - Aufrufen der Funktion zur Anpassung der Dropdown
Parameter Lernen	the error category - Display charging process - Drop-down menu for selection of the main Bayes net - Drop-down menu for selection of the Parameter learning	- Selection of the selected error category. - Error category dependent loading Parameter list with matrix of the structure - Erlernen des neuen Bayes Netzes - Storage of the newly trained network ent-

6.4.1. Tab Gesamtübersicht

The Tab *Total Übersicht* is loaded when opening the Mösgplircehckheein. Here you can see the Bayes Networks "preform" or "blow-molded part" and to obtain an overview of this (see Figure 6.5). Error nodes (yellow color) and cause nodes (green color) are displayed in different colors for the operator.

Adaptation within this work was the possibility of the Bayes net selection. It can be freely selected which Bayesian network is to be displayed. The selection is made via a drop-down menu. Source code 6.5 shows the programming of the user interface. In this area, inputs and outputs are created and placed on the interface. In lines 4 to 6, the drop-down menu for network selection is created. In line 8 a button for updating the graphic follows and in line 10 the visualization is created.

```
1 ## Tab OVERVIEW
2 tab Item(tabName = "overview",
3   # choose between 'preform' and 'blow moulding'.
4   selectInput(inputId = "idi_bayesnettype",
5     label = "Selection of preform or blow-molded part",
6     choices = c("preform", "blow molding")),
7   # refresh button for plot
8   actionButton(inputId = "idi_ac_refresh BN", "Refresh"),
9   # output plot
10  uiOutput(outputId = "ido_bayesnetz_total_plot"),
11  # loading process
12  add_busy_spinner(spin = "fading-circle", timeout = 100)
13 ), # end tab overview
```

Source code 6.5: Tab *Total Overview* User Interface Shiny

To reload the graph, use the previously referenced "Refresh" button (line 8). The display is an interactive graph. This allows individual nodes to be moved and zoomed. By clicking on a node, further nodes which have an edge to this node are highlighted and others are greyed out. This also serves to provide an overview. To illustrate a loading process, a rotating circle is displayed in the upper right corner (see source code 6.5 line 12). Inputs during a loading process are discarded. The programming on the server side is shown in source code B.8.

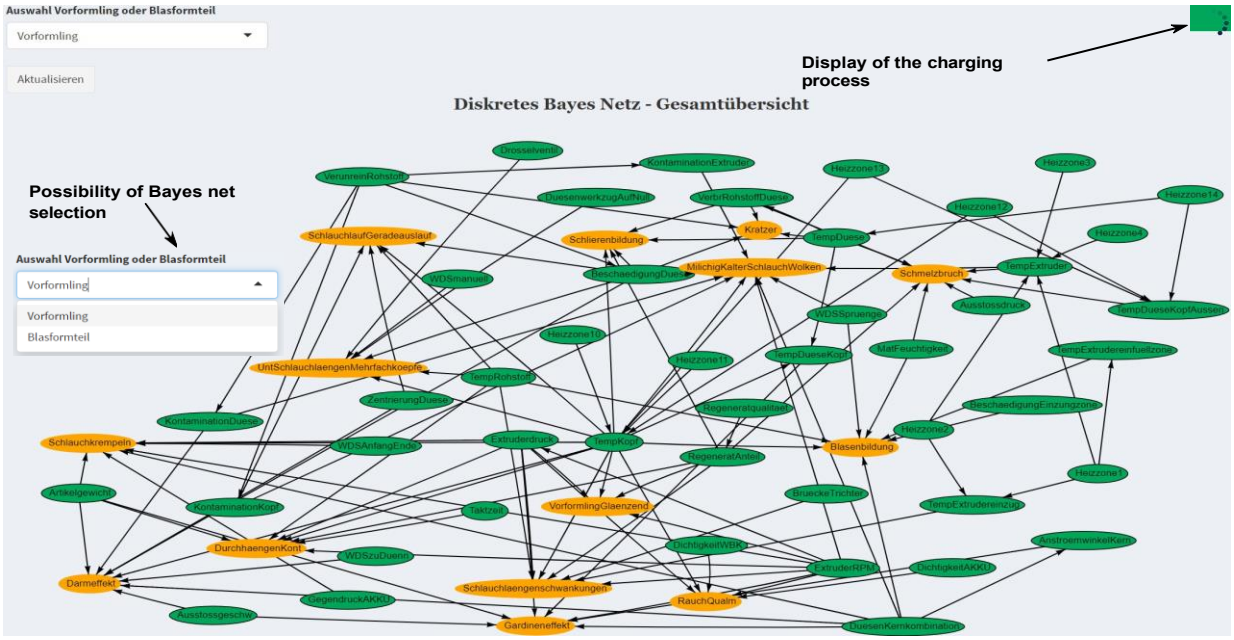


Figure 6.5.: Tab *Total overview* from the user interface of the knowledge repository extrusive blow molding, selected Bayes net "preform".

6.4.2. Tab process intervention

The Process Intervention tab is used particularly frequently by a setter, since this is where the error correction or error cause determination takes place. In Figure 4.4, the status at the beginning of the

work shown. The adapted and extended state can be seen in Figure 6.6.
Under *(1)* the operator is given the option to select the type of cause (most probable or most influential). Drop-down menu *(2)* is used to categorize whether the existing defect pattern is assigned to the preform or the finished blow-molded part. The last selection option is offered by the drop-down menu *(3)*, where the current defect pattern is selected. After the corresponding configuration, the process can be started via the "Defect cause determination" button. The assistance system displays the predicted cause to the setter both textually and graphically. In the graphical output, the error node is displayed in green color and the predicted cause node is displayed in red color. In the text output, both the cause of the fault and the state in which it is located are included. The state is relevant for the setter, e.g. in the case of temperatures, since he needs the information from the assistance system as to whether this is too low or too high.

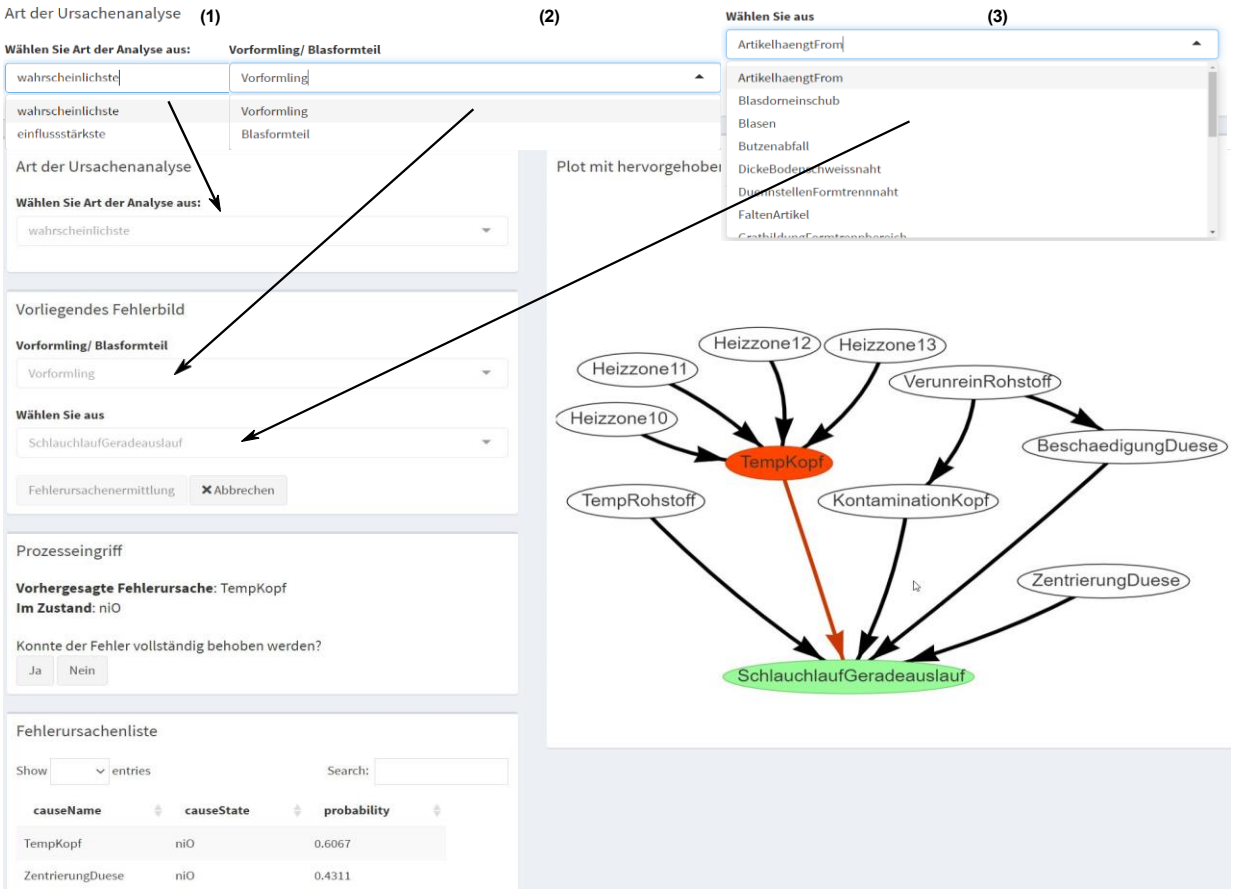
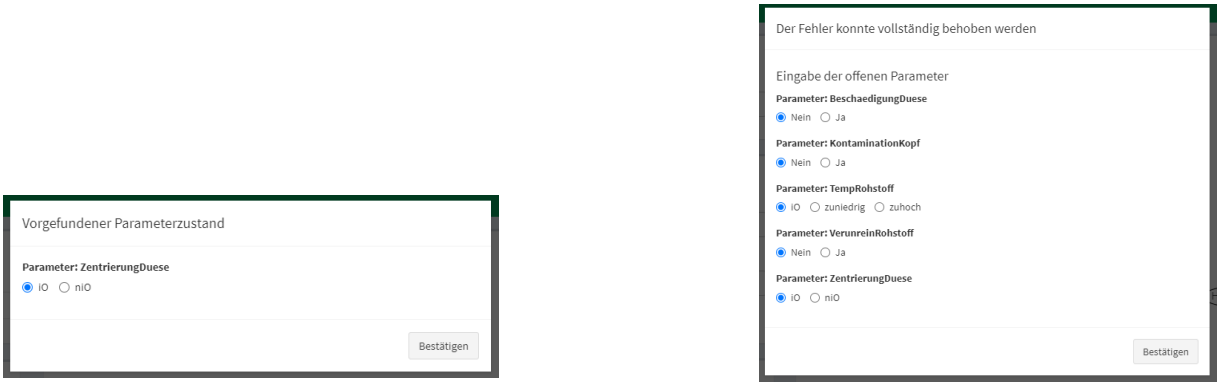


Figure 6.6.: Tab *process intervention* from the user interface of the knowledge base extrusion blow molding, with algorithm "most probable" cause, Bayes net "pre-forming" and error image "tube run straight".

The user interface expects feedback from the operator as to whether the error has been corrected or not. If the input is made that an existing error is not corrected, a window appears in the case of a parameter that cannot be detected automatically, in which the existing state of the node/parameter is queried (see Figure 6.7(a)). After subsequent

Confirm, a new cause is predicted, taking into account that the previous cause is in a known state (extension of the conditions, when calculating conditional probability). If an error is completely eliminated and this is acknowledged by the operator, a window appears in which nodes/parameters states not detected during the process are queried (see Figure 6.7(b)). Only nodes/parameters that cannot be captured automatically are found there. By default, each node is set to a "good" state, this minimizes the effort for a setter as he only needs to make an input in case of abnormal states. The data is required to be able to map the process image completely and to retrain the network with collected data at a later stage.



(a) (b)

Figure 6.7.: Tab *Process Intervention* Modal Window; (a) Query of the present state of a manual parameter, (b) Query of the manual parameters at the end of the troubleshooting process, where the state is unknown to the system.

In the *process intervention* tab, adjustments and extensions were necessary in the area of *user interface* and server side. The structure and the design have been adapted. As described above, drop-down menus and buttons have been added (user interface side, see source code 6.6). The tab is graphically divided into two columns, in the following source code the left column is shown. The functional process has been fundamentally rebuilt on the server side.

```
1 ## Tab Error Cause
2 tabItem(tabName = "errorcause",
3   fluidRow(
4   ## left column
5     column(width = 4,
6       ## type cause analysis
7       box(width = NULL, title = "Type of root cause analysis",
8         uiOutput(outputId = "ido_causetyp")
9       ),
10      ## fault box
11      box(width = NULL, title = "Present error image",
12        # drop-down menu error category
13        selectInput(inputId = "idi_bayesnetart",
14          label = "Preform/blow molding",
15          choices = c("preform",
16            "blow molding")),
```

```
17          # drop - down menu error pattern
18          uiOutput( outputId = " ido _errorpattern "),
19          # button start root cause analysis
20          action Button ( inputId = " idi_ab_errorcausecal",
21            label = " error cause determination "),
22          # cancel button
23          action Button ( inputId = " idi_ab_stop ",
24            label = " Cancel ",
25            icon = shiny :: icon (" times")),
26          # loading process
27          add _busy _spinner( spin =" fading - circle ", timeout
            =100)
28        ),
29 ---
30 )),
```

Source Code 6.6: Tab *Process Intervention* User Interface Shiny, Adjustments and Enhancements

6.4.3. Tab Error overview

The examination of a single error pattern and the possible causes defined for it are implemented in the *Error overview* tab (see Figure 6.8). In addition to the status found at the beginning, the selection option of the error images has been expanded. The user can select the relevant error via two drop-down menus (1. error category and 2. error image). The visualized Bayes net highlights the error node in a light blue color. In contrast to the visualization of a main Bayes net in the tab *Total Overview,* this is not an interactive graphic. A zoom is therefore not possible. Care has been taken in the design to choose sufficiently large fonts. By using static graphics, resources can be saved.

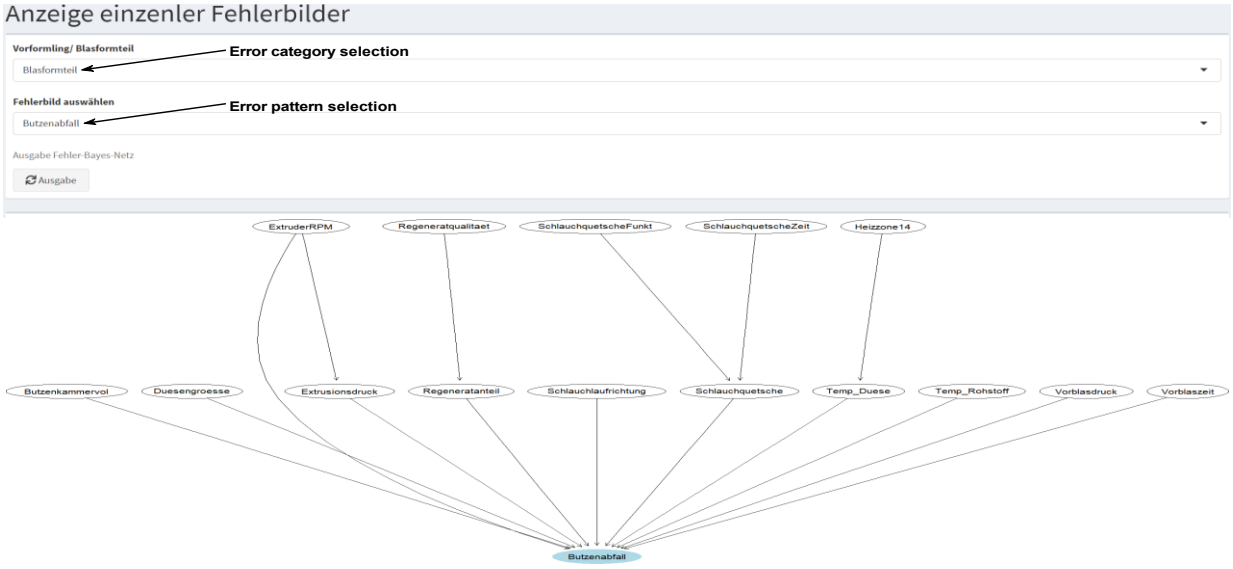


Figure 6.8: *Error overview* tab from the user interface of the knowledge repository Extru- sion blow molding; Selected error category "Blow molding part" and error image "Slug waste"

An excerpt of the server-side programming for the *Error Overview* tab is shown in source code B.9. Depending on the selected error category, the drop-down menu is updated with possible selectable error patterns. At the same time, when an error category is selected, the corresponding parameter list with contained matrix structure is loaded from the working directory. Based on this, a subnet is generated and output for the selected error image.

6.4.4. Tab Parameters Learning

The tab *Parameter Learning* is one of the two tabs that represent and enable the main functions of the assistance system, along with the tab for determining the cause of errors. The basic structure of the tab was available at the beginning of the work, so only adjustments were made. An operator is given the opportunity to distinguish between the two main networks and then upload a record (*Browse* button). For training the uploaded dataset had to be saved before (old state). After the new state the saving is executed automatically and it is only necessary to "click" on the button *Learn parameters*. After learning has been carried out, the operator receives a message that the parameters in the network have changed.

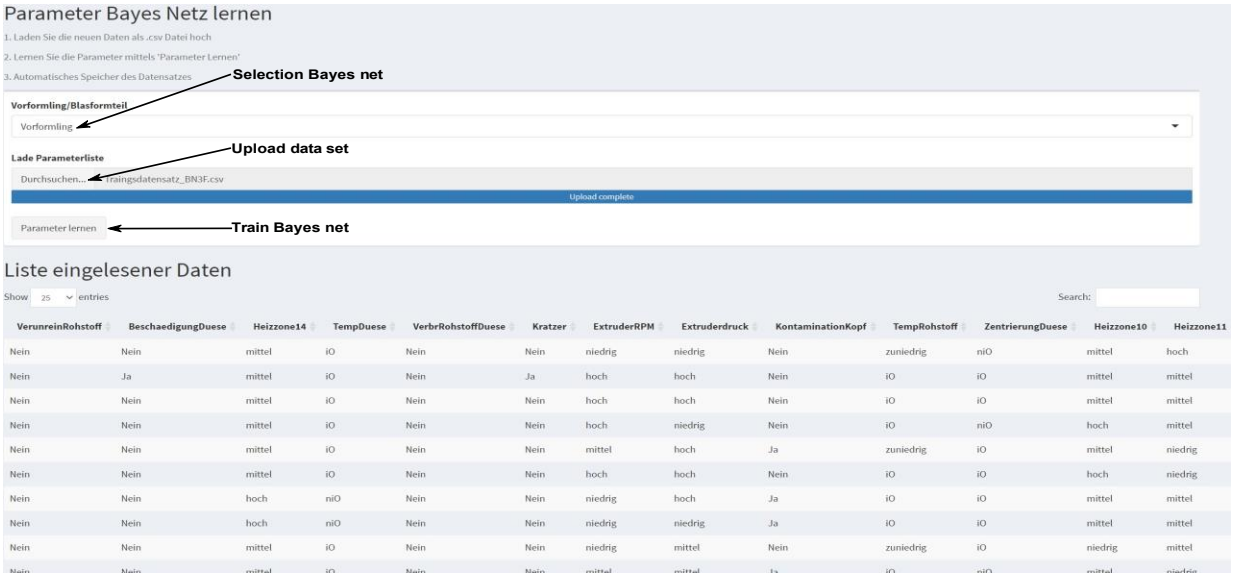


Figure 6.9.: Tab *Parameter Learning* from the User Interface of the Knowledge Store Extrusive Blow Molding; Selected Bayesian Network "Preform".

In source code B.10 a section of the tab *Parameter Learning of* the server side is shown. There the selection of a main net is implemented. For initialization the Bayes net "preform" is loaded by default. In lines 5 to 10, the query for the selected net (query input parameter *idi_bnart*) and the corresponding loading of the Bayes net takes place. By means of *observeEvent() the system* reacts to value changes of the parameter. The lines 13 to 34 realize the learning process, there it reacts on "click" of the button *idi_learn* (parameter learning). In 20 to 25 a query for the selected main Bayes net takes place again, the trained net is saved accordingly.

7. Evaluation, analysis and discussion

The previously described implementation approaches and implementations (see chapter 6) are evaluated, analyzed and discussed in the following. In addition to functional proofs, a qualitative evaluation of the different algorithms takes place. A quantitative evaluation, e.g. the progression of the predicted causes until an error is corrected, considered over the number of performed training runs or a general "hit rate", cannot be presented in this thesis. Statements and evaluations about this can only be made after the knowledge store has a certain usage time and performed learning processes. An AI develops over its used time. Therefore, quantitative evaluations are not possible at the beginning. Due to the lack of real data, tests are carried out with synthetic data.

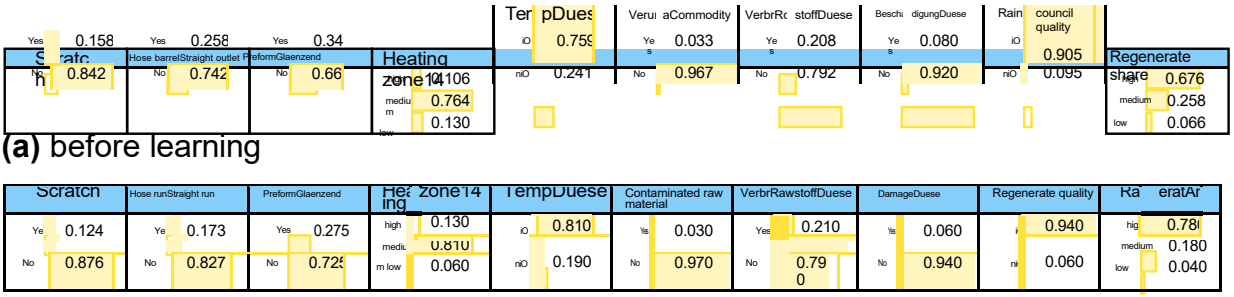
7.1. Proof of function Parameter learning

For the synthesis of the Bayesian network to the environment, it is possible to manually trigger the "parameter learning" process. The proof of function is shown in the following and is performed via the system interface. Using the function *rbn()* a synthetic training data set (see Figure C.1) has been created for the reduced main network "preform". This is deployed and used during the test. Table 7.1 shows the documented proof of function, where the functions have been tested iteratively from top to bottom. The individual steps performed are shown there. There are no negative findings. For the test, the manually learned Bayes net was used with the three defect patterns "scratch", "preform shiny" and "tube run straight".

Table 7.1: "Parameter learning" function verification protocol

Action/ FunctionFound/	**Result**	**Comment**
Selection Bayes net	i.O.	
Up-load record	i.O.	
Data record archiving	OK.	In the Data_archive folder, file data_download
Start learning processi	.O.	Pop-up window with warning that parameters have been changed
Saving a newly trained Bayesian network	OK	
Learning process completed	OK	

A comparison of the distributions of a selected number of nodes from previously described Bayes net before and after "parameter learning" can be seen in Figure 7.1. Upper plot (a) shows the distributions before learning and lower plot (b) shows the distributions after training. It can be seen that the parameters have been changed, but the sum of the distributions of a node remains at 1. A proof of correctness can be dispensed with, since the algorithm for determining the parameters comes from a library function and was not designed independently.



(a) before learning

(b) after learning process

Figure 7.1: Comparison of distributions after parameter learning, manually learned Bayesian network, training dataset created with *rbn()*

The bar graphs shown (Figure 7.1) are the absolute occurrence probabilities of a node. The overview of the occurrence in a specific combination of states of the parent nodes is anchored in the CPT of a node. To view this, it is possible to display the desired tables (see Figure 7.2).

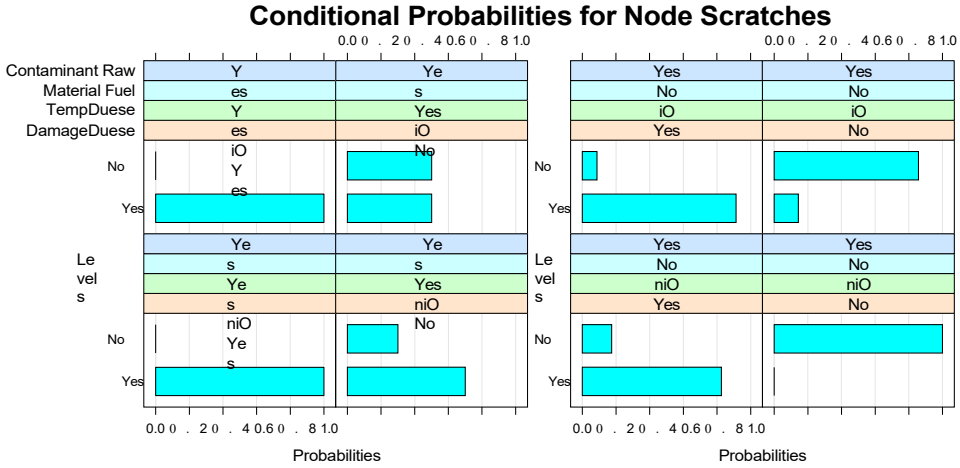


Figure 7.2: Conditional probability table of the node "Scratch" after parameter learning

7.2. Algorithms Error cause most probable or most influential

The correctness of the two implemented algorithms for the cause of the error is checked. For this purpose, a manually learned Bayes net is used. Since the correctness of the implementation and structure of the algorithm is tested with respect to an existing Bayes net and not in

With respect to reality, only a general learned network is required. It is irrelevant whether it was trained with random, synthetic or real data.

To test the *most probable* cause algorithm (see source code B.7), the fault pattern *hose run straight ahead* (see figure A.3) is selected. The conditional probability formula is used as the basis for calculating the inference. It is not the calculation of the conditional probability that is questioned, but whether the algorithm outputs the cause which is in a bad state (event) when the fault (evidence) has occurred. In the following, the equations calculated by the algorithm are presented in descending order of the result (see Figure 7.3).

$$
\begin{aligned}
P(\text{TempKopf} = \text{niO} \mid \text{SchlauchlaufGeradeauslauf} = \text{Ja}) &= 0.6072 \\
P(\text{ZentrierungDuese} = \text{niO} \mid \text{SchlauchlaufGeradeauslauf} = \text{Ja}) &= 0.4353 \\
P(\text{Heizzone12} = \text{hoch} \mid \text{SchlauchlaufGeradeauslauf} = \text{Ja}) &= 0.4052 \\
P(\text{Heizzone13} = \text{hoch} \mid \text{SchlauchlaufGeradeauslauf} = \text{Ja}) &= 0.4043 \\
P(\text{KontaminationKopf} = \text{Ja} \mid \text{SchlauchlaufGeradeauslauf} = \text{Ja}) &= 0.2968 \\
P(\text{Heizzone11} = \text{niedrig} \mid \text{SchlauchlaufGeradeauslauf} = \text{Ja}) &= 0.2766 \\
P(\text{Heizzone10} = \text{niedrig} \mid \text{SchlauchlaufGeradeauslauf} = \text{Ja}) &= 0.2608 \\
P(\text{Heizzone10} = \text{hoch} \mid \text{SchlauchlaufGeradeauslauf} = \text{Ja}) &= 0.2403 \\
P(\text{Heizzone12} = \text{niedrig} \mid \text{SchlauchlaufGeradeauslauf} = \text{Ja}) &= 0.2058 \\
P(\text{Heizzone13} = \text{niedrig} \mid \text{SchlauchlaufGeradeauslauf} = \text{Ja}) &= 0.2053 \\
P(\text{Heizzone11} = \text{hoch} \mid \text{SchlauchlaufGeradeauslauf} = \text{Ja}) &= 0.2016 \\
P(\text{TempRohstoff} = \text{zuniedrig} \mid \text{SchlauchlaufGeradeauslauf} = \text{Ja}) &= 0.1408 \\
P(\text{BeschaedigungDuese} = \text{Ja} \mid \text{SchlauchlaufGeradeauslauf} = \text{Ja}) &= 0.1191 \\
P(\text{VerunreinRohstoff} = \text{Ja} \mid \text{SchlauchlaufGeradeauslauf} = \text{Ja}) &= 0.0506 \\
P(\text{TempRohstoff} = \text{zuhoch} \mid \text{SchlauchlaufGeradeauslauf} = \text{Ja}) &= 0.0419
\end{aligned}
$$

Figure 7.3: Calculations of the assistance system to determine the *most probable* cause of failure, results sorted in descending order.

In previous calculations, probabilities are calculated only for cause nodes which are in a bad state. The case for the conditional probability
"TempHead=iO" as an example is therefore not calculated.
To check the correctness of the probabilities, the Bayes net is loaded with the software *GeNIe*. The calculations are also performed there (see Figure A.4). It can be seen that the *TempHead* node is also determined as the most probable cause of the error. After feedback to the system that the error has not been corrected and the node "TempHead" is in the state "niO", calculations are performed again. In this one, the knowledge that the state of the node "TempHead" is known is taken into account. Example equation:

$$P(\text{CenteringDuese} = \text{niO} \mid \text{HoseRunStraight} = \text{Yes} \ \& \ \text{TempHead} = \text{niO})$$

An overview of the results is shown in Table 7.2. The most probable cause of the error is now identified as "heating zone12" in the "high" state. A comparison with the outputs from *GeNIe* shows identical results (see Figure A.5).

Table 7.2: Results from the calculation of the conditional probabilities with the evidence → *Hose runstraightout = Yes & TempHead = niO*

Cause node	State	Probability
Heating zone12	high	46.50
Heating zone13	high	46.40
CenteringDuese	niO	35.38
Heating zone11	low	30.54
Heating zone10	low	29.24
Heating zone10	high	29.00
Heating zone11	high	26.75
ContaminationHead	Yes	26.06
Heating zone13	low	23.29
Heating zone12	low	21.53
TempCommodity	increasingly low	13.60
DamageDuese	Yes	9.57
Contaminated raw material	Yes	4.56
TempCommodity	too high	3.26

Considering that the algorithm performs calculations based on the underlying trained Bayesian network, it is able to determine the most probable cause. However, since the process image is only iteratively integrated into the calculations, deviations can occur in reality.

The proof for the algorithm of the *most influential* cause is performed analogously. The identical Bayes net is used and the error pattern "hose run straight ahead" is also examined. The conditional probability serves as the mathematical basis. In contrast to the calculation of the most probable cause, "event" and "evidence" are interchanged. The probability is determined that a fault is *active,* given that a cause is in an abnormal state, see the following equation.

$$P(\text{error} = \textit{true} \mid \text{cause} = \textit{state}_{bad})$$

Table 7.3 shows the results of the calculation for the *most influential* cause. In order to validate the results, the analogous determination was carried out using the *GeNIe* software (see Figure A.6). In both cases identical probability values are calculated. Thus, the most influential cause is predicted to be the abnormal centering of the nozzle when the fault pattern "hose run straight out" has occurred. The test is limited to a correct determination in relation to the present Bayes net and does not represent a necessary connection to reality in the test case. This is sufficient for a functional verification.

Table 7.3: Results from calculation of the conditional probability with the event → *Hose runstraight runout = Yes*, to determine the most influential cause of error

Cause node	State	Probability
CenteringDuese	niO	64.19
Contaminated raw material	Yes	40.91
DamageDuese	Yes	37.95
ContaminationHead	Yes	37.49
TempCommodity	increasingly low	32.43
TempCommodity	too high	31.25
TempHead	niO	29.68
Heating zone11	high	28.20
Heating zone13	low	27.15
Heating zone13	high	26.62
...	...	...

When comparing both algorithms, it can be seen that different nodes are predicted as the cause of a fault that has occurred. In the case of the most probable cause for the error "hose run straight out" it is the *head temperature*. In the case of the most influential cause, it is the *centering of the nozzle*. The operator/setter would like to have predicted the cause which is actually responsible for the error. Example error pattern scratch:

most likely cause

$$P\,(cause = state_{bad}\,|error = true) \rightarrow \text{TempDuese}$$

strongest cause

$$P\,(error = true|cause = state_{bad}\,) \rightarrow \text{BeschaedigungDuese}$$

The probability that the nozzle temperature is in an abnormal state is 60%, whereas it only influences the fault pattern by 30%. The damaged nozzle raises the probability of occurrence of the fault to 90% and has a probability of occurrence of 45%. Is the most influential or probable cause responsible for the fault? An influential cause in this sense means that if it is present, the occurrence of a fault is significantly favored, but it may be that it has only a very low probability of occurrence. A probable cause is defined as the cause that is "most likely" to occur in case of a confirmed error. However, this cause may not have a great influence on the error, this is only taken into account in the calculation to a limited extent. Both algorithms require a proof under real application. This makes it possible to identify the "better" algorithm (higher hit rate) by benchmarking. In case of a non-existing possibility to capture the complete process state (states of all nodes) automatically, it is assumed that the algorithm for the most probable cause is the appropriate one in reality. In this case, it receives suggestions, which with the greatest probability

and at the same time favour the error. In the other case, the probability of occurrence is neglected, which in reality would lead to additional work for the setter. If there is the possibility to capture the complete process state automatically, the predicted most influential cause could be compared to it and only be output if it matches. The assumption would be that in this case the most influential cause algorithm is the "best" one. This assumption is taken up and described in more detail in subsection 8.4.1.

7.3. Proof of function Fault cause determination

The next step is to examine the functionality for determining the cause of the error. The tests are performed in the *Process Intervention* tab (Figure 6.6) of the application. Table C.1 shows the individual tests required for the detection. The tests did not produce a negative partial or overall result. The function is therefore given and correctly describes the process according to Figure 7.4. The quality and correctness of the results in relation to reality cannot yet be assessed at this stage, as previously mentioned.

Figure 7.4 describes the process of determining the cause of the error and correcting the error. Process steps with a green background require an action by the setter/operator. The causes of faults are suggested by the system until the operator reports that the fault has been rectified.

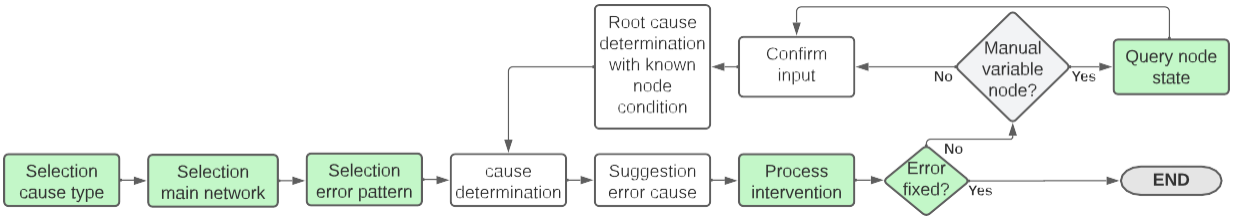


Figure 7.4: Process of fault cause determination and fault elimination

7.4. Comparison algorithm most likely cause with and without dynamic adjustment

The task within this thesis has been to investigate and evaluate a possible dynamization of the assistance system. In the following section, the algorithm for the most probable cause of an error is compared with and without the inclusion of acquired knowledge. The implemented algorithm determines the most probable cause considering known node states. This consideration represents the dynamic adaptation of the system. If an error cause is proposed and the operator reports back to the system that the error pattern has not been eliminated, the state of the cause is recorded and taken into account in renewable calculation. The operator receives a newly determined cause, which is predicted as the most probable one according to the current "process image" (cf. Figure 5.14).

The algorithm to be compared does not have this adaptation. Thus, at the beginning of an error, the causes are determined, which are then sorted in descending order of probability. In each iterative loop pass, until the error is completely eliminated, the top error cause is taken from this list and examined by the setter for its influence (e.g. checking whether "contaminated raw material" is present). A recalculation does not take place during the process.
For the comparison of the two algorithms, the defect pattern "scratch" is used, here this is based on the part network of phase
"Preform" used as a basis. The first prediction of the most probable cause results in a "*TempDuese*", which is in the state "niO". Here, only calculations based on the Bayes net are performed. Table 7.4 shows the results from first prediction in left sub-table. If the error is not eliminated by first cause and the feedback is given that "TempDuese" is OK, the setter checks in the next step whether burnt raw material is present on the nozzle.
With newly implemented algorithm the knowledge (*TempDuese = iO*) is used for a new prediction. The result is shown in the right sub-table (see Table 7.4). After this, an operator next checks the nozzle for damage. The possible cause
is therefore no longer "combustion raw material" after a new calculation, but rather "damage to the gas". By including the current process image, calculations can be adjusted. The sequence has changed in contrast to the first calculation (left). This makes it possible for an operator to determine the cause of the error in a more targeted manner. Although this algorithm requires more computing power, it enables an adapted prediction.

Table 7.4: Determination of most probable cause for defect pattern "Scratch", with second before-say node condition of "TempDuese" also taken into account

P(cause=state(bad)|error=true)			*P(cause=state(bad)|error=true&TempDuese=iO)*		
causeName	**causeState**	**prob**	**causeName**	**causeState**	**prob**
TempDuese	niO	49.12	Beschaedigun	Yes	68.02
VerbreRohsto.	Yes	47.01	Contaminated pipe	Yes	25.41
Damage.	Yes	45.26	VerbreRohsto	Yes	7.51
Heating zone14	low	26.05	Heating zone14	high	0.16
Heating zone14	high	21.51	Heating zone14	low	0.14
ImpureRoh.	Yes	16.90			

7.5. Comparison of cause determination algorithm implemented at the beginning of the work and newly implemented for the most influential cause

At the beginning of the paper, one root cause detection algorithm is already implemented, which is replaced by two newer ones in the course. The flow chart of the old algorithm is shown in

Figure 7.5. Here, the relative changes in the occurrence probability of a fault pattern are calculated by a change of state (from a "bad" state to a "good" one) of a cause node. In this case, the state change that has the maximum change is considered to be the most influential cause for the occurrence of a fault. The newly implemented algorithm, on the other hand, does not calculate a relative change, but the absolute probability of occurrence of a fault pattern.

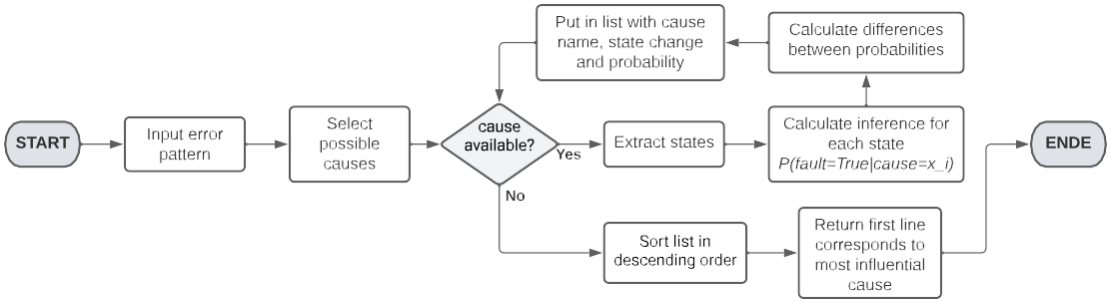


Figure 7.5: Flowchart algorithm determination of causes at the beginning of the work

The results of the newly implemented algorithm provide information about the probability of an error occurring, but no statements can be made about how much this is reduced once the cause node is in a normal/good state. For this reason, the assumption is made that the probability of a fault is minimized as soon as the cause is in a normal state. This assumption is refuted by the following overview. Table 7.5 shows to the left hand the results of the new algorithm and to the right hand the results of the algorithm at the beginning of the work. There it can be seen that the orders of the cause nodes have changed. Although the centering of the nozzle is still in the first place, shifts have taken place behind it. These are based on the remaining probability of occurrence when the cause node is in a normal state.

Table 7.5: Most influential cause for error pattern "hose run-out"; (**left**) newly implemented algorithm, (**right**) old algorithm at the beginning of the work

Cause-Node	**State**	**Wkt.**
CenteringDuese	niO	64.19
Contaminated raw material	Yes	40.91
You.	Yes	37.95
ContaminationCo.	Yes	37.49

Cause node State wt.
CenteringDuese niO to iO 46.76
ContaminationCo. Yes to No 14.90
ContaminationCommodity Yes to No 14.81
DamageDu. Yes to No 13.92

The newly implemented algorithm has been proven to provide the operator with the most influential cause, but this does not necessarily mean that the probability of error is minimized by changing the state to a normal state. This is only achieved with the old algorithm.
Which algorithm in real use provides the operator with the fastest and most accurate error cause, which is responsible for the occurrence of an error pattern, cannot be determined in this work.

be proven. This must be verified as soon as the system is used in a real environment. For the verification in a digital environment, a realistic machine model with identical behavior would have to be developed, which is beyond the scope of the work.

7.6. Evaluation of the different implementations of the fault cause investigations and remedies

At the current stage, no quantitative evaluations can be carried out, since the knowledge store must first be in use for a certain time and must adapt to the real environmental conditions. Functional proofs were possible with the help of synthetic data. Furthermore, in the following a qualitative evaluation of the three existing implementations for the determination of the cause of the error takes place. Hereby the handed over state at the beginning and the two newly implemented algorithms are compared under different categories. For the evaluation a point system and a weighting of the categories are created.
Evaluation scheme:

- Points: very good (3), medium (2), poor (1)
- Weighting: very strong (5), strong (4), medium (3), weak (2), very weak (1)
- Evaluation: The higher the total score, the better the result.

The structure of the evaluation scheme (possible points and weightings) is based on own thoughts. The evaluation categories (see Table 7.6) are chosen in such a way that they give specifications to an algorithm in a certain way. These are of interest to the user. The score achieved should give information about the quality and usefulness of the system for a setter. The maximum score that can be achieved is 72 points.

Table 7.6: Qualitative evaluation scheme of existing algorithms and implementations for fault cause identification and elimination

	Number of causes until error corrected **(5)**	Work effort servant **(3)**	Operability **(4)**	Hit rate "First Try" **(4)**	Accuracy in the calculation **(2)**	Memory requirement **(1)**	Computing time **(1)**	Consideration of knowledge **(4)**	**Total points**
Old stock for Handover	2	1	2	2	3	1	1	1	***41/72***
Re-implementation "most influential"	1	2	3	1	2	2	2	3	***47/72***
Re-implementation "most likely"	3	2	3	3	2	2	2	3	***65/72***

When assigning the scores, one category was selected at a time and then the three existing implementations were evaluated. If a score occurs more than once in a category, these implementations were evaluated in the same way. The condition that all three types of points are always awarded does not apply. Categories such as *number of causes until error is corrected* or *hit rate "first try"* require an estimate (subjective evaluation). Other categories, however, can be "measured" and the results compared. This is possible, for example, for the category *calculation time* (see Table 7.7).

Table 7.7: Measurement of the computing time of the implemented algorithms

Algorithm	Average duration of the calculation in seconds
Old stock	7.9
New most influential	0.5
New most likely	0.5

After qualitative evaluation, the newly implemented algorithm for determining the most probable cause achieves the highest score (65/72 see Table 7.6). This therefore appears to be the most suitable for use. Later tests in reality will confirm or refute this result.

7.7. Discussion

The algorithm for determining the *most influential* fault cause loses its strength due to the currently non-existent possibility of an automatic comparison with the machine or process state (see subsection 8.4.1). It is possible that an influential cause only occurs in a small percentage of fault cases. However, if it occurs, it is mainly responsible for the defect pattern. Due to this assumption, the assistance system will predict a cause in "many" cases when determining the most influential failure cause, which is not present in the real process. Due to the omission of the automatic process adjustment and thus manual acknowledgement by an operator, an increased effort arises for the operator until the actual cause of the error is localized.
The comparison with the present process state should at the same time be a dynamic adaptation of the system. Statements of the system to an operator would thus be made depending on the state and the operator would only receive suggestions for which an intervention in the process and changes/adjustments are possible.

When calculating conditional probabilities based on a Bayes net, the function *cpquery()*[1] from the **R package** "bnlearn" is used. This is not an exact calculation, but an estimation. If an inference is queried twice, two different results will result from an identical query. To reduce the deviations, it is possible to specify the number of samples to be drawn (parameter *n*) as an option with

[1] *cpquery(fitted, event, evidence, method = "ls",...):* Estimates the conditional probability of an event given evidence using the method specified in the argument method. [35]

to be specified. This significantly increases memory requirements and computation time. In the process of determining the cause of the error, it can happen that the number of conditions increases when calculating an inference. The estimation becomes more complex and deviations larger, a high number of samples thus unavoidable.
The influence of the parameter n on the result is shown below; the exact result can only be achieved with $n = 10^6$. In the real application it is necessary to check whether the currently used number provides the desired result.

$$P\,(HeatingZone14 = low | Scratch = Yes \& TempDuese = niO \& V\,erbrRaw\,Duese = Yes)$$
$$Parameter\ n = 10^4 : P\,(x) = 0.50$$
$$Parameter\ n = 10^5 : P\,(x) = 0.52$$
$$Parameter\ n = 10^6 : P\,(x) = 0.53$$
$$GeNIe\ exact\ inference: P\,(x) = 0.53$$

The use of a knowledge store requires data for learning its network and for later adjustments, but states of nodes are also queried during use. It is therefore essential to be able to capture parameters of the process/machine. In the blow molding process used as an example, however, a large number of the cause nodes cannot be automatically detected at the current time. These include environmental influences, which the company does not record, or damage and functional tests on system components, which represent a complexity of their own for automatic recording.
This circumstance makes data collection more difficult and thus results in a lower number of data per time, since the use of a human is required. At the same time, a certain uncertainty factor/noise is implied by ma- nual/ visual judgement of an operator. When classifying "raw material moisture", a setter has to check by hand if the material is too moist or ok. If ten operators will perform this classification, the result will be that some will classify it as too wet and some will classify it as okay. This problem arises due to non-automatic detection. With a sensor for moisture measurement, this problem can be circumvented. In addition to the error-proneness due to human judgement, there is also an increase in the effort required for this, as the system needs feedback on the current state of the process. Section 8.3 revisits the issue of data acquisition.

The structure of the two Bayes networks was defined with the help of an expert. Here, possible causes were defined for a fault pattern and, if necessary, dependencies among the causes were determined. Since no real data is available at this point, this is the only possible process for creating the structure. In case of existing data, there is the option to learn the structure by means of algorithms [34]. This step should be performed as soon as a sufficient amount of data has been collected. In this process, the strength of an edge can also be determined and compared with the manually defined structures. The **R package** "bnlearn" offers various

functions that make it possible to determine the strength of edges between nodes and to create directed graphs where only significant edges are considered. Among other things, it is possible to use *arc.strength()*[2] to determine the strength of edges based on a DAG and data set used to learn the structure.

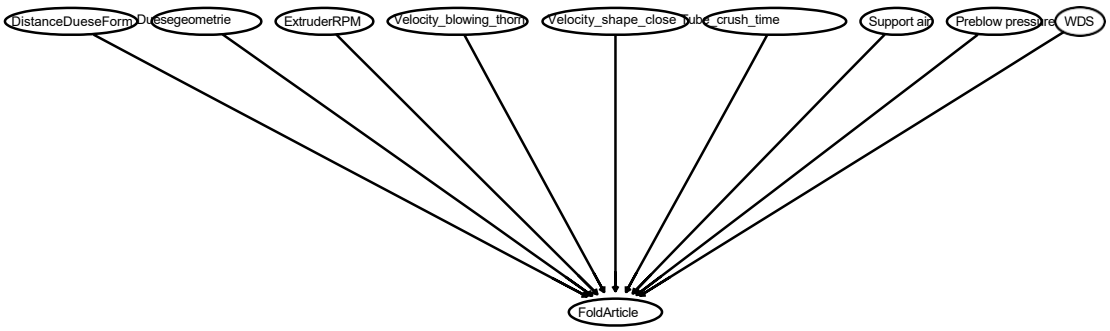


Figure 7.6.: Fault pattern *folds in the article* from main Bayes net "blow-molded part" with nine edges on the fault nodes

The knowledge used to create the structures is based on experience and impressions, a combination with correlations from data would supposedly improve the result. The error patterns often show a high number of edges on the error nodes and only few or no edges between cause nodes (see Figure 7.6). Defining dependencies between cause nodes and specifying the correct direction of the edge is in many cases beyond expert knowledge. The large number of edges pointing to a node reduces the result accuracy of the system. The more edges pointing to a node, the larger the amount of data needed to correctly estimate distributions. Each node has a CPT that must be filled in for a network to be complete (see Table 6.5). Combinatorics can be used to determine the number of possible combinations that can occur.

Calculation example: *Two nodes with two states each point to a node with two states; calculation of possible combinations* $n = 2^3 = 8$. *In this case, the CPT has eight entries. If another node with three states is added, the number of combinations increases from* $n = 8$ *to* $n = 8 \cdot 3 = 24$.

General formula for calculating the combinations:

$$n = 2^{\text{Number of nodes with 2 states}} \cdot 3^{\text{Number of nodes with 3 states}} \cdot \ldots = \prod_{i=0}^{|y} i^{Number\ of\ \text{nodes with i states}}$$

The goal should be to reduce or minimize the number of edges overall and specifically on a node. This keeps CPT's small, reduces the need for data and increases the accuracy of the results. One option to reduce edges to one node is to introduce so-called "intermediate" nodes.

[2] *arc.strength(x, data, criterion = NULL, ...):* Measure the strength of probabilistic relations expressed by the arcs of a Bayesian network. [35]

Nodes". In the area of the heating zones, these have already been implemented (see Fig. 7.7). Heaters that are assigned to an area, e.g. the head or extruder, are routed to an intermediate node, from where "one" edge continues to the error node. In the example shown, the number of edges could be reduced from four to one. Further possibilities would be a grouping of connected nodes (e.g. speeds, function tests).

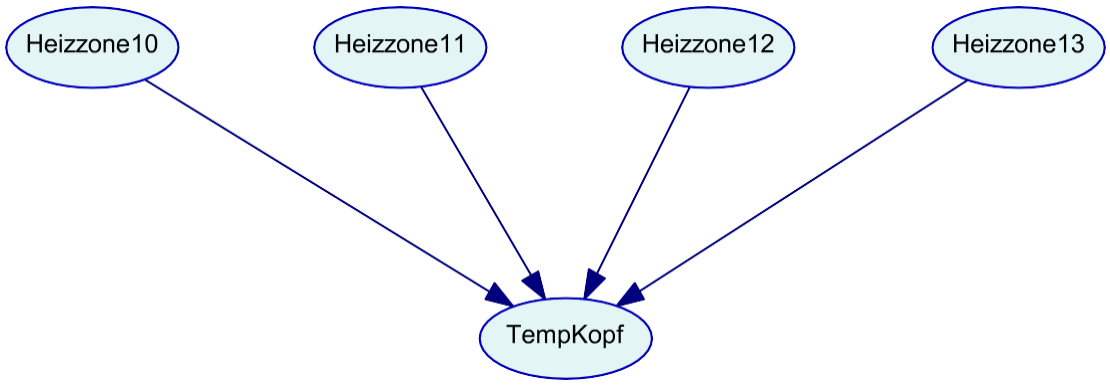


Figure 7.7: Introduction of intermediate nodes using the example of heating zones for machine component head

The use of intermediate nodes reduces the number of edges, but it requires a new logic to assess the states of such a node, since this must be derived from the parent nodes, and information is also lost by "merging". An error node receives the statement that the temperature at the head may be out of order, but in doing so it does not receive any information about the heating zone. Implemented algorithms would have to be adapted in order to generate further information from the intermediate node in a next step.

Are there possibilities/options to optimize the existing assistance system? One option was discussed earlier, the reduction of edges to one node by inserting intermediate nodes. Another possibility to reduce edges is the removal of nodes with a very low probability of occurrence, whereby results can be distorted and information is lost. The previously mentioned large proportion of node states that cannot be automatically detected also contributes to a reduction in the quality of the results. Here it is important to reduce the proportion of such states to a minimum (Section 8.3). The general lack of large and high-quality data pools [53] also ensures in this work that the subject of
"Generation of synthetic datasets" was employed.
The generation of synthetic data without consideration of distributions and dependencies of the nodes creates a data set which has only a very small relation to reality. If a Bayesian network is learned with this data, predictions of the assistance system are not meaningful. Due to the random creation, no dependencies can be determined within the data. An example is the determination of the most influential error cause for the error pattern "article hangs in shape" (see Figure 7.8). The predicted probabilities span an interval of 26.33% - 27.04% in the calculation. Whereas in the probability of occurrence clear

results are possible, these are not possible when determining the influence on a fault.

causeName Hose	causeState	probability
running direction	niO	0.2704
Ejector time	high	0.2685
Deaeration time	low	0.2677
Blowing time	high	0.2675
Deforming aid	niO	0.2675
BeschaedigungForm	Yes	0.2668
Blowing time	low	0.2658
Form_Sandstrahlung	niO	0.2656
Deaeration time	high	0.2653
Ejector time	low	0.2633

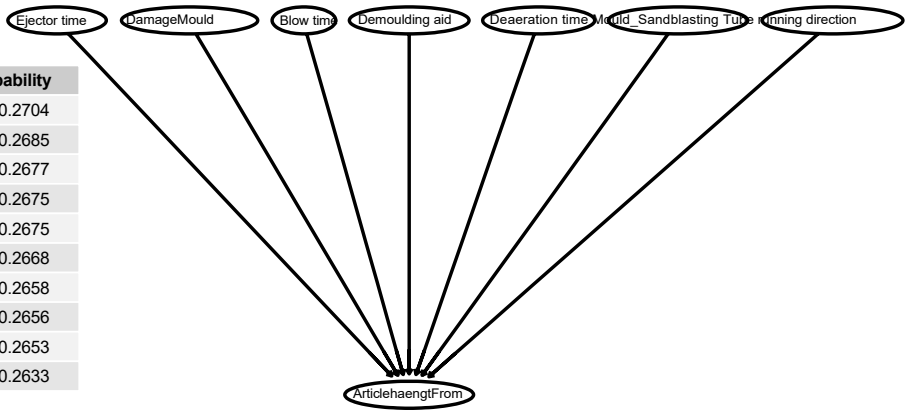


Figure 7.8: Determination of the most influential cause of the "article hangs in shape" fault pattern, fault pattern trained with random synthetic data

The generation of synthetic data based on rules takes distributions and dependencies into account, although not every dependency is considered there either. The effort for the generation is much more complex compared to random and it requires higher knowledge or data from the real process. The implementation according to subsection 6.2.3 using the function *rbn()* needs a learned Bayes net. In this work the CPT's were filled in manually for a partial net (defect images: scratches, tube run straight and preform shiny). For the whole mesh the effort is too high and it would not be possible to enter realistic data, because too detailed knowledge would be needed. Distributions of nodes could only be roughly estimated. A complete differentiation of the individual combinations is only possible to a limited extent for a human. An optional variant, which could be implemented by humans under certain circumstances, would be the specification of tendencies. This option, since it would also be performed manually, requires a large amount of time. However, these data enable the assistance system to achieve meaningful results.

Due to the currently high proportion of node states that cannot be detected automatically and the lack of a complete image of the current machine state, the determination of causes continues to take place "offline". Decisions and predictions are made on the basis of historical data with which the Bayesian network was trained, and only partial knowledge from the current process state is integrated. The approach to root cause determination with complete knowledge of a machine state opens up new possibilities. The implemented algorithms are only designed for "offline" analysis with additional partial knowledge. The analysis approach is specially designed and requires extensive analysis in real-world applications to identify its strength.
Statements about the stability of the user interface as well as the susceptibility to errors during operation cannot be completely substantiated with evidence at this point in time. Tests and abort behavior were carried out in parallel during development, but it is possible that rare errors occur during use, which must be corrected.

8. Summary and outlook

In this thesis a general concept for an assistance system was developed, which supports an operator and influences the decision making process. For the interaction between the system there is a graphical user interface. The concept was exemplarily implemented using the example of a knowledge repository for the setup process of an extrusion blow molding machine. In addition to completing the structure for the "preform" phase, a second one was created for the "blow molding" phase.

Possibilities of a dynamic adaptation of the system should be investigated. These were implemented in the algorithms for the root cause analysis. Here, existing knowledge from the current process state is taken into account in inference calculations. Dynamic Bayes nets were not used. Two approaches for the determination of the fault cause were developed and implemented. On the one hand, it is possible to predict the most influential and on the other hand the most probable cause of the fault. In the later evaluation the old and the newly implemented algorithms are compared with a qualitative evaluation. Here, the algorithm for the most probable cause of error succeeds in obtaining the most points.

For data acquisition, the system requires a connection to the machine, and various approaches are explained that enable access to Siemens controllers. Conceptually, the approach is implemented using "snap7". If the assistance system is in the machine network, it can access the controller via Python and "snap7" and query current parameter states. The problem of the lack of real data, which is frequently encountered in the AI world (also in this work), was taken as an opportunity to investigate possibilities of data stock expansion and complete generation of synthetic data. Random synthetic data without consideration of distributions and dependencies and rule-based synthetic data with consideration of distributions and dependencies were simulated. With random data the reference to reality is lost, thus they serve for function proofs, however, they cannot replace real process data. Parallel to the conception and realization of the algorithms and functions, a user interface for the operator has been designed and implemented. Via several tabs, it offers the setter the possibility to interact with the assistance system and to receive graphical outputs.

Apart from functional verification and a qualitative evaluation, no further tests were carried out. The realization and evaluation took place on the basis of rule-based synthetic data, which can reflect reality. After an implementation on site, a test under real process conditions has to be carried out as well as long-term tests for further evaluations. The optimization of such an assistance system will continue in the future. Thus

In the following sections, open points and possible potential for improvement, which provide stability and increase the quality of results, are described.

8.1. Use of a dynamic Bayes net

Representations often require abstraction of complex operations and processes in order to perform mathematical computations in the underlying state space [44]. Subsection 2.2.2 describes DBN's in general and Section 3.2 presents their current use. However, they will not be discussed further in the rest of the paper. In a DBN, the temporal evolution of a node is modeled. However, dynamic adaptation does not describe changes in the network structure over time, but rather changes in the distributions of individual nodes [44]. A so-called adaptation to the environment takes place under the assumption and inclusion of new measured values.

It is to be examined to what extent the use of a dynamic Bayes net brings advantages for this process. After a possible implementation, the results achieved by means of a static and dynamic Bayesian network are to be compared (criteria: Hit rate, number of causes until error correction, memory requirements, computing time, usability, etc.). Disadvantages of a DBN are the large memory requirement as well as the computing power and required computing time. By appending new time slices, the current environment / process image is represented. The process of determining the cause of the error and correcting the error cyclically records new measured values (prerequisite 100% automatically recordable data). New time slices are created with these. To a certain extent, the creation of new time series replaces the retraining in static Bayesian networks. The blow molding process used as an example can be regarded as static despite thermal processes. In dynamic processes such as milling processes or driving maneuvers of a car, where distributions of nodes can change strongly over time, the use of DBN's is reasonable and already widespread [27].
The structure and construction is more complex compared to a static Bayes net. In addition, the **R** package "bnlearn" supports discrete and continuous nodes, but only static Bayes nets and no dynamic ones. For modeling purposes, the package "dbnR" [1], which is based on "bnlearn", can be used. This is an **R** package created by *Quesada* for creating and using DBNs.

8.2. Working with real data

The present work was designed without having real data from the process. Only synthetic data were used for tests on the assistance system and implementation of functions and algorithms. Functional proofs could thus be proven, but no statements could be made about the quality of the results.
With the recording of real machine data and general data acquisition, new

[1] **R package** "dbnR": *https://cran.* r-project. org/web/packages/dbnR/dbnR. pdf

tests and analyses are possible. After a certain amount of real data has been collected, it is necessary to train the Bayesian network structures iteratively and to perform tests of the assistance system in a real process environment. In these tests it is possible to make statements about the quality of the predictions. By working with real data, an ideal adaptation to the working environment can be generated. Assistance system and process/machine are in perfect synthesis. The quality of the predicted error causes is improved by the use of real data.
A prerequisite for working with real data is a fully constructed and implemented machine connection. Section 5.2 presents general solution approaches for machine connection. A first conceptual implementation is described in subsection 6.2.1. This must be converted to a "long-term solution" and set up. In order for the assistance system to support the setter in his work, it needs to be synthesized to the real environment. For optimization a complete data acquisition is needed. For this purpose, it is advisable to increase the degree of automation of the data collection.

8.3. Increase the degree of automation of data acquisition

Achieved results of an AI are strongly dependent on the quality of the available and used data. In the thesis it is pointed out that the percentage of automatically detectable parameters for the nodes of the Bayesian network structures is below 50%. In order to increase the quality of the results it is mentioned in chapter 7 that the percentage of automatically detectable parameters has to be increased to almost 100%. At the current time, in addition to the automatic acquisition of parameters, parameters/node states are manually queried and entered. This reduces the speed of data acquisition and increases the risk of error, while at the same time increasing the effort required to collect data.
Possibilities to increase the degree of automation would be to implement additional sensors or camera technology. Sensors for moisture measurement could be used to determine the *material moisture*. By means of a digital scale, connected to the machine, it would be possible to automatically record the condition of the knot *article weight*. Intelligent camera technology could be used to detect *damage to the nozzle*, *burnt raw material on the nozzle surface* or the quality of the preform or blow moulded part. Automatic detection not only minimizes the risk of error, but also increases the speed of data acquisition. It is possible to collect a larger amount of data in a short time.

8.4. Algorithm for determining the cause of errors

8.4.1. Algorithm for determining the most influential cause of error

The algorithm used by the assistance system is selected manually by the operator (by default, the *most probable* cause is determined, see Figure 6.6(1)). If the automation level is 100%, as described in the previous section, it is advantageous to select the algorithm for the *most influential* cause of the error (the most influential real cause of the error is determined).

The cause is determined and identified → *the* fault is rectified as quickly as possible). Before this, it is necessary to adapt this in that after the determination, a comparison with the current process state takes place and it is checked whether the cause is located in this state as predicted, otherwise the determination would have to be repeated on the basis of the newly acquired knowledge. An example is shown in Figure 8.1. In the case of a suggested cause of error "Damaged nozzle", an automatic comparison with the real process takes place. If a "damaged nozzle" is actually present, the operator is informed of this, otherwise a new calculation takes place.

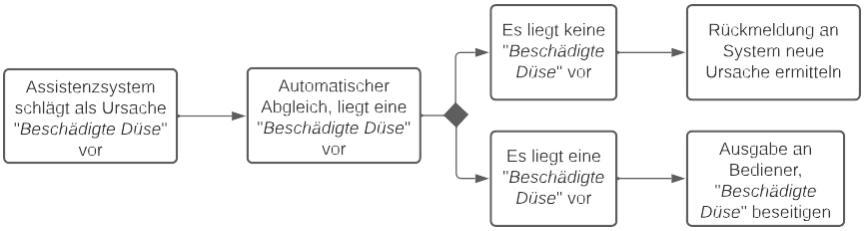


Figure 8.1: Example of cause determination algorithm most influential including automatic state adjustment of the real process

The adaptation makes it possible to determine the actual *most influential* cause of the error and to minimize the probability of error occurrence. The general process would be as follows, see Figure 8.2: The steps between "Input error pattern" (select error pattern) and "Fix error" (fix error based on predicted cause) would be automatic.

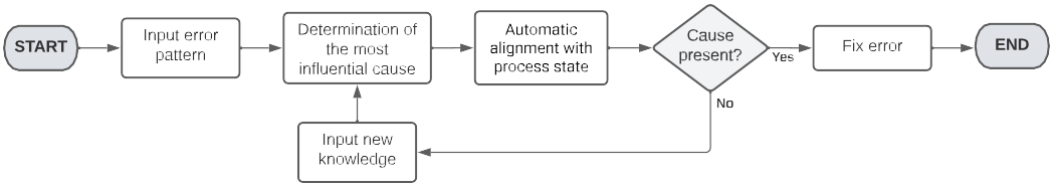


Figure 8.2: Flowchart algorithm for determining the most influential cause of error, 100% automatic data collection

8.4.2. Combination of algorithms most probable or most influential cause of error

If the degree of automation of the data collection cannot be increased sufficiently, it is possible to combine the two existing algorithms for fault cause detection. A possible combination could be designed as shown in Figure 8.3. At the beginning, the most influential cause of the fault is determined, then it is checked whether the predicted cause can be automatically matched with the current process state. If this is possible, an alignment takes place. If the cause is as predicted, it is output to the setter, who can intervene in the process and rectify the fault. If the cause is not as predicted, the most influential cause is determined again with the newly acquired knowledge. If the automatic process adjustment cannot be carried out, the probability of occurrence of this cause is calculated under known knowledge and then weighted. If it is

above a defined limit value, it is output to the setter. If it is below the limit value, it is not output and a new most influential error cause is determined with the help of the new knowledge.

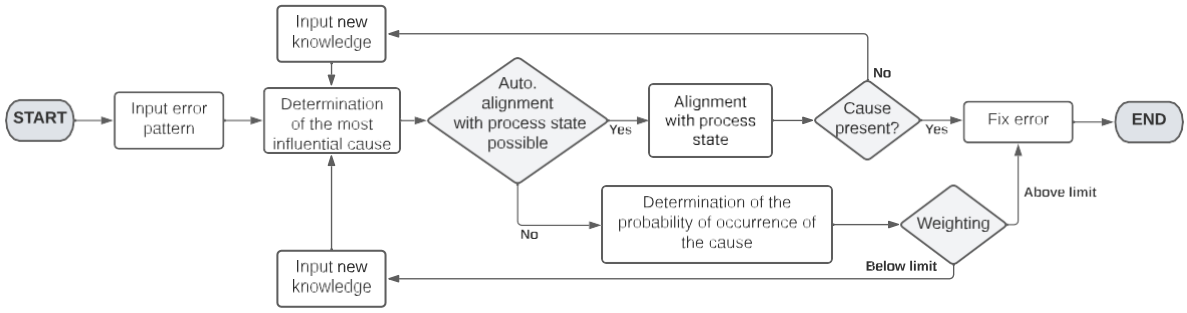


Figure 8.3: Flow chart of the combined algorithms for determining the cause of the error (most influential and most probable)

8.5. Automatic retraining

The environment in which the assistance system is located changes over time. In order to adaptively adjust to these changes, the network parameters need to be retrained. This topic has already been mentioned in section 2.3, but has not been further elaborated in this thesis. The learning and retraining of the network parameters is performed manually at the current state. As a next step to improve the system, an integration for automatic retraining is conceivable. Two different concepts are addressed.

8.5.1. Cyclic retraining

The cyclic and automatic retraining of the Bayes net after a new data set has been acquired is to be tested. During the process of root cause determination and error correction, new data is captured in a so-called observation during each iterative loop pass. Before this newly acquired data set can be used for re-training, data preparation (e.g., classification of numerical parameters) is required. After this, it is possible to use the data set to train the network and then make the new network available to the system again. Figure 8.4 shows the integration of cyclic re-training into the process of fault location.

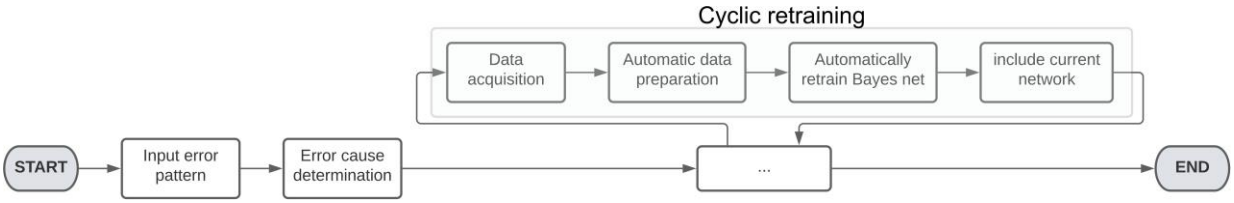


Figure 8.4: Flowchart of cyclic retraining integrated into root cause identification process

Post-training with only one observation will be based on the change of the network parameters

have only a marginal to undetectable influence. However, this process increases the risk to learn the network with incorrect data and to degrade the result quality of the predictions. For this reason, additional monitoring of data quality and checking for correctness is necessary.

8.5.2. Automatic training when a certain amount of data is reached

Another variant of how a Bayesian network independently adapts to the environment is automatic training after a certain number of data sets have been collected. Here, too, real data must first be automatically prepared for training. After reaching a predefined number of data sets, automatic learning then takes place. Because a larger amount of data is used for training, the effects on the parameters become more visible. However, here too, special focus must be placed on checking and monitoring the data quality. If incorrect data is used, a network may be trained incorrectly.
Automatic adaptive adjustment provides the opportunity to constantly adapt to the new environment and continue to achieve "high level" results, but it also provides the risk of deterioration through learning. New literature is currently focusing more on continuous learning. Not only are methods developed in which parameters are retrained, but in [29] *Silva* evaluates various solutions for continuous structure adaptation.

8.6. Introduction quality control

In order to increase the quality of results of the assistance system, a quality control would be conceivable. The first step would be a benchmarking. Here, it is examined how many causes are suggested until the error pattern is actually eliminated. With the help of newly received data sets from the process, the assistance system is retrained. It is expected that in the course of time the number of suggested causes will approach one (the first suggested cause corrects the error pattern =ideal case).
The data is collected via a benchmarking system running in the background and can be displayed.
can be set. Besides changes during the learning process, it is also possible to compare the two implemented algorithms (most likely and most influential). Through benchmarking, statements about efficiency can be made. Figure C.2 shows a fictitiously created benchmark diagram. The diagram shows the development of three real error patterns (number *of proposed causes, number of training runs*). As the number of training runs increases, the number of causes predicted decreases until the error is corrected.

Appendix

A. Bayes nets

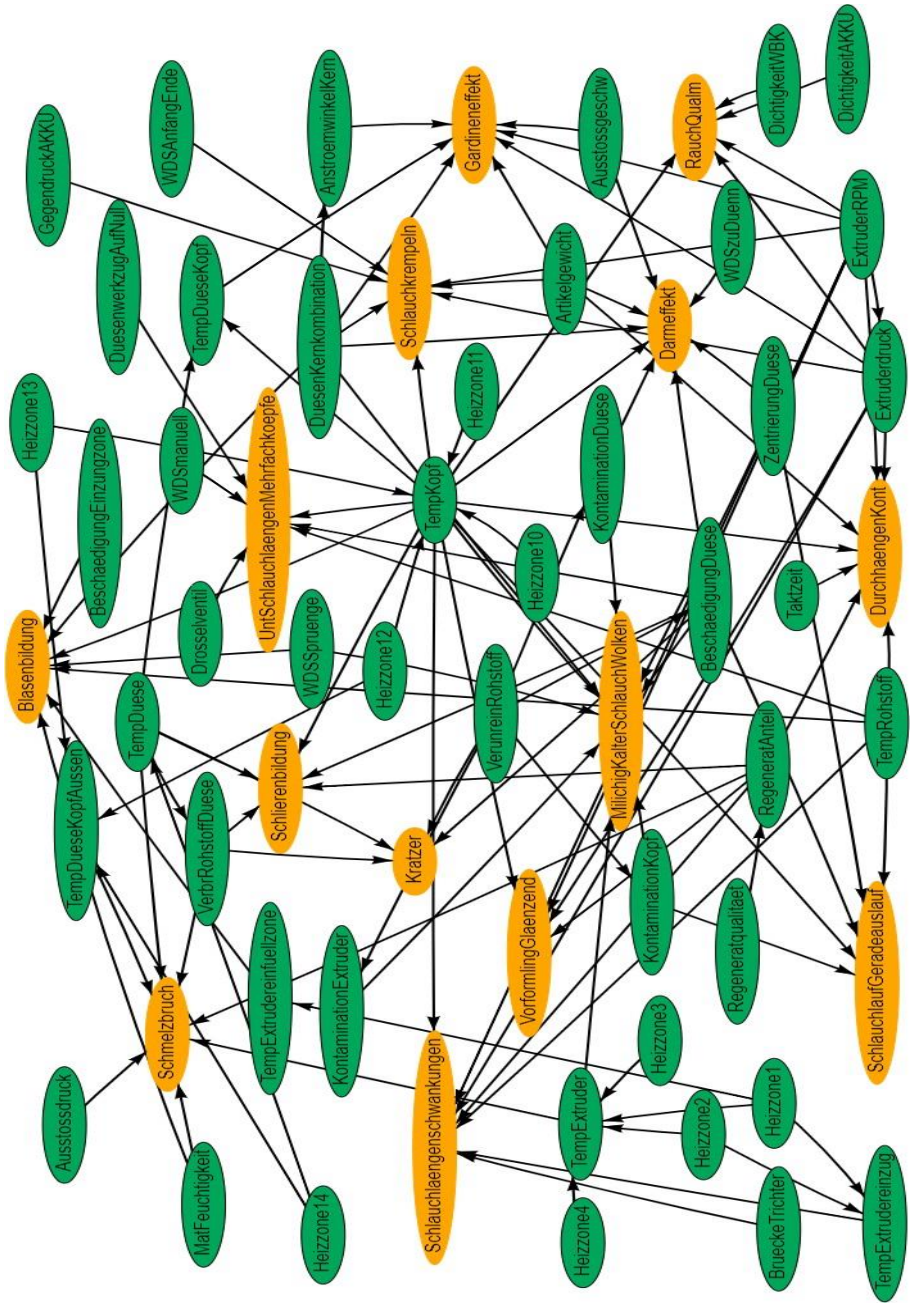


Figure A.1.: Bayes net structure for "preform" phase

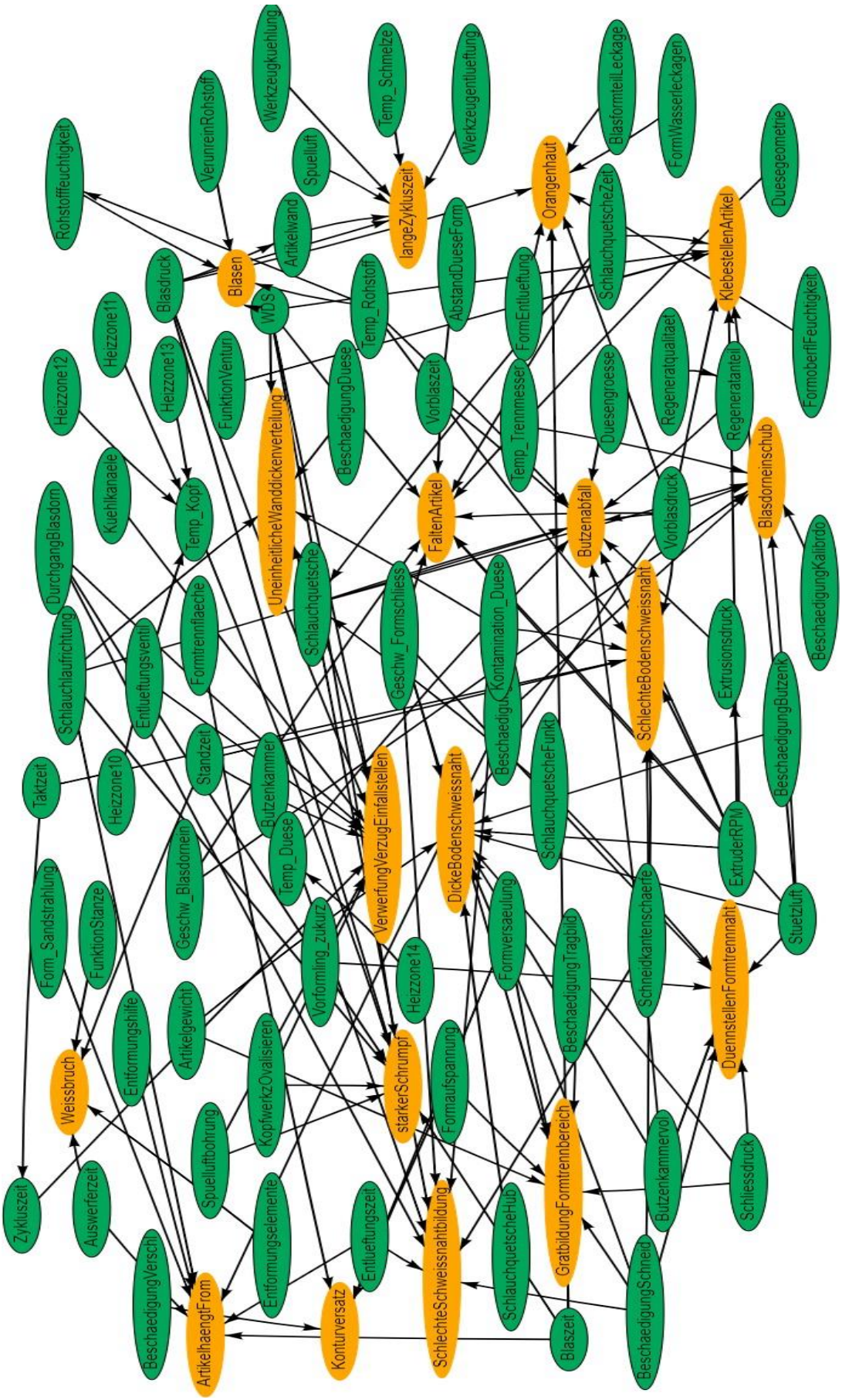


Figure A.2.: Bayesian network structure for the "blow-moulded part" phase

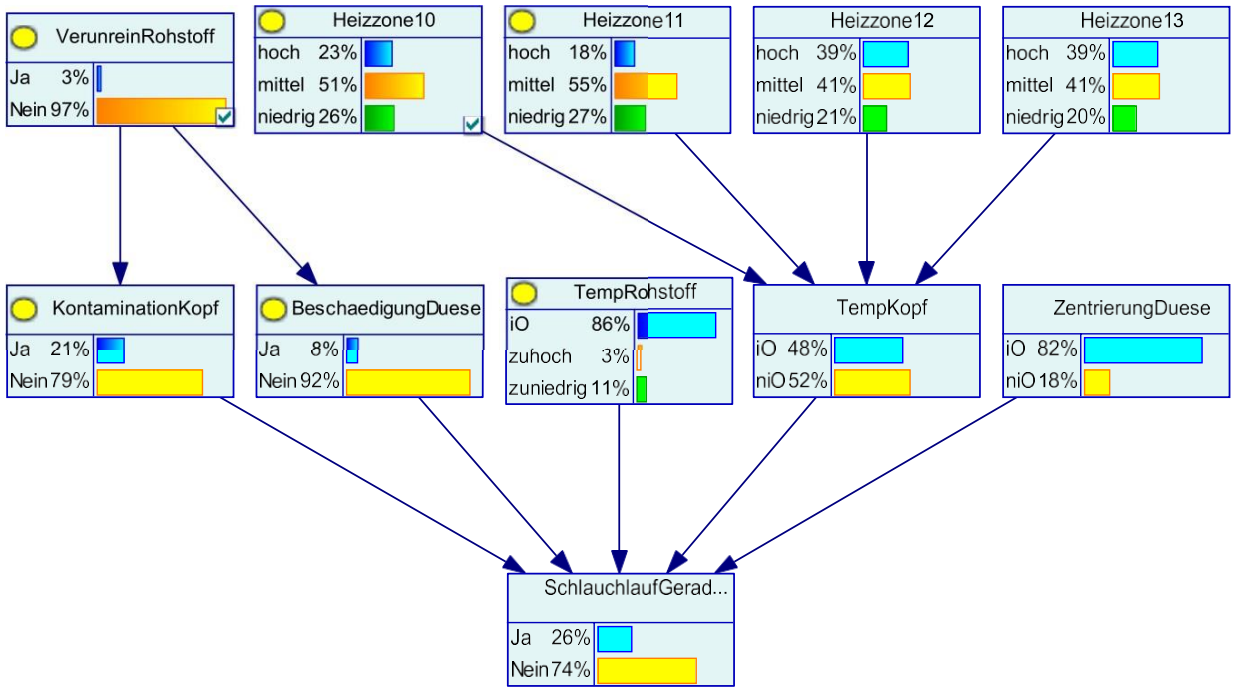


Figure A.3.: Manually learned Bayesian network, section of error image "Hose run-straight run", bar chart representation using GeNIe software

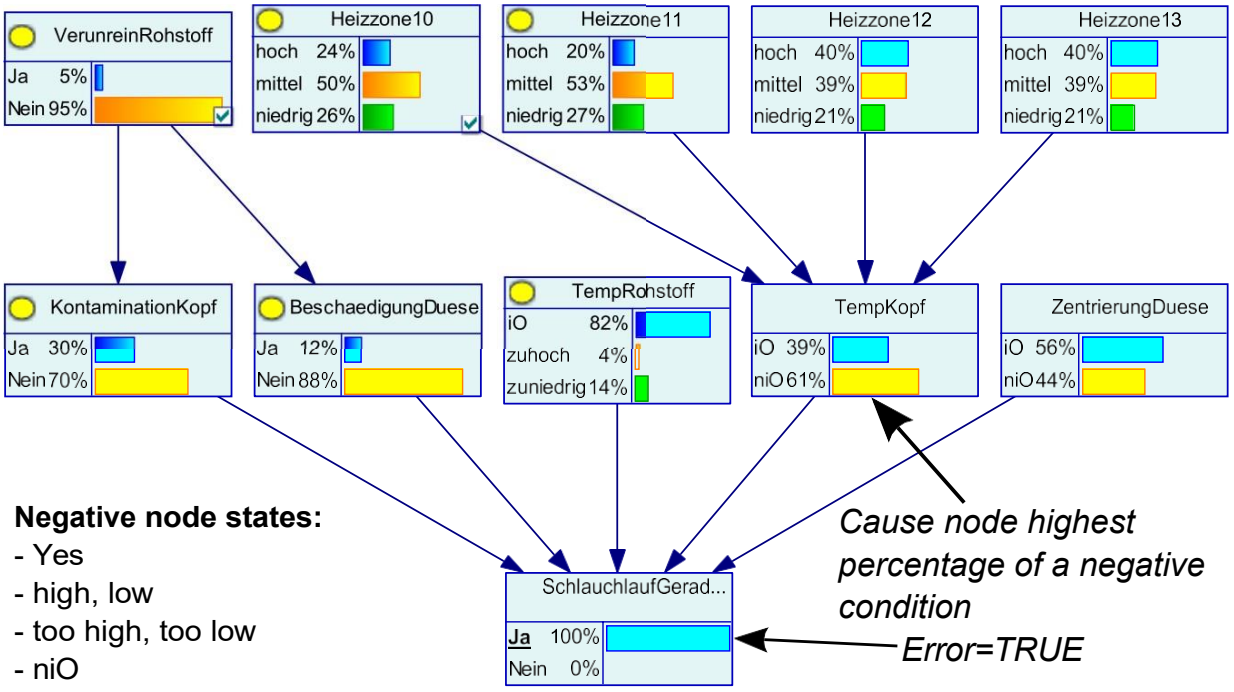


Figure A.4.: Manually learned Bayes net with occurred error, section error image "Hose runstraight runout", bar chart display using GeNIe software

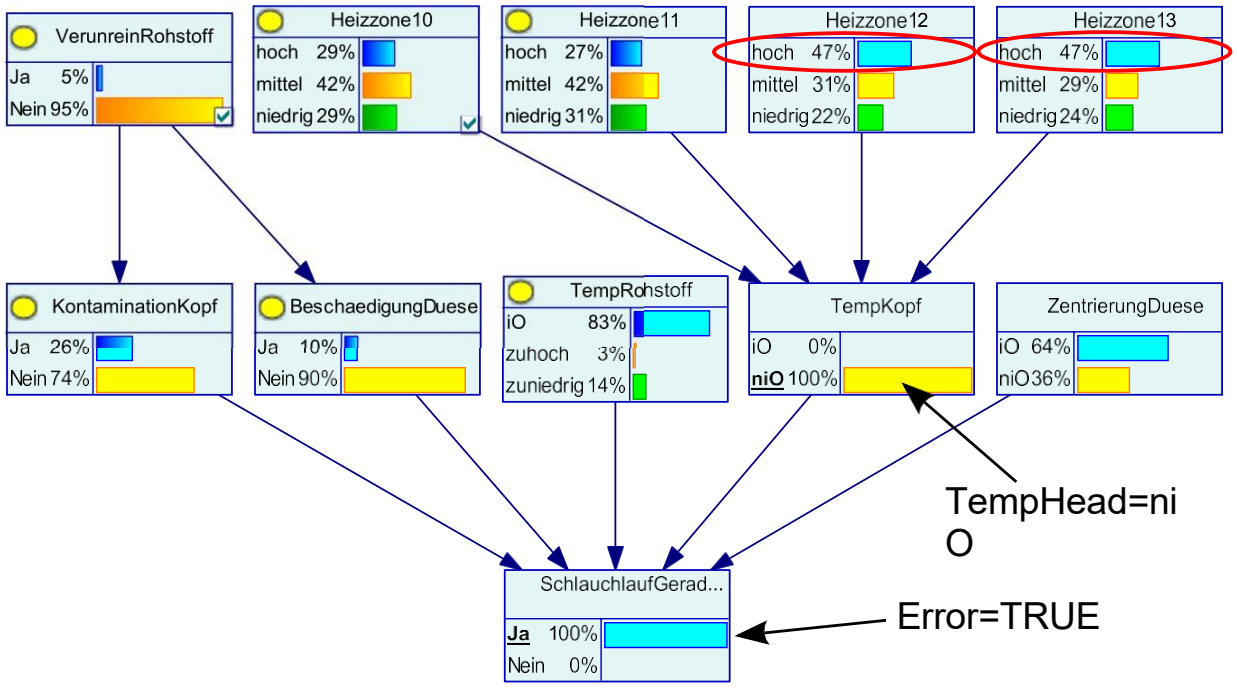


Figure A.5.: Manually learned Bayes net with occurred error and node "Temp- head" in "niO", section error image "Hose run-straight run-out", bar chart representation by means of software GeNIe

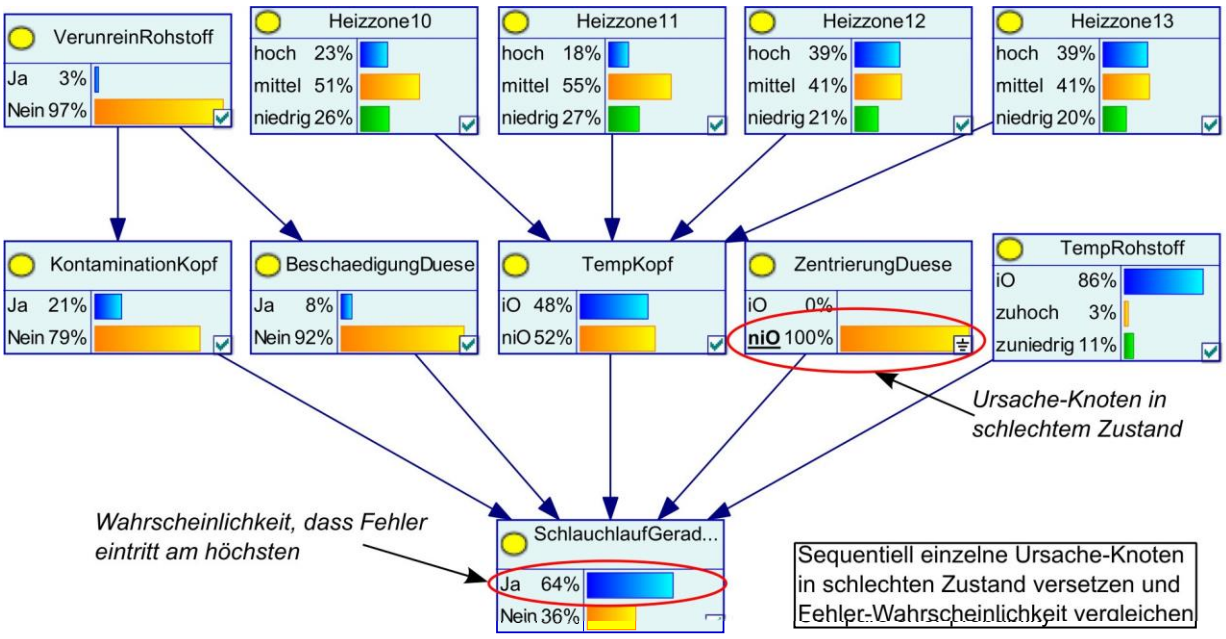


Figure A.6.: Manually learned Bayes net highest probability for occurrence of the fault under "CenteringDuese" in "niO", section fault pattern "Hose running- Straight running", bar chart representation by means of software GeNIe

B. Source code

```
classify Into 3 <- function ( data ){
  # find min value
  min <- min ( data ) #
  find max value max <-
  max( data ) # data
  width
  width <- ( max - min ) #
  lower limit
  lower <- ( min + ( width * 0.1)) #
  upper limit
  upper <- ( max - ( width * 0.1)) #
  initialize new data set
  newdata <- data
  # classify
  for ( i in 1: nrow ( data )) {
    if (( data [i ,1] >= min ) && ( data [i ,1] < lower )){ newdata [i
      ,1] <- " low "}
    else if(( data [i ,1] >= lower ) && ( data [i ,1] < upper )){
      newdata [i ,1] <- " middle "}
    else if(( data [i ,1] >= upper ) && ( data [i ,1] <= max )){
      newdata [i ,1] <- " high "}
    else
      cat(" Date not assignable: ", data [i ,1])
  }
  return ( newdata )
}

# classify data set Temp Duese Temp Duese
_classified <- classify Into 3 ( Temp Duese )
```

Source code B.1: Function to classify numerical data into three classes *low*, *medium* and *high*

```
## create multiple model strings from excel list and save as txt - file
## hand over: prepared excel list ( customize Excellist ()) with structure matrix , file / net name and name
     of the first error
## return : none , save txt - file at WD , as table error name and modelstring
excel2 ms <- function ( excellist , netname , firstfault ){
  # column number of first error
  k <- which ( colnames ( excellist) == firstfault )
  # minus (k -1) because first error at column k
  mslist <- matrix ( nrow = 1 , ncol = ( ncol ( excellist) -(k -1))) #
  create model string and save at array
  for ( i in k: ncol ( excellist)) {
    mstring <- create MStrings ( excellist , i)
    mslist [i -( k -1)] <- mstring        }
  # names columns with error names
  colnames ( mslist) <- colnames ( excellist[ k: ncol ( excellist)])
```

```
  # save model strings as < netname >. txt
  write . table ( mslist , file = paste0 (" MS", netname , ". txt", sep ="" ))
}
## create DAG from model string file
## hand over: model string file with multiple model strings
##              i. e. read . table (' MSVorformling . txt ')
## return : DAG
create DAG <- function ( msfile ){
  msstring <- ""
  # extract to a single string for ( i in
  1: length ( msfile )) {
    msstring <- paste0 ( msstring , msfile [ i], sep = "")
  }
  # convert model string to DAG
  msstring <- collapsModelstring
  ( msstring ) dag <- collapsed Ms2 DAG
  ( msstring ) return ( dag)
}
```

Source code B.2: Function *excel2ms()* and *createDAG()* to create a DAG from existing Excel table

```
## Generator for random synthetic data set for one error net
## hand over: excel list ( customize Excellist ()), error name and length of data set ## return :
data set as data frame with factors
simulate Rand Dataset <- function ( paralist , errorname , n){
  # create DAG from error with modelstring from ' paralist '
  modelstring <- create MStrings ( paralist , errorname )
  dag <- model2 network (
  modelstring ) # select all
  nodes
  nodes <- data . frame ( matrix ( ncol = length ( nodes( dag)), nrow = 3))
  colnames( nodes) <- nodes( dag)
  # initialize data
  data <- data . frame ( matrix ( ncol = length ( nodes( dag)), nrow = n)) colnames(
  data ) <- nodes( dag)
  # simulate random data n observation of each node for ( i in1:
  length ( nodes)) {
    # extract node state if(
    colnames( nodes)[ i] ==
    errorname )
      nodes[, i] <- c(" No "," Yes","-")
    else {
      index <- match ( colnames( nodes)[ i], paralist[,"
      netname "]) nodes [1 , i] <- paralist[ index ," state
      _default"] nodes [2 , i] <- paralist[ index ," state _2 "]
      nodes [3 , i] <- paralist[ index ," state _3 "]
    }
    # generate a random data set of one node prob <-
    runif (1 , min = 0.1, max = 0.9) if( nodes [3 , i]
    ! = "-")
      rand <- sample (1:3, n, replace =TRUE , prob =c( prob ,(1- prob )* 0.3,(1- prob )* 0.7))
    else
      rand <- sample (1:2, n, replace =TRUE , prob =c( prob ,(1- prob ))) data
    [, i] <- as. factor( nodes[ rand , i])}
  return ( data )
}
```

Source code B.3: Function *simulateRandDataset()* to generate random synthetic data for an error subnet

```
## Generator for random synthetic data set for the hole network
## hand over: excel list (customize Excellist ()), length of data set, first error ## i. e. "
scratch" or "slug waste "
## return : data set as data frame with factors simulate Rand
DatasetBig <- function (paralist, n, firsterror){
  # set indexes
  firsterr <- match (firsterror, colnames(
  paralist)) lasterr     <- length (colnames(
  paralist))
  # extract all node names
  names <- c( colnames(paralist)[ firsterr: lasterr], paralist[,"
  netname "]) nodes <- data . frame ( matrix ( ncol = length ( names), nrow = 3,
  dimnames = list( c( NULL ),
  names))) # initialize data set
  data <- data . frame ( matrix ( ncol = length ( names), nrow = n, dimnames =
  list( c( NULL ), names)))
  # simulate random data n observation of each node for ( i in1:
  length ( nodes)) {
    # extract node state if(! is. na( match ( colnames(
    nodes)[ i], colnames( paralist))))
      nodes[, i] <- c(" No "," Yes","-")
    else {
      index <- match ( colnames( nodes)[ i], paralist[,"
      netname "]) nodes [1, i] <- paralist[ index ," state
      _default"] nodes [2, i] <- paralist[ index ," state _2 "]
      nodes [3, i] <- paralist[ index ," state _3 "]
    }
    # generate a random data set of one node prob <-
    runif (1, min = 0.1, max = 0.9) if( nodes [3, i]
    ! = "-")
      rand <- sample (1:3, n, replace =TRUE , prob =c( prob ,(1- prob )* 0.3,(1- prob )* 0.7))
    else
      rand <- sample (1:2, n, replace =TRUE , prob =c( prob ,(1- prob ))) data
    [, i] <- as. factor( nodes[ rand, i])
  }
  return ( data )
}
```

Source code B.4: Function *simulateRandDatasetBig()* to generate random synthetic data for an entire Bayes net

```
# Import packages
import snap7
from snap7 import util
import pandas as pd
from datetime import datetime


def read_data ():      # define read data function # get

data block
# get data from DB201 byte 0 to 15 data =
client. db_read (201, 0, 16)
# get all other required data blocks
...
...
...


```

```
# get data from data block # DB201
. DBB0
Status_x = util. get_bool( data , 0
, 0) # DB201 . DBW2
Temp_Z 12 = util. get_int( data ,
2) # DB201 . DBD10
Taktzeit = util. get_real( data , 10)
# get all other required data from block
---
---
---

# create data set
rowdata = {' date': [ datetime . now (). strftime ("%Y-%m-%d %H:% M:%S")],
        'Status_x ':  [Status_x ],
        'Temp_Z 12 ':  [Temp_Z 12],
        'Takt time  [cycle time ],
        ':
        # all others  parameters
        ---: []
}

# convert to data frame
dataset_func = pd. Data Frame ( data = rowdata )

return dataset_func


if name == '__main ':
# connect to plc from system
client = snap7 . client. Client ()
# plc information
ip = ' 192 . 168 . 30 . 1'      # example ip address
rack_num = 1                    # example rack number
slot_num = 1                    # example slot number
client. connect( ip , rack_num , slot_num )

# check connection
print(' Connected :', client. get_connected ())
print('                                                ')

# read current data from data block
dataset = read_data ()
print('---------------------------------- Read data                    ')
```

Source code B.5: Connection to the machine, concept code *python-snap7* Connection setup and data query

```
## fill missing value in the data set ## to
complete data set after a RCA
## hand over: data set with missing values ##
return : complete data set
filldataset <- function ( data ,
  excellist){ # number of rows and cols
  dim = dim ( data )
  # loop through each column for ( i
  in1: dim [2]) {
    # index where is a NA value temp <- which
    (! is. na( data [, i]))
```

```
    # when only found one NA if(
    length ( temp ) == 1){
      # search line index of parameter in parameter list
      index <- which ( colnames( data )[ i]== excellist[,"netname "])
      # when state found in the last observation , set all obs to this state if( temp ==
      dim [1]){
        data [, i] <- data [ temp , i]}
      # when state not found in the last observation and not default state else if( data
      [ temp , i] ! = excellist[ index ," state _default"]){
        # set observations to the found index to the found state data
        [1: temp , i] <- data [ temp , i]
        # set observations from found index to end on default state data [( temp
        +1): dim [1], i] <- excellist[ index ," state _default"]}
      else {
        # when state not found in the last observation but default state , all def data [, i] <-
        data [ temp , i]}}
  }
  return ( data )
}
```

Source code B.6: Function *filldataset()* to complete incomplete datasets by deriving states of manual nodes

```
## calculate the most probable cause , can consider known good cause condition
## probability : how likely it is that the cause will occur
## hand over: bayes net , one error name , list of known good cause conditions
## return : a decreasing probability list
## P( cause = TRUE | error= TRUE : cause _i= TRUE ...)
causeanalysis _likeliestdyn <- function ( bayesnet , error , kno Evi=NULL , kno State = NULL ){
  # Initialization
  error_lvl <- " Yes" # error = TRUE
  evidence <- paste ("(", error ," == '", error_lvl ,"'", sep ="") # evidence string
  wktlist <- data . frame ( matrix ( ncol = 3 , nrow = 0)) # initialize matrix
  colnames( wktlist) <- c(' cause Name ',' cause State ',' probability ') # names columns
  j <- 0 # set index counter for matrix

  nodelist <-       Influ Nodes ( bayesnet , error) # all cause nodes
  if (! is. null( kno Evi)) {
    # remove known causes from list
    nodelist <- nodelist [! nodelist% in% kno Evi]
    # create string for causes with known state
    knowledge <- known Causes ( knoEvi , kno State )
    # add to evidence string ( formula )
    evidence <- paste ( evidence , knowledge , sep ="")
  } # end if
  evidence <- paste ( evidence , ")", sep = "") # close evidence string

  for ( i in1: length ( nodelist)) { # for every cause node
    expression <- paste0 (" bayesnet [['", nodelist[ i], " ']][[' prob ']]")
    problist <- eval( parse ( text = expression )) # select node states
    statenames <- rownames( problist) # select state names
    for ( n in1: length ( statenames )) { # for every cause node state
      if ( statenames [ n]! =" No " & statenames [ n]! =" iO " & statenames [ n]! ="
      medium "){
        event <- paste ("(", nodelist[ i]," == '", statenames [ n]," ')", sep = "")
        cmd = paste (" cpquery ( bayesnet ,", event ,", evidence ,", n = 10 ^ 4 )", sep = "")
        prob <- eval( parse ( text = cmd )) # calculate inference
        prob <- round ( as. numeric( prob ) ,4)
        j <- j + 1 # increment index counter
```

```
36            wktlist[j,] <- c(nodelist[i], statenames[n], prob) # set to probability list
37          } # end if
38        } # end for
39      } # end for
40      # Sort list indescending order
41      wktlist <- wktlist[order(as.numeric(wktlist$probability), decreasing = TRUE),] return (wktlist)
42  }
43
```

Source code B.7: Function *causeanalysis_likeliestdyn()* to determine the most probable cause of an error

```
1  # #### TAB OVERVIEW #####
2  ## Detect if tab changed or button pressed to
3  Listen <- reactive ({
4    list(input$tabs,input$idi_ac_refresh BN)
5  })
6
7  ## draw bayes net interactive
8  observe Event (to Listen (),{
9    output$ido_bayesnetz_total_plot <- renderUI ({
10     isolate (dag <- bnfit2 DAG (renew BN (input$idi_
11     bayesnettype ))) nodelist <- as.list(leaf.nodes(dag))
12     nchighlight <- list(background = "yellow", border = "black") bnviewer ::
13     viewer(dag,
14       bayesian Network. width = "100%", bayesian Network.
15       height = "90vh", bayesian Network.layout = "
16       layout_with_dh", bayesian Network.title = "
17       Discrete Bayes Net -
18         Overall view", node
19       .shape = "ellipse",
20       node.colors = list(background = "#00a65a
21         ", border = "black",
22         highlight = nchighlight),
23       node.font = list(color = "black", face = "Arial", size
24         = 25),
25       edges.width = 3, options.
26       highlightNearest = TRUE,
27       clusters = list(
28         list(label = "error
29           ", shape = "icon",
30           icon = list(color = "orange"),
31           nodes = nodelist))
32     )
33   })
34 })
```

Source code B.8: Tab *Total Overview* Server Side Shiny

```
1  ## drop down menu to choose fault pattern
2  output$bayesnet_errorpattern <- renderUI ({
3    selectInput (inputId = "
4    bayesnet", label = "Select error
5    image",
6    choices = c(leaf.nodes(renew BN (input$ idi_bntype ))))
7  })
8
9  ## load correct excel list and bayes net observe
   Event (input$ idi_bntype,{
```

```
  # when ' preform ' was selected if(
  input$ idi_bntype == " preform "){
    # load parameter list
    matrix $ structure <- readxl :: read _xlsx (" parameter list . xlsm ",
    sheet = " error preform ")
    matrix $ structure <- customize Excellist ( matrix $ structure ) cat("
    parameter list for preform is loaded \n")}
  # when ' blow moulding ' was selected
  else if( input$ idi_bntype == " blow
    molding "){ # load parameter list
    matrix $ structure <- readxl :: read _xlsx (" parameter list . xlsm ",
    sheet = " error blow mold part ")
    matrix $ structure <- customize Excellist ( matrix $ structure ) cat("
    Parameter list for blow molding is loaded \n")}
  else { cat(" Error evaluating Bayes net type \n")}
})
```

Sourcecode B.9: Tab *Error Overview* Server Side Shiny, Adjustment Selection Error Category

```
## load current bayes net for the first launch
actual <- reactive Values ( bn = renew BN (" preform "))

## load select bayes net ' preform ' or ' blow mold
part ' observe Event ( input$ idi_bnart ,{
  if( input$ idi_bnart == " preform "){
    actual$bn <- renew BN ( input$
    idi_bnart)}
  else if( input$ idi_bnart == " blow molding
    "){ actual$bn <- renew BN ( input$
    idi_bnart)}
})

## learning was started observe
Event ( input$ idi_learn ,{
  # save data input as . csv write . csv2 ( data (),"  Data
  _archive / data _download . csv ") output$ ido
  _startlearn <- renderText ({
    # learn parameters
    actual$bn <- train BN ( bnfit2 DAG (
    actual$bn)) # save new bayes net
    isolate ( if( input$ idi_bnart == " preform "){ write .
      net(" actual_bn_ preform . net", actual$bn)
      " The learning was executed....."}
    else if( input$ idi_bnart == " blow molding "){
      write . net(" actual_bn_ blow molding . net",
      actual$bn) " The learning was executed."})
  })
  # hint after learning parameters show
  Modal( modalDialog (
  title = " New parameters",
  footer = modalButton (" Confirm "),
  h4 (' ATTENTION , the parameters have been
  relearned. '))) # disable button learn
  load Data Ready $ok <- FALSE
})
```

Source Code B.10: Tab *Parameter Learning* Server Side Shiny, Adjustment Selection Main Bayes Net

Figu

	Impurity	Raw material Damage	Extruder	RPM Extruder pressure	Heating zone10	Regenerate	TempHead Temp	Raw material Centering	Duese Tube run	Straight run Preform	Glaenzend
1	No	No	high	high	mediu	high	iO	increasin	iO	No	No
2	No	No	low	low	high	high	niO	iO	iO	No	No
3	No	No	low	mediu	mediu	high	iO	iO	iO	No	No
4	No	No	high	high	high	high	niO	too high	niO	Yes	No
5	No	No	low	mediu	mediu	high	iO	iO	iO	No	Yes
6	No	No	high	high	high	high	niO	too high	iO	No	No
7	Yes	Yes	high	low	low	high	niO	iO	iO	No	Yes
8	No	No	medium	mediu	high	medium	iO	iO	iO	No	No
9	No	No	high	high	mediu	low	niO	too high	iO	No	No
10	No	No	high	high	low	medium	niO	iO	niO	Yes	No
11	No	No	high	mediu	mediu	high	niO	iO	iO	No	Yes
12	No	No	low	low	mediu	high	niO	iO	iO	Yes	No
13	No	No	low	low	mediu	high	niO	iO	iO	No	Yes
14	No	No	high	high	mediu	high	iO	iO	iO	No	Yes
15	No	No	high	low	low	medium	niO	iO	iO	Yes	Yes
16	No	No	low	mediu	mediu	high	niO	iO	iO	No	No
17	No	No	low	low	low	high	iO	too high	iO	No	No
18	No	No	high	high	mediu	high	iO	iO	iO	No	No
19	No	No	high	high	high	high	niO	iO	iO	No	No
20	No	No	medium	mediu	mediu	medium	iO	increasin	iO	Yes	Yes
85	No	No	high	mediu	mediu	high	niO	iO	iO	Yes	No
86	No	No	high	high	mediu	high	niO	iO	iO	No	No
87	No	No	low	mediu	mediu	high	iO	iO	niO	No	No
88	No	No	low	low	low	low	iO	iO	iO	No	Yes
89	No	No	high	high	mediu	high	iO	iO	niO	No	No
90	No	No	medium	mediu	low	high	niO	iO	iO	No	No
91	No	No	low	low	mediu	medium	niO	iO	iO	No	No
92	No	No	medium	mediu	high	high	iO	iO	iO	No	No
93	Yes	Yes	medium	mediu	mediu	high	niO	iO	iO	Yes	No
94	No	No	low	low	high	high	iO	iO	iO	No	No
95	No	No	high	mediu	mediu	high	iO	iO	iO	No	No
96	No	No	low	high	mediu	high	iO	increasin	iO	No	No
97	No	No	medium	mediu	low	high	iO	iO	iO	No	No
98	No	No	low	low	low	high	iO	iO	iO	No	Yes
99	No	No	high	high	high	high	iO	iO	iO	No	No
100	No	No	low	low	low	medium	niO	iO	iO	Yes	Yes

Table C.1: Protocol function verification "fault cause determination

Action/ FunctionFound/	Result	Comment
Selection Type of analysis	OK	Choice between most likely and Most Influential Cause
Selection Main Bayes Net	.O.	Preform or blow molded part
Selection Defect imagei.	O.	
Start error cause determination	OK	
Output Error cause	OK	Graphical and textual output Opening
Feedback Error not corrected	OK	a pop-up window
Confirmation via pop-up window, if necessary, query button status	OK	Query only for manual parameters
New proposed cause	OK	Calculation taking into account the state of the node
Feedback Error correction	OK	Opening a pop-up window for
Query open node states		confirmation
confirm a deal	OK	Only from open manual nodes
	OK	

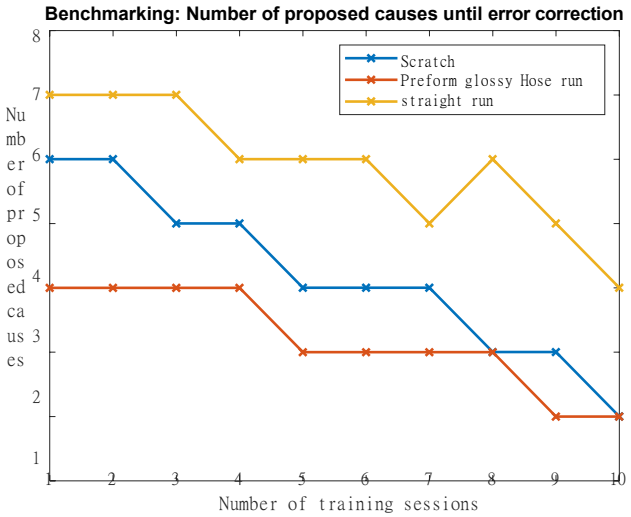


Figure C.2: Benchmarking example: Number of defect causes until defect elimination after number of training runs for the defect images scratch, preform shiny and tube run straight run

Literature

[1] Ropponen A. and Ritala R. "Production-line wide dynamic Bayesian network model for quality management in papermaking". In: *18th European Symposium on Computer Aided Process Engineering*. Edited by Brunswick B. and Joulia X. Vol. 25 Computer Aided Chemical Engineering. Elsevier, 2008, pp. 979-984. doi: https://doi.org/10.1016/S1570- 7946(08) 80169 - 1. url: https :/ / www . sciencedirect . com / science / article / pii / S1570794608801691.

[2] Truong-Duc-Duy A. and Tan P. V. "A Python Framework for Model-Based Design, Commission and Monitor the Thermal Process". In: *2021 International Conference on Electri- cal, Communication, and Computer Engineering (ICECCE)*. 2021, pp. 1-4. doi: 10.1109/ ICECCE52056.2021.9514203.

[3] Weiser A. "Probabilistic prediction of lane changes for highly automated driving on highways". Diss. Karlsruhe Institute of Technology (KIT), 2018. 236 p. doi: 10.5445/IR/1000087225.

[4] Zheng A., Casari A. and Lotze T. *Feature construction for machine learning: principles and techniques of data preparation*. Animals. O'Reilly, 2019. 214 p. isbn: 9783960102502. url: https://books.google.de/books?id=O5iSDwAAQBAJ.

[5] Brandherm B. "Embedded nth-order dynamic Bayesian networks". Diss. Ntw.- Faculty of Engineering I Saarland University, 2006. doi: http://dx.doi.org/10.22028/D291- 25859.

[6] Brandherm B. "Processing sensor data in dynamic bayesian networks". In: *9th GI Workshop Aedaptivity and User Modeling in Interactive Software Systems"*. Ed. by N. Henze. University of Dortmund, 2001, pp. 288-293.

[7] Liu B. "Lifelong machine learning: a paradigm for continuous learning". In: *Front. Comput. Sci.* 11.3 (June 2016), pp. 359-361. issn: 2095-2228. doi: 10.1007/s11704-016-6903-6. url: https://doi.org/10.1007/s11704-016-6903-6.

[8] Beeley C. *Hands-On Dashboard Development with Shiny. A practical guide to building effective web applications and dashboards*. Packt Publishing, 2018. 76 p. isbn: 9781789615623. url: https://books.google.de/books?id=YPFsDwAAQBAJ.

[9] Beeley C. and Sukhdeve S. R. *Web Application Development with R Using Shiny. Build stunning graphics and interactive data visualizations to deliver cutting-edge analytics*. Packt Publishing, 2018. 238 p. isbn: 9781788998284. url: https://books.google.de/books?id= 7dFwDwAAQBAJ.

[10] Piech C. and Sahami M. *Parameter Estimation*. May 2017. url: https://web.stanford.edu/class/archive/cs/cs109/cs109.1192/reader/11%20Parameter%20Estimation.pdf.

[11] Vetterli C. and Brand T. *Root-cause analysis*. url: https://leanhealth.ch/transformation/ what/concept.php?ID=7 (visited Jan. 11, 2022).

[12] Kasper D. *Detecting driving maneuvers with object-oriented Bayes nets in highway scenarios*. Tübingen, 2012. url: http://nbn-resolving.de/urn:nbn:de:bsz:21-opus- 68002.

[13] Kötke D. "Technical guide for extrusion blow molding machines. Specifically for technical blow molding equipment (TBA)". Bachelor thesis. Lüder/ Reinstorf: Ostfalie University of Applied Sciences Faculty of Mechanical Engineering, Institute of Constriction and Applied Mechanical Engineering, 2015.

[14] Nardella D. *Snap7*. Jan. 21, 2022. url: http://snap7.sourceforge.net/.

[15] Oehm D. *Simulating data with Bayesian networks*. 2019. url: http://gradientdescending.com/simulating-data-with-bayesian-networks/ (visited Jan 26, 2022).

[16] Lyapina E. *Open data sources for industrial AI. Applying AI to the manufacturing processes*. March 2, 2020. url: https://aithority.com/guest-authors/open-data-sources-for-ai-in-industry/ (visited Jan. 26, 2022). AiThority, AITechnology Insights.

[17] Fernandes E. C., Iuri L., Camatti J. A., Brown L. and Borsato M. "Flexible Production Data Generator for Manufacturing Companies". In *Procedia Manufacturing* 51 (2020). 30th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM2021), Pp. 1478-1484. issn: 2351-9789. doi: https://doi.org/10.1016/j.promfg.2020.10.205.
url: https://www.sciencedirect.com/science/article/pii/S2351978920320680.

[18] Arroyo-Figueroa G. and Sucar L. E. *Temporal Bayesian Network of Events for Diagnosis and Prediction in Dynamic Domains*. Mexico: Instituto de Investigaciones Electricas, 2005. ch. Applied Intelligence 23, pp. 77-86. url: https://link.springer.com/content/pdf/ 10.1007/s10489-005-3413-x.pdf.

[19] Molenaar G. and Preeker S. *python-snap7*. Jan 21, 2022. url: https://python-snap7.readthedocs.io/en/1.1/index.html.

[20] Weidl G., Rode M., Horch A., Shaw C. and Vollmer A. *Automatic root cause analysis of faults and malfunctions in rolling mills - an overview and practical examples*. Extract from Steel and Iron. 2004, S. 399-410.

[21] Wellenreuther G. and Zastrow D. *Automatisieren mit SPS: Theorie und Praxis*. Vieweg technical books of technology. Vieweg+Teubner Verlag, 2005. isbn: 9783834890184. url: https://books.google.com/books?id=v0ApBAAAQBAJ.

[22] Kleinings H. *What is Continuous Machine Learning?* Oct. 28, 2021. url: https://levity.ai/blog/what-is-continuous-machine-learning (visited Jan. 25, 2022).

[23] Wen H. *Bayes Parameter Estimation (BPE) tutorial*. Rhea. 2014. url: https :/ / www . projectrhea . org /rhea / index . php / Bayes _ Parameter _ Estimation# :~: text = Bayes % 20parameter % 20estimation% 20(BPE)%20s, random % 20variables% 20with % 20unknown % 20parameters . &text = Our % 20goal % 20is % 20to % 20compute , probability % 20density % 20function%20of%20X. (visited 01/28/2022).

[24] Resnizky H. G. *Learning Shiny. Make the most of R's dynamic capabilities and implement web applications with Shiny*. Packt Publishing, 2015. 246 p. isbn: 9781785281990. url: https://books.google.de/books?id=P%5C_1%5C_CwAAQBAJ.

[25] Kinzig J. *Reading process data from a SIMATIC S7-1200/ S7-1500 with OPC UA and Python (OPC UA Secure)*. May 15, 2021. url: https://siincos.en/index.php/en/ blog/control-technology/programming/31- readout- process-data- s7-1200- s7- 1500-opc-ua-and-python.

[26] Rüdenauer J. "Use of probabilistic methods for decision making in RoboCup". Di- ploma thesis. University of Stuttgart, Faculty of Computer Science, Electrical Engineering and Information Technology, Germany, Nov. 2003, p. 145. url: http://www2.informatik.uni-stuttgart.de/cgi- bin/NCSTRL/NCSTRL_view.pl?id=DIP-2103&engl=0.

[27] Gomer K., Lu H., Weidl G., Dang T., Breuel G., Schlechtriemen J. and Rosentiel W. "Clearance assessment for lane change maneuvers with Bayes nets". In: June 2015, p. 12. url: https :// www . researchgate . net / publication / 279886700 _ clearance_evaluation _ for _ lane_change_maneuvers_with_Bayes_nets.

[28] Murphy K. P. "Dynamic Bayesian Networks: Representation, Inference and Learning." Diss. Computer Science, University of California, Jan. 2002.

[29] Pereira L. A. P., Nunes J. B N., Perkusich M. B., Gorgonio K. C., Almeida H. O., and Perkusich A. "Continuous learning of the structure of Bayesian networks: a mapping study". In: Nov. 2018, p. 20. isbn: 978-1-83962-322-6. doi: 10.5772/intechopen.80064.

[30] van der Gaag L. C., Renooij S., Aleman C. Witteman andB. and Taal B. "Probabilities for a probabilistic network: A case study in esophageal cancer". In *Artificial Intelligence in Medicine* 25 (June 2002), pp. 123-148. doi: 10.1016/S0933-3657(02)00012-X.

[31] Bonnet M. *Plastics engineering. Fundamentals, processing, material selection and case studies*. Springer Fachmedien Wiesbaden, 2016. ch. chapter 4.

[32] Borth M. "Knowledge extraction on Bayes net sets". Dissertation on machine learning using Bayes nets as part of a project for Daymler Chrysler. Diss. University of Ulm. url: http://vts.uni-ulm.de/docs/2004/4366/vts_4366.pdf.

[33] Erfert M. "Extrusion blow molding, fundamentals of manufacturing processes with system analysis of machine and process". internal document. 2017.

[34] Scutari M. and Denis J.-B. *Bayesian Networks With Examples in R*. London, United Kingdom: RCR Press, Taylor and Francis Group, 2014.

[35] Scutari M., Silander T. and Ness R. *Package 'bnlearn' Bayesian Network Structure Learning, Parameter Learning and Inference*. Version 4.8-20211225. Switzerland, 25 Dec. 2021. url: https://www.bnlearn.com/.

[36] Selch M., Hänel A., Frieß U. and Ihlenfeldt S. "Quality Monitoring Of Coupled Digital Twins For Multistep Process Chains Using Bayesian Networks". In: ed. by Herberger D. and Hübner M. Proceedings of the Conference on Production Systems and Logisticsg. Institutional Repository of Leibniz Universität Hannover, 2021, pp. 415-425. doi: https://doi.org/10.15488/11266. url: https://www.repo.uni-hannover.de/handle/123456789/11353.

[37] Thielen M., Hartwig K. and Gust P. *Blow moulding of hollow plastic parts*. 2nd, updated edition. Carl Hanser Verlag Munich, 2019.

[38] Amin Md. T. "Fault Detection and Root Cause Diagnosis using Dynamic Bayesian Network." Diss. Jan. 2018.

[39] Erdelyi-Alarcon P. and Sulaimanov N. *Synthetic data in risk management*. BIGDATA INSIDER. Oct. 15, 2021. url: https://www.bigdata-insider.en/synthetic-data-im-risk-management-a-1059442/ (visited Jan. 24, 2022).

[40] Eyerer P., Hirth T. and Elsner P. *Polymer Engineering*. 1st ed. Berlin and Heidelberg: Springer Berlin Heidelberg, 2008.

[41] Kraaijeveld P., Druzdzel M., Onisko A. and Wasyluk H. "GeNIeRate: An interactive generator of diagnostic Bayesian network models". In: (July 2008).

[42] Judea Pearl. Ed. by Pearl J. San Francisco (CA): Morgan Kaufmann, 1988. isbn: 978-0-08-051489-5. doi: https://doi.org/10.1016/B978-0-08-051489-5.50016-3.

[43] Isermann R. *Fault-Diagnosis Systems: An Introduction from Fault Detection to Fault Tolerance*. Springer Berlin Heidelberg, 2005. doi: https://doi.org/10.1007/3-540-30368-5. url: https://books.google.de/books?id=6yUfoZhGMYoC.

[44] Kohlhaas R., Szekeresch F., Gindele T. and Dillmann R. "Dynamic Bayesian Network Library. A C++ framework for computations on dynamic Bayesian networks." In: SSpringer Berlin Heidelberg," 2009, p. 8. url: https://link.springer.com/content/pdf/10.1007%2F978-3-642-10284-4_30.pdf.

[45] Kruse R., Borgelt C., Braune C., Klawonn F., Moewes C. and Steinbrecher M. *Computational Intelligence. A methodological introduction to artificial neural networks, evolutionary algorithms, fuzzy systems and Bayes nets*. Magdeburg: Springer Fachmedien Wiesbaden, 2015.

[46] Neapolitan R. and Neapolitan R.E.n. *Learning Bayesian Networks*. Northeastern Illinois University, Chicago, Illinois: Pearson Prentice Hall, Jan. 2003. isbn: 9780123704771. doi: 10.1145/1327942.1327961.

[47] Paluv R. *Machine Learning in 6 steps - This is what the process chain looks like*. plus-iT. Aug 30, 2018. url: https://plus-it. en/blog/machine-learning-in-6-steps-this-is-what-the-process-chain-looks-like. the-process-chain-out/ (visited Jan. 28, 2022).

[48] Pühringer R. "Bayesian Networks for Predictive Maintenance". Diss. Vienna: Vienna University of Technology, 2018. 146 p. doi: 10.34726/hss. 2018.43900. url: https://resolver.obvsg.at/urn:nbn:at-ubtuw:1-109891.

[49] Dienst S., Ansari-Ch. F., Holland A. and Fathi M. "Necessity of using Dynamic Bayesian Networks for feedback analysis into product development". In: *2010 IEEE International Conference on Systems, Man and Cybernetics*. 2010, pp. 939-946. doi: 10.1109/ICSMC. 2010. 5641887.

[50] Flügge S., Zimmer S., and Petersohn U. *Knowledge representation and diagnostic inference using Bayesian networks in the medical discourse domain*. Dresden: Dresden University of Technology, 2019. url: https://nbn-resolving.org/urn:nbn:de:bsz:14-qucosa2-351331.

[51] Krings S. *Fundamentals of artificial intelligence*. 2016.

[52] Moecker S. "Application development on the mobile Android system for interaction with process data on a Siemens SIMATIC PLC". Bachelor Thesis. FH Düsseldorf, 2012. url: https://opus4.kobv.de/opus4-hs-duesseldorf/frontdoor/index/index/docId/562.

[53] Weise S. and Chumtong J. *AI and Europe- aspiration and reality. Results of the study Änalysis of Current Global AI Developments with Focus on Europe"*. in brief. Konrad-Adenauer-Stiftung e.V., Dec. 2020. url: https :// www . kas . de /en/ kurzum / detail/ - /content/ki-and-europe-claims-and-reality.

[54] Wu S., Zhang L., Wenpei Z., Liu Y. and Lundteigen M. "A DBN-based risk assessment model for prediction and diagnosis of offshore drilling incidents". In *Journal of Natural Gas Science and Engineering* 34 (June 2016). doi: 10.1016/j.jngse.2016.06.054.

[55] Nikolenko S. I. *Synthetic Data for Deep Learning*. Springer Optimization and Its Applications 978-3-030-75178-4. Springer, Nov. 2021. doi: 10.1007/978-3-030-75178-4.

[56] Klaeger T., Gottschall S. and Oehm L. "Data Science on Industrial Data-Today's Challenges in Brown Field Applications." In: *Challenges* 12.1 (2021). issn: 2078-1547. doi: 10.3390/challe12010002. url: https://www.mdpi.com/2078-1547/12/1/2.

[57] Yang Pang T., Xiang Yu T. and Feng Song B. "Fault diagnosis for mechanical system using dynamic Bayesian network". In: *IOP Conference Series: Materials Science and Engineering* 1043.3 (Jan. 2021), pp. 032062. doi: 10 . 1088 /1757 - 899x /1043 /3/032062. url: https://doi.org/10.1088/1757-899x/1043/3/032062.

[58] D2000 Dev Team. *Siemens SIMATIC S7 ISO on TCP*. Feb 15, 2021. url: https :// doc. ipesoft. com/display/D2DOCV12EN/Siemens+SIMATIC+S7+ISO+on+TCP (visited Jan 21, 2022).

[59] Bruder U. *Plastics technology made easy. Materials - processing - tool design - cost calculation - post-processing - joining methods - material selection - design rules. - Process optimization - troubleshooting*. Munich: Carl Hanser Verlag GmbH Co. KG, 2016.

[60] Kjærulff U. B. and Madsen A. L. *Bayesian Networks and Influence Diagrams: A Guide to Construction and Analysis*. Springer New York Heidelberg Dordrecht London, 2013. doi: https://doi.org/10.1007/978-1-4614-5104-4.

[61] Yin Wee Y., Ping Cheah W., Tan S. C. and Wee K. "A method for root cause analysis with a Bayesian belief network and fuzzy cognitive map". In: *Expert Systems with Applications* 42.1 (2015), pp. 468-487. issn: 0957-4174. doi: https://doi.org/10.1016/j.eswa.2014.06.037. url: https://www.sciencedirect.com/science/article/pii/S0957417414003789.